SIMULIA

3DEXPERIENCE

ABAQUS 2026 FD04
INTRODUCTION & SPATIAL MODELING GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....5
Abaqus Introduction & Spatial Modeling Guide....6
What's New....7
Introduction to Abaqus....8
Overview of the Abaqus Finite Element System....8
Abaqus Products....8
Analysis....8
Preprocessing and Postprocessing....9
Add-on Analysis....9
Optional Analysis Functionality....10
Associative Interfaces and Geometry Translators....10
Translator Utilities....10
Other Utilities....10
Documentation....10
Abaqus Documentation....10
Demonstrating Abaqus Capabilities....11
Reviewing the Results of an Abaqus Simulation....11
Input Syntax Rules....12
Keyword Lines....12
Data Lines....14
Sets....14
Labels....15
Repeating Data Lines....16
Conventions....18
Degrees of Freedom....18
Axisymmetric Elements....19
Electromagnetic Elements....19
Activation of Degrees of Freedom....19
Internal Variables in Abaqus/Standard....19
Coordinate Systems....19
Units....20
Rotation and Angle Measures....20
International System of Units (SI)....20
American or English Units....20
Symbols Used in Abaqus for Units....21
Defining a Unit System for Translating Data with Other Solvers....21
Time Measures....24
Local Tangent Directions on Surfaces in Space....24
Rotation of the Local Directions....25
Convention Used for Stress and Strain Components....26
Nonisotropic Material Behavior....26
Zero-Valued Stress Components....27
Shear Strains....27

Stress and Strain Measures....27
Total (Integrated) Strain....27
Green's Strain....28
Nominal Strain....28
Logarithmic Strain....28
Stress Invariants....28
Finite Rotations....29
Compound Rotations....30
Abaqus Model Definition....34
The Input File....34
Model Data....35
Required Model Data....35
Optional Model Data....35
History Data....36
Required History Data....37
Optional History Data....37
Including Model or History Data from an External File....38
Including an Encrypted Data File....38
Parametric Modeling....40
Introduction....40
Parameters....41
Independent Parameters....41
Dependent Parameters....41
Rules for Parameters....42
Parametrization of Input Quantities....44
Parametrizing Individual Input Quantities....44
Parametrizing Groups of Input Quantities (Expressional Dependence)....44
Parametrizing Groups of Input Quantities (Tabular Dependence)....45
Python Language....46
Statement Length and Continuation Lines....46
Comments....46
Parameter Names....46
Data Types....46
Type Conversion....47
Numeric Operators....47
Functions....48
Character String Operators....49
Execution of Parametrized Input....49
Parameter Check Jobs....49
Display of Parametric Input....49
Data File Display....50
References....50
Spatial Modeling....51
Node Definition....52
Node Definition....53
Parametric Shape Variation....76
Nodal Thicknesses....81

Normal Definitions at Nodes....84
Transformed Coordinate Systems....87
Adjusting Nodal Coordinates....91
Element Definition....93
Element Definition....94
Element Foundations....106
Defining Reinforcement....107
Defining Rebar as an Element Property....119
Orientations....140
Surface Definition....153
About Surfaces....154
Element-Based Surface Definition....157
Node-Based Surface Definition....173
Analytical Rigid Surface Definition....175
Eulerian Surface Definition....186
Operating on Surfaces....188
Rigid Body Definition....190
What Is a Rigid Body?....190
Determining When to Use a Rigid Body....191
Creating a Rigid Body....191
Rigid Body Mass and Inertial Properties....196
Kinematics of a Rigid Body....197
Defining Loads on Rigid Bodies....200
Rigid Bodies with Temperature Degrees of Freedom....201
Thermal Expansion of a Rigid Body....202
Modeling Contact with a Rigid Body....202
Using Rigid Bodies in Geometrically Linear Abaqus/Standard Analysis....203
Integrated Output Section Definition....204
Introduction....204
Creating an Integrated Output Section....205
Mass Adjustment....209
Defining Mass Adjustment....209
Adjusting the Total Mass of an Element Set to a Known Value....209
Redistribution of Mass to Raise the Minimum Stable Time Increment to a Target Value....210
Redistribution of Mass Using a Scale Factor....211
Nonstructural Mass Definition....213
Nonstructural Mass....213
Defining Nonstructural Mass....214
Distribution Definition....217
Distributions....218
Specifying the Location of a Distribution....218
Defining a Distribution Table....219
Defining Distributions by Importing Field Data from an Output Database File....220
Applying Distributions....221
Examples....221
Display Body Definition....225
What Is a Display Body?....225
Creating a Display Body....225

Using Display Bodies with Connectors....226
Input File Template....227
Assembly Definition....228
Introduction....228
Defining Parts, Part Instances, and the Assembly....229
Organizing the Model Definition....232
Rules for Defining an Assembly....232
Coordinate System Definitions....239
Limitations....241
Input File Template....242
Probability Density Function....243
Introduction....243
Applications....243
Probability Density Function....243
Truncated Probability Density Function....248
Output....249
Limitations....249
Input File Template....249
References....250

Trademarks and Legal Notices

Trademarks

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

Legal Notices

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

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

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

© Dassault Systèmes Americas Corp., 2026.

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

Abaqus Introduction & Spatial Modeling Guide

The Abaqus Introduction & Spatial Modeling Guide discusses an overview of the Abaqus finite element system and the basic modeling concepts in Abaqus, such as defining nodes, elements, and surfaces, and the conventions and input formats that should be followed when using Abaqus.

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.

Abaqus/Standard and Abaqus/Explicit can be run as batch applications (see *Abaqus/Standard and Abaqus/Explicit Execution* for details) or through the interactive Abaqus/CAE environment (see *Abaqus/CAE Execution* for details on how to start Abaqus/CAE). The main input to the Abaqus/Standard and Abaqus/Explicit analysis products is a file containing the options required for the simulation and the data associated with those options. There may also be supplementary files, such as restart or results files from previous analyses, or auxiliary data files, such as a file containing an acceleration record or an earthquake record for dynamic analysis. The input file is usually created by Abaqus/CAE or another preprocessor. Both input file usage and Abaqus/CAE usage information are provided.

As described in *Abaqus Model Definition*, the main input file consists of two sections: model input and history input. The input is organized around a few natural concepts and conventions, which means that even though input files for complex simulations can be large, they can be managed without difficulty. The basic syntax rules that govern an Abaqus input file are discussed in *Input Syntax Rules*. The *Abaqus Keywords Guide* contains a complete description of all the input options available in Abaqus/Standard and Abaqus/Explicit. For a detailed introduction to using Abaqus for your analyses, it is recommended that you follow the self-paced tutorials in *Getting Started with Abaqus/CAE*. Refer to the *Abaqus/CAE User's Guide* for detailed information on working with Abaqus/CAE.

What's New

This page describes recent changes in Introduction & Spatial Modeling.

2026 FD04

Modeling Surface Tension Loads with Eulerian Material Surfaces

You can now specify surface tension loads on Eulerian material surfaces in Abaqus/Explicit.

Benefits: You can model large deformation under surface tension loads.

For more information, see *Eulerian Surface Definition*, *Surface Tension Loads*, and **DSLOAD*.

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 *Abaqus Introduction & Spatial Modeling Guide*.

2024 FD01

Generating a Node Set from a Surface

You can specify the name of a previously defined surface, in which case the nodes included in the surface will be assigned to the specified node set. This method can be used only to define sorted node sets.

Benefits: This surface-based node set feature makes it easier to create a node set that includes all the nodes on a predefined surface.

For more information, see *About Surfaces* and *Generating a Node Set from a Surface*.

Introduction to Abaqus

Overview

The capabilities of the Abaqus finite element analysis technology have been proven to be reliable, robust, and accurate for many years in a variety of industries for mission-critical applications.

References:

- *Execution Procedures*

Overview of the Abaqus Finite Element System

The Abaqus finite element system includes:

- Abaqus/Standard, a general-purpose finite element program;
- Abaqus/Explicit, an explicit dynamics finite element program;
- Abaqus/CAE, an interactive environment used to create finite element models, submit Abaqus analyses, monitor and diagnose jobs, and evaluate results; and
- Abaqus/Viewer, a subset of Abaqus/CAE that contains only the postprocessing capabilities of the Visualization module.

Several add-on 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/Aqua contains optional features that are specifically designed for the analysis of beam-like structures installed underwater and subject to loading by water currents and wave action. The Abaqus/Design option enables you to perform design sensitivity analysis (DSA). Abaqus/AMS is an optional eigensolver that works within Abaqus/Standard providing very fast solution of large symmetric eigenvalue problems. The Abaqus co-simulation technique provides several applications, available as separate add-on capabilities, for coupling between Abaqus and third-party analysis programs. Abaqus/Foundation is an optional subset of Abaqus/Standard that provides more cost-efficient access to the linear static and dynamic analysis functionality in Abaqus/Standard. These options are available only if your license includes them.

Abaqus Products

Analysis

Abaqus/Standard

This general-purpose analysis product can solve a wide range of linear and nonlinear problems involving the static, dynamic, thermal, electrical, and electromagnetic response of components. Abaqus/Standard

includes all analysis capabilities except those provided in the Abaqus/Explicit program and the add-on analysis functionality described below.

Abaqus/Explicit

This product provides nonlinear, transient, dynamic analysis of solids and structures using explicit time integration. Its powerful contact capabilities, reliability, and computational efficiency on large models also make it highly effective for quasi-static applications involving discontinuous nonlinear behavior.

Preprocessing and Postprocessing

Abaqus/CAE

This product is a Complete Abaqus Environment that provides a simple, consistent interface for creating, submitting, monitoring, 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, generating a mesh, submitting analysis jobs, and interpreting results.

Abaqus/Viewer

This subset of Abaqus/CAE contains only the postprocessing capabilities of the Visualization module. It uses the output database (`.odb`) to obtain results from the analysis products. The output database is a neutral binary file. Therefore, results from an Abaqus analysis run on any platform can be viewed on any other platform supporting Abaqus/Viewer. It provides deformed configuration, contour, vector, and *X–Y* plots, as well as animation of results.

Add-on Analysis

Abaqus/AMS

This add-on analysis capability for Abaqus/Standard allows the user to select the automatic multi-level substructuring (AMS) eigensolver when performing a natural frequency extraction.

Abaqus/Aqua

This add-on analysis capability for Abaqus/Standard and Abaqus/Explicit provides a capability for calculating drag and buoyancy loads based on steady current, wave, and wind effects for modeling offshore piping and floating platform structures. Abaqus/Aqua is applicable for structures that can be idealized using line elements, including beam, pipe, and truss elements.

Abaqus/Design

This add-on analysis capability for Abaqus/Standard allows the user to perform design sensitivity analysis (DSA). The derivatives of output variables are calculated with respect to specified design parameters.

Optimization module

This capability is available in Abaqus/CAE to perform shape and topology optimization. This functionality requires an additional license to submit an optimization process for analysis.

Abaqus/Foundation

This analysis option offers more efficient access to the linear static and dynamic analysis functionality in Abaqus/Standard.

CZone for Abaqus

This add-on capability for Abaqus/Explicit provides access to a state-of-the-art methodology for crush simulation based on CZone technology from Engenuity, Ltd. Targeted toward the design of composite components and assemblies, CZone for Abaqus provides for inclusion of material crush behavior in simulations of composite structures subjected to impact. This functionality requires an additional license.

Optional Analysis Functionality

Co-simulation with MpCCI

This add-on analysis capability for Abaqus can be used to solve multiphysics problems by coupling Abaqus with any third-party analysis program that supports the MpCCI interface.

Associative Interfaces and Geometry Translators

3DEXPERIENCE Associative Interface

This add-on capability for Abaqus/CAE creates a connection between a **3D**EXPERIENCE session and an Abaqus/CAE session. This connection can be used to transfer model information from **3D**EXPERIENCE to Abaqus/CAE. Subsequent modifications to the model in **3D**EXPERIENCE can be propagated to the Abaqus/CAE model while retaining any analysis features (such as loads or boundary conditions) that were defined on the model in Abaqus/CAE. The **3D**EXPERIENCE model in an assembly file (`.eaf`) format can also be imported directly into Abaqus/CAE.

Translator Utilities

Abaqus translators are provided with the release. They are invoked through the Abaqus execution procedure. The translators and the commands to invoke them are described in *Execution Procedures*.

Other Utilities

Additional programs are included with the release. They are all invoked through the Abaqus execution procedure. The utilities and the commands to invoke these programs are described in *Execution Procedures*.

Documentation

Abaqus Documentation

Abaqus documentation is available in several locations.

- You can install the HTML documentation locally (on a server at your site). An improved search capability using Exalead CloudView is available. See the SIMULIA Installation Guide for more information.

- A user assistance portal is available at *https://help.3ds.com/* that uses EXALEAD-based search; supports viewing in Chrome and Firefox; and offers the ability to refine your search results (for example, to a specific guide).
- You can download PDF versions of the Abaqus guides from the guide overviews.

The Abaqus guides provide a reference to using Abaqus/Standard (including Abaqus/Aqua, Abaqus/Design, and Abaqus/Foundation) and Abaqus/Explicit (including Abaqus/Aqua). 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).

Throughout the guides the term Abaqus is most commonly used to refer to both Abaqus/Standard and Abaqus/Explicit; the individual product names are used to indicate when information applies to only that product. Product identifiers appear at the beginning of sections in many guides indicating the products to which the information in the section applies.

Demonstrating Abaqus Capabilities

Many analyses that demonstrate the numerous capabilities of Abaqus are discussed in the *Abaqus Example Problems Guide*, the *Abaqus Benchmarks Guide*, and the *Abaqus Verification Guide*. These examples can help you become familiar with the functionality that Abaqus provides and the structure of the Abaqus input file. For example, *Beam impact on cylinder* discusses the various modeling techniques that can be used to analyze the dynamic response of a cantilever beam.

Reviewing the Results of an Abaqus Simulation

Information on requesting output from an Abaqus simulation is discussed in *About Output*. Requested results from an Abaqus simulation are viewed through the Visualization module in Abaqus/CAE (also licensed separately as Abaqus/Viewer). The output database file is read by the Visualization module in Abaqus/CAE to create contour plots, animations, *X*–*Y* plots, and tabular output of Abaqus results. See *Viewing results* for detailed information on using the Visualization module in Abaqus/CAE.

Input Syntax Rules

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Abaqus Model Definition*

Overview

This section describes the syntax rules that govern an Abaqus input file.

All data definitions in Abaqus are accomplished with option blocks—sets of data describing a part of the problem definition. You choose those options that are relevant for a particular application. Options are defined by lines in the input file. Three types of input lines are used in an Abaqus input file: *keyword* lines, *data* lines, and *comment* lines. Only 7-bit ASCII characters are supported in keyword lines and data lines, and a line feed is required at the end of each line in an input file.

- Keyword lines introduce options and often have *parameters*, which appear as words or phrases separated by commas on the keyword line. Parameters are used to define the behavior of an option. Parameters can stand alone or have a value, and they may be required or optional.
- Data lines, which are used to provide numeric or alphanumeric entries, follow most keyword lines.
- Any line that begins with stars in columns 1 and 2 (**) is a comment line. Such lines can be placed anywhere in the file. They are ignored by Abaqus, so they will be printed only in the initial listing of the file. There is no restriction on how many or where such lines occur in the file.

Relevant parameters and data lines (including the number of entries per data line) are described in the sections of the *Abaqus Keywords Guide* describing each option. This section describes the general rules that apply to all keyword and data lines.

Keyword Lines

The following rules apply when entering a keyword line:

- The first non-blank character of each keyword line must be a star (*).
- The keyword must be followed by a comma (,) if any parameters are given.
- Parameters must be separated by commas.
- Blanks on a keyword line are ignored.
- A line can include no more than 256 characters, including blanks. There are additional limitations when encrypting an input file (see *Encrypting and Decrypting Abaqus Input Data*).
- Keywords and parameters are not case sensitive.
- Parameter values usually are not case sensitive. The only exceptions to this rule are those imposed externally to Abaqus, such as file names on case-sensitive operating systems.

- Keywords, parameters, and, in most cases, parameter values need not be spelled out completely, but there must be enough characters given to distinguish them from other keywords, parameters, and parameter values that begin in the same way. Abaqus first searches each associated text string for an exact match. If an exact match is not found, Abaqus then searches based upon the minimum number of unique characters in each keyword, parameter, or parameter value, as the case may be. Embedded blanks can be omitted from any item in a keyword line. If a parameter value is used to provide a number or a file name, the complete value should be provided.
- If a parameter has a value, the equal sign (=) is used. The value can be an integer, a floating point number, or a character string, depending on the context. For example,

  ```
  *ELASTIC, TYPE=ISOTROPIC, DEPENDENCIES=1
  ```

- When the parameter value is a character string that represents the name of an item, you should not use case as a method of distinguishing values unless the values are enclosed within quotation marks. For example, Abaqus does not distinguish between the following definitions:

  ```
  *MATERIAL, NAME=STEEL
  *MATERIAL, NAME=Steel
  ```

- The same parameter should not appear more than once on a single keyword line. If a parameter has multiple settings on a single keyword line, Abaqus ignores all but one of the settings.
- Continuation of a keyword line is sometimes necessary; for example, because of a large number of parameters. If the last character on a keyword line is a comma, the next line is interpreted as a continuation of the line. For example, the **ELASTIC* keyword line above could also be given as

  ```
  *ELASTIC, TYPE=ISOTROPIC,
  DEPENDENCIES=1
  ```

- Certain keywords must be used in conjunction with other keywords; for example, the **ELASTIC* and **DENSITY* keywords must be used in conjunction with the **MATERIAL* keyword. These related keywords must be grouped in a block in the input file; unrelated keywords cannot be specified within this block.
- Some options allow the INPUT or FILE parameter to be set equal to the name of an alternate file. Such file names can include a full path name or a relative path name. Relative path names must be with respect to the directory from which the job was submitted. If no path is specified, the file is assumed to be in the directory from which the job was submitted. A substructure database must be in the same directory from which the job was submitted; a full path name cannot be used to specify a substructure database name.

  For files referenced by the INPUT parameter, the file name must include any extension (for example, `elem.inp`). For files referenced by the FILE parameter, the name must be given without an extension in most cases since Abaqus assumes that the file to be read has the correct extension for the file type that is relevant to the option: `.res` for restart files (*Restarting an Analysis*) and `.fil` for results files (*About Output*). However, special rules may apply when a results file (`.fil`) or an output database file (`.odb`) is relevant for the option (see *Initial Conditions* and *Sequentially Coupled Thermal-Stress Analysis* for details).

  The import and part instance options allow the LIBRARY parameter to be set to a value that specifies the previous analysis from which to import the element sets or instance (see **IMPORT* and **INSTANCE*).

  The file or library name must have the correct case on computers with case-sensitive operating systems. Regardless of whether the user specifies only a file name, a relative path name, or a full path name, the complete name including the path can have a maximum of 256 characters. All spaces within a file name, a relative path name, or a full path name are ignored unless the name is enclosed in quotation marks, in which case all spaces within the name are maintained.

Data Lines

Data lines are used to provide data that are more easily given in lists than as parameters on an option. Most options require one or more data lines; if they are required, the data lines must immediately follow the keyword line introducing the option. The following rules apply when entering a data line:

- A data line can include no more than 256 characters, including blanks. Trailing blanks are ignored.
- All data items must be separated by commas (,). An empty data field is specified by omitting data between commas. Abaqus will use values of zero for any required numeric data that are omitted unless a default value is specified.
- A line must contain only the number of items specified. Abaqus ignores extra items on a line, and no warning message is issued.
- Empty data fields at the end of a line can be ignored.
- Floating point numbers can be given with or without an exponent. Any exponent, if input, must be preceded by E or D and an optional (−) or (+). The following line shows four acceptable ways of entering the same floating point number:

  ```
  -12.345    -1234.5E-2   -1234.5D-2   -1.2345E1
  ```

- Integer data items can occupy a maximum of 9 digits.
- Character strings can be up to 80 characters long and are not case sensitive.
- Continuation lines are allowed in specific instances (see *Element Definition*). If allowed, such lines are indicated by a comma as the last character of the preceding line. A single data item cannot be entered over multiple lines.

In many cases the choice of parameters used with an option determines the type of data lines required. For example, there are five different ways to define a linear elastic material (*Elastic Behavior*). The data lines you specify must be consistent with the value of the TYPE parameter given on the **ELASTIC* option.

Sets

One of the most useful features of the Abaqus data definition method is the availability of *sets*. A set can be a set of nodes or a set of elements. You provide a name (1–80 characters, the first of which must be a letter) for each set. That name then provides a means of referencing all of the members of the set. As an example suppose that, for the structure shown in *Figure 1*, we wish to apply symmetry boundary conditions at all of the nodes in the set `MIDDLE` and that the edge `SUPPORT` is pinned.

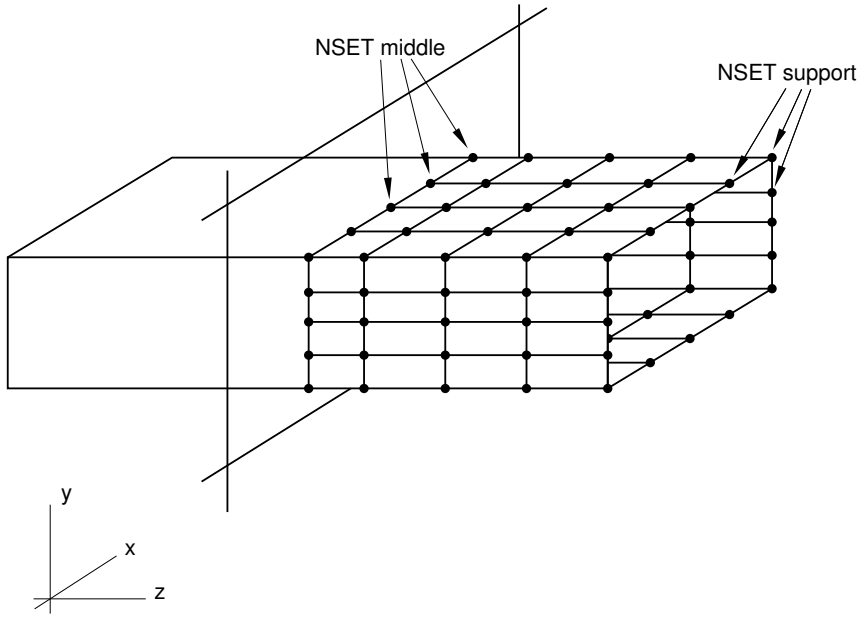


Figure 1: Example of the use of sets.

We assemble the relevant nodes into sets and specify the boundary conditions by

```
*BOUNDARY
 MIDDLE, ZSYMM
 SUPPORT, PINNED
```

Sets are the basic reference throughout Abaqus, and the use of sets is recommended. Choosing meaningful set names makes it simple to identify which data belong to which part of the model. Further discussion of sets is provided in *Node Definition* and *Element Definition*.

Labels

Labels such as set names, surface names, and rebar names are case insensitive unless enclosed within quotation marks (except when they are accessed from user subroutines; see *About User Subroutines and Utilities*). Labels can be up to 80 characters long. All spaces within a label are ignored unless the label is enclosed in quotation marks, in which case all spaces within the label are maintained. A label that is not enclosed within quotation marks must begin with a letter, may not include a period (.), and should not contain characters such as commas and equal signs. These restrictions do not apply to labels enclosed within quotation marks except if the label is a material name. A material name must always start with a letter, even if the name is enclosed within quotation marks.

Labels cannot begin and end with a double underscore (e.g., __STEEL__). This label format is reserved for internal use by Abaqus.

The following are examples of labels entered with and without the use of quotation marks:

```
*ELEMENT, TYPE=SPRINGA, ELSET="One element"
1,1,2
*SPRING, ELSET="One element"
1.0E-5,
*NSET, ELSET="One element", NSET=NODESET
*BOUNDARY
nodeset,1,2
```

Repeating Data Lines

Some options list only a single data line. In cases where only one data line is allowed, this is indicated by the data line title “First (and only) line.” An example of this is the **DYNAMIC* option. In many cases the single data line shown can be repeated to define one variable as a function of another; this choice is indicated by a note after the data line. For example, a table of biaxial test data can be given to define a hyperelastic material:

```
*BIAXIAL TEST DATA
T_B^1, ε_B^1
T_B^2, ε_B^2
T_B^3, ε_B^3
Etc.
```

There is no limit on the number of data lines allowed, but the data must be given in a certain order, as explained below.

Many options require more than one data line; these are indicated by the data line titles “First line:”, “Second line:”, etc. For example, exactly two data lines must be used to define a local orientation for a shell element (**ORIENTATION*), and at least three data lines are required to define anisotropic elasticity (**ELASTIC*).

In many cases the data lines can be repeated, which is indicated by a note after the data lines. As with repetition of a single data line, it is important that sets of data lines be given in the correct order so that Abaqus can interpolate the data properly.

Example: Multiple Data Lines due to Field Variable Dependence

Any time an option can be defined as a function of field variables, you must determine the number of data lines required to define the option completely. (See *Specifying Field Variable Dependence* for more information.) For example, two data lines are required if stress-based failure criteria (**FAIL STRESS*) are defined as a function of two field variables. This pair of data lines is repeated as often as necessary to define the failure criteria completely:

```
*FAIL STRESS, DEPENDENCIES=2
```

first pair:
$X_t^1,\ X_c^1,\ Y_t^1,\ Y_c^1,\ S^1,\ ,\ \sigma_{biax}^1$
$fv_1^1,\ fv_2^1$

second pair:
$X_t^2,\ X_c^2,\ Y_t^2,\ Y_c^2,\ S^2,\ ,\ \sigma_{biax}^2$
$fv_1^2,\ fv_2^2$

third pair:
$X_t^3,\ X_c^3,\ Y_t^3,\ Y_c^3,\ S^3,\ ,\ \sigma_{biax}^3$
$fv_1^3,\ fv_2^3$

```
Etc.
```

(In this example the last field on the first data line of each pair was omitted, which means that the stress-based failure criteria are not temperature dependent.)

If the stress-based failure criteria were defined as a function of nine field variables, a set of three data lines would be repeated as often as necessary:

```
*FAIL STRESS, DEPENDENCIES=9
```

first set

$$\begin{cases} X_t^1,\ X_c^1,\ Y_t^1,\ Y_c^1,\ S^1,\ \ ,\ \sigma_{biax}^1 \\ fv_1^1, fv_2^1, fv_3^1, fv_4^1, fv_5^1, fv_6^1, fv_7^1, fv_8^1 \\ fv_9^1 \end{cases}$$

second set

$$\begin{cases} X_t^2,\ X_c^2,\ Y_t^2,\ Y_c^2,\ S^2,\ \ ,\ \sigma_{biax}^2 \\ fv_1^2, fv_2^2, fv_3^2, fv_4^2, fv_5^2, fv_6^2, fv_7^2, fv_8^2 \\ fv_9^2 \end{cases}$$

```
Etc.
```

Ordering the Data Lines

Whenever one variable is defined as a function of another, the data must be given in the proper order so that Abaqus can interpolate for intermediate values correctly. The variable being defined is assumed to be constant outside the range of independent variables given, except for nonlinear elastic gasket thickness behavior involving damage where the data are extrapolated based on the last slope computed from the user-specified data.

If the property being defined is a function of only one variable (such as the **BIAXIAL TEST DATA* shown above), the data should be given in the order of increasing value of the independent variable.

If the property being defined is a function of multiple independent variables, the variation of the property with respect to the first variable must be given at fixed values of the other variables, in ascending values of the second variable, then of the third variable, and so on. The data lines must always be ordered so that the independent variables are given increasing values. This process ensures that the value of the material property is completely and uniquely defined at any values of the independent variables upon which the property depends.

As an example, consider isotropic elasticity defined as a function of three field variables (but not of temperature):

```
*ELASTIC, DEPENDENCIES=3
E1, ν1,  , 1, 1, 1
E2, ν2,  , 2, 1, 1
E3, ν3,  , 1, 2, 1
E4, ν4,  , 2, 2, 1
E5, ν5,  , 1, 3, 1
E6, ν6,  , 2, 3, 1
E7, ν7,  , 1, 1, 2
E8, ν8,  , 2, 1, 2
E9, ν9,  , 1, 2, 2
E10, ν10, , 2, 2, 2
E11, ν11, , 1, 3, 2
E12, ν12, , 2, 3, 2
E13, ν13, , 1, 1, 3
E14, ν14, , 2, 1, 3
E15, ν15, , 1, 2, 3
E16, ν16, , 2, 2, 3
E17, ν17, , 1, 3, 3
E18, ν18, , 2, 3, 3
```

Conventions

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Spatial Modeling*
- *About Output*
- *Boundary Conditions*

Overview

The conventions that are used throughout Abaqus are defined in this section.

Degrees of Freedom

Except for axisymmetric elements and electromagnetic elements, the degrees of freedom are always referred to as follows:

1	*x*-displacement
2	*y*-displacement
3	*z*-displacement
4	Rotation about the *x*-axis, in radians
5	Rotation about the *y*-axis, in radians
6	Rotation about the *z*-axis, in radians
7	Warping amplitude (for open-section beam elements)
8	Pore pressure, hydrostatic fluid pressure, or acoustic pressure
9	Electric potential
10	Connector material flow (units of length)
11	Temperature (or normalized concentration in mass diffusion analysis)
12	Second temperature (for shells or beams)
13	Third temperature (for shells or beams)
14	Etc.
30	20th temperature (for shells or beams)
31	Slurry concentration
32	Electric potential in the electrolyte
33	Ion concentration in the electrolyte
34	Species concentration (for solid electrolytes and solid-state batteries)

Here the x-, y-, and z-directions coincide with the global X-, Y-, and Z-directions, respectively; however, if a local transformation is defined at a node (see *Transformed Coordinate Systems*), they coincide with the local directions defined by the transformation.

A maximum of 20 temperature values (degrees of freedom 11 through 30) can be defined for shell or beam elements in Abaqus/Standard.

Axisymmetric Elements

The displacement and rotation degrees of freedom in axisymmetric elements are referred to as follows:

1	r-displacement
2	z-displacement
5	Rotation about the z-axis (for axisymmetric elements with twist), in radians
6	Rotation in the r–z plane (for axisymmetric shells), in radians

Here the r- and z-directions coincide with the global X- and Y-directions, respectively; however, if a local transformation is defined at a node (see *Transformed Coordinate Systems*), they coincide with the local directions defined by the transformation.

Electromagnetic Elements

Electromagnetic elements in Abaqus/Standard are used to define the element shape and to discretize the continuum. The eddy current and magnetostatic analyses formulations use magnetic vector potential as a degree of freedom (see *Boundary Conditions* and *Boundary Conditions*).

Activation of Degrees of Freedom

Abaqus/Standard and Abaqus/Explicit activate only those degrees of freedom needed at a node. Therefore, some of the degrees of freedom listed above might not be used at all nodes in a model, because each element type uses only those degrees of freedom that are relevant. For example, two-dimensional solid (continuum) stress/displacement elements use only degrees of freedom 1 and 2. The degrees of freedom actually used at any node are the envelope of those needed in each element that shares the node.

Internal Variables in Abaqus/Standard

In addition to the degrees of freedom listed above, Abaqus/Standard uses internal variables (such as Lagrange multipliers to impose constraints) for some elements. Normally you need not be concerned with these variables, but they may appear in error and warning messages and are checked for satisfaction of nonlinear constraints during iteration. Internal variables are always associated with internal nodes, which have negative numbers to distinguish them from user-defined nodes.

Coordinate Systems

The basic coordinate system in Abaqus is a right-handed, rectangular Cartesian system. You can choose other systems locally for input (see *Node Definition*), for output of nodal variables (displacements, velocities, etc.) and point load or boundary condition specification (see *Transformed Coordinate Systems*), and for material or kinematic joint specification (see *Orientations*). All coordinate systems must be right-handed.

Units

Abaqus has no units built into it except for rotation and angle measures. Therefore, the units chosen must be self-consistent, which means that derived units of the chosen system can be expressed in terms of the fundamental units without conversion factors.

Rotation and Angle Measures

In Abaqus rotational degrees of freedom are expressed in radians, and all other angle measures are expressed in degrees (for example, phase angles).

International System of Units (SI)

The International System of units (SI) is an example of a self-consistent set of units. The fundamental units in the SI system are length in meters (m), mass in kilograms (kg), time in seconds (s), temperature in degrees kelvin (K), and electric current in amperes (A). The units of secondary or derived quantities are based on these fundamental units. An example of a derived unit is the unit of force. A unit of force in the SI system is called a newton (N):

$$1 \text{ newton} = 1 \text{ kg m/s}^2.$$

Similarly, a unit of electrical charge in the SI system is called a coulomb (C):

$$1 \text{ coulomb} = 1 \text{ A s}.$$

Another example is the unit of energy, called a joule (J):

$$1 \text{ joule} = 1 \text{ N m} = 1 \text{ A volt s} = 1 \text{ kg m}^2/\text{s}^2.$$

The unit of electrical potential in the SI system is the volt, which is chosen such that

$$1 \text{ joule} = 1 \text{ volt C} = 1 \text{ volt A s}.$$

Sometimes the standard units are not convenient to work with. For example, Young's modulus is frequently specified in terms of megapascals (MPa) (or, equivalently, N/mm^2), where 1 pascal = 1 N/m^2. In this case the fundamental units could be tonnes (1 tonne = 1000 kilograms), millimeters, and seconds.

American or English Units

American or English units can cause confusion since the naming conventions are not as clear as in the SI system. For example, 1 pound force (lbf) gives 1 pound mass (lbm) an acceleration of g ft/sec^2, where g is the value of acceleration due to gravity. If pounds force, feet (ft), and seconds are taken as fundamental units, the derived unit of mass is lbf sec^2/ft. Since density is commonly given in handbooks as lbm/in^3, it must be converted to lbf sec^2/ft^4 by

$$1 \text{ lbm/in}^3 = \frac{12^3}{g} \text{ lbf sec}^2/\text{ft}^4.$$

Frequently it is not made clear in handbooks whether lb stands for lbm or lbf. You need to check that the values used make up a consistent set of units.

Two other units that cause difficulty are the slug, defined as the mass that is accelerated at 1 ft/sec^2 by 1 lbf, and the poundal, defined as the force required to accelerate 1 lbm at 1 ft/sec^2. Useful conversions are

$$1 \text{ slug} = g \text{ lbm}$$

and

$$1 \text{ lbf} = g \text{ poundals},$$

where g is the magnitude of the acceleration due to gravity in ft/sec^2.

Symbols Used in Abaqus for Units

Units are indicated for the value to be given on load and flux types as follows:

Dimension	Indicator	Example (S.I. units)
length	L	meter
mass	M	kilogram
time	T	second
temperature	θ	degree Celsius
electric current	A	ampere
force	F	newton
energy	J	joule
electric charge	C	coulomb
electric potential	φ	volt
mass concentration	P	Parts per million
amount of substance	n	mol

Defining a Unit System for Translating Data with Other Solvers

In Abaqus you can specify a unit system in the model to use when translating data with other solvers. This unit system is used in the matrix generation and substructure generation procedures to create a Simpack Flexible Body Interface (FBI) file, for importing fields from other solvers in a sequentially coupled multiphysics analysis, and for co-simulation between Abaqus and other solvers. The unit system definition has no effect on the Abaqus simulation; you must ensure that a consistent set of units is used.

In a matrix generation procedure, the unit system is stored on the binary SIM file containing the generated matrices. In a substructure generation procedure, the unit system is stored on the binary SIM file containing the substructure.

If you generate flexible body entities from an Abaqus/Standard substructure for the Simpack flexible body dynamics solver, you must specify units. Two approaches are available. In the first approach you specify a unit system in the Abaqus/Standard model directly. The specified unit system is not used during the substructure generation procedure. However, the unit information is stored with the substructure and is accessed during the creation of the flexible body. In the second approach (in which you do not specify the unit system), you must run the **abaqus tosimpack** translator in stand-alone mode and define the model units on the command line (see *Translating an Abaqus Substructure to a Simpack Flexible Body*).

When importing external fields or transferring data in a co-simulation between Abaqus and another solver, unit conversion is available if the solver supports unit conversion through the SIMULIA Co-Simulation Services. If the solver does not support unit conversion or Abaqus does not specify a unit system, consistent units between all solvers are expected.

There are several ways to specify the unit system:

- You can specify a standardized unit system; for example ,SI for the International System of units, MKS for meter-kilogram-second, GCS for centimeter-gram-second, and FPS for foot-pound-second.
- You can specify the predefined unit conversion factors by specifying the physical dimension (for example, length, mass, force, time, temperature, electric current, amount substance, and luminosity) and a unit symbol

listed in the table below. The unit symbol defines the conversion factor between the Abaqus model and the SI system. For example, if the Abaqus model length unit is mm, the primary conversion factor is 0.001.

To use units that are not defined in the list of unit symbols (such as the angstrom unit), you can specify the SI unit conversion factor directly.

When translating temperature, you must specify the absolute zero temperature on the chosen temperature scale, in addition to the temperature conversion factor. The absolute zero temperature is defined by specifying physical constants.

The MKS, GCS, and FPS unit systems should be used only for mechanical analysis since they do not define quantities such as temperature. When defining units of a mechanical system, you must specify one of the basic triples: *length-mass-time* or *length-force-time*. These two methods are mutually exclusive. If you specify *length-mass-time*, the force unit is defined implicitly. If you specify *length-force-time*, the mass unit is defined implicitly.

Table 1: Length unit symbols.

Unit Name	Unit Symbol	Conversion Factor
Meter (default)	m	1.0
Centimeter	cm	1.0E-2
Millimeter	mm	1.0E-3
Kilometer	km	1.0E+3
Inch	in	0.0254
Foot	ft	0.3048
Yard	yd	0.9144
Mile	mi	1609.344

Table 2: Mass unit symbols.

Unit Name	Unit Symbol	Conversion Factor
Kilogram (default)	kg	1.0
Gram	g	1.0E-3
Tonne (metric ton)	t	1.0E+3
Pound	lb	0.45359237
Kilopound	klb	453.59237
Ounce	oz	0.0283495
Slug	slug	14.593903
Slinch (dozen slug)	slinch	175.126835246476
US ton (short ton)	uton	907.185

Table 3: Time unit symbols.

Unit Name	Unit Symbol	Conversion Factor
Second (default)	s	1.0
Millisecond	ms	1.0E-3
Hour	h	3600.0

Unit Name	Unit Symbol	Conversion Factor
Minute	min	60.0
Day	d	86400.0

Table 4: Temperature unit symbols.

Unit Name	Unit Symbol	Conversion Factor
Kelvin (default)	K	1.0
Celcius	C	1.0
Farenheit	F	0.5555556

Table 5: Current unit symbols.

Unit Name	Unit Symbol	Conversion Factor
Ampere (default)	A	1.0
milli-Ampere	mA	1.0E-3
mico-Ampere	microA	1.0E-6
nano-Ampere	nA	1.0e-9
pico-Ampere	pA	1.0e-12

Table 6: Amount substance unit symbols.

Unit Name	Unit Symbol	Conversion Factor
mole (default)	mole	1.0

Table 7: Luminosity unit symbols.

Unit Name	Unit Symbol	Conversion Factor
candella (default)	candella	1.0

Input File Usage: Use the following option to define the unit system:

**UNIT SYSTEM*, SYSTEM=*SI*, *MKS*, *CGS*, or *FPS*
physical dimension name, *unit symbol*, *conversion factor*

For example, the following input specifies the MKS system to define units for meter, kilogram, and second:

**UNIT SYSTEM=MKS*, SYSTEM=MKS

For example, the following input specifies the millimeter unit of length:

**UNIT SYSTEM*
LENGTH, mm

For example, the following input specifies the Celsius unit for temperature:

**UNIT SYSTEM*
TEMPERATURE, C
**PHYSICAL CONSTANTS*, ABSOLUTE ZERO=273.15

For example, the following input specifies the angstrom unit of length:

```
*UNIT SYSTEM
LENGTH, ,1.0E-10
```

Abaqus/CAE Usage: Defining a unit system for an Abaqus model is not supported in Abaqus/CAE.

Time Measures

Abaqus has two measures of time—step time and total time. Except for certain linear perturbation procedures, step time is measured from the beginning of each step. Total time starts at zero and is the total accumulated time over all general analysis steps (including restart steps; see *Restarting an Analysis*). Total time does not accumulate during linear perturbation steps.

Local Tangent Directions on Surfaces in Space

Local tangent directions are needed on surfaces in space; for example, to provide a convention for describing components of slip on an element-based contact surface or components of stress and strain in a shell. The convention used in Abaqus for such directions is as follows.

The default local 1-direction is the projection of the global x-axis onto the surface. If the global x-axis is within 0.1° of being normal to the surface, the local 1-direction is the projection of the global z-axis onto the surface. The local 2-direction is then at right angles to the local 1-direction, so that the local 1-direction, local 2-direction, and the positive normal to the surface form a right-handed set (see *Figure 1*). The positive normal direction is defined in an element by the right-hand rotation rule going around the nodes of the element. The local surface directions can be redefined; see *Orientations*.

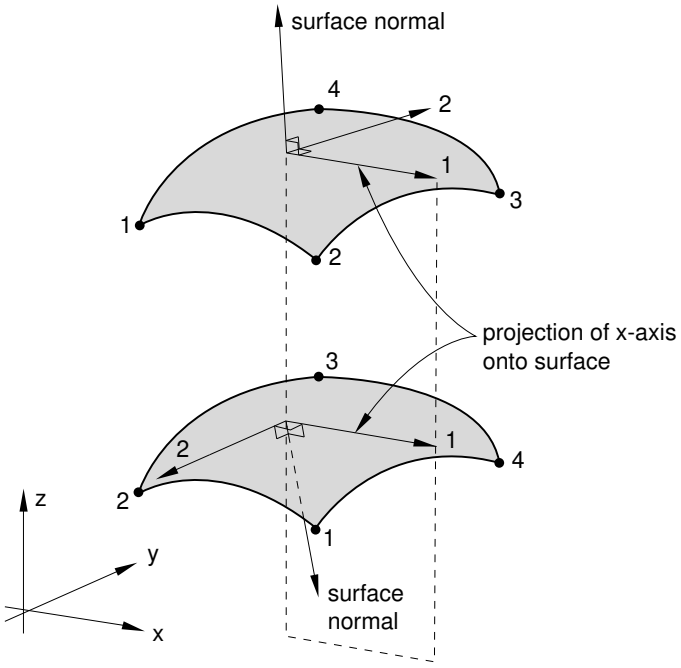


Figure 1: Default local surface directions.

The local 1- and 2-directions become local 2- and 3-directions, respectively, when considering gasket elements or the local systems associated with integrated output sections (*Integrated Output Section Definition*) or user-defined sections (*Section Output from Abaqus/Standard*).

For "line"-type surfaces defined on beam, pipe, or truss elements in space, the default local 1-direction and 2-direction are tangential and transverse to the elements. In this case the local surface directions can also be redefined as described in *Orientations*.

Rotation of the Local Directions

For geometrically linear analysis, stress and strain components are given by default in the material directions in the reference (initial) configuration.

For geometrically nonlinear analysis, small-strain shell elements in Abaqus/Standard (S4R5, S8R, S8R5, S8RT, S9R5, STRI3, and STRI65) use a total Lagrangian strain, and the stress and strain components are given relative to material directions in the reference configuration. Gasket elements are small-strain small-displacement elements, and the components are output by default in the behavior directions in the reference configuration.

For finite-membrane-strain elements (all membrane elements, S3/S3R, S4, S4R, SAX, and SAXA elements) and for small-strain shell elements in Abaqus/Explicit, the material directions rotate with the average rigid body motion of the surface to form the material directions in the current configuration. Stress and strain components in these elements are given relative to these material directions in the current configuration.

For a more thorough discussion of the definition of the rotated coordinate directions in membrane elements; S3/S3R, S4, and S4R elements; S3RS, S4RS, and S4RSW elements; and SAXA elements, see:

- *Membrane elements*,
- *Finite-strain shell element formulation*,
- *Small-strain shell elements in Abaqus/Explicit*, and
- *Axisymmetric shell element allowing asymmetric loading*.

You can determine whether the local system associated with a user-defined section is fixed or rotates with the average rigid body motion; see *Section Output from Abaqus/Standard* for details.

You can determine whether the local system associated with an integrated output section is fixed, translates with average rigid body motion, or translates and rotates with the average rigid body motion; see *Integrated Output Section Definition* for details.

See *Contact Formulations in Abaqus/Standard* for information on how the local tangent directions evolve during an Abaqus/Standard contact analysis.

Convention Used for Stress and Strain Components

When defining material properties, the convention used for stress and strain components in Abaqus is that they are ordered:

σ_{11}	Direct stress in the 1-direction
σ_{22}	Direct stress in the 2-direction
σ_{33}	Direct stress in the 3-direction
τ_{12}	Shear stress in the 1–2 plane
τ_{13}	Shear stress in the 1–3 plane
τ_{23}	Shear stress in the 2–3 plane

For example, a fully anisotropic, linear elasticity matrix is

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \tau_{12} \\ \tau_{13} \\ \tau_{23} \end{Bmatrix} = \begin{bmatrix} D_{1111} & D_{1122} & D_{1133} & D_{1112} & D_{1113} & D_{1123} \\ & D_{2222} & D_{2233} & D_{2212} & D_{2213} & D_{2223} \\ & & D_{3333} & D_{3312} & D_{3313} & D_{3323} \\ & \text{symm.} & & D_{1212} & D_{1213} & D_{1223} \\ & & & & D_{1313} & D_{1323} \\ & & & & & D_{2323} \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \gamma_{12} \\ \gamma_{13} \\ \gamma_{23} \end{Bmatrix}.$$

The 1-, 2-, and 3-directions depend on the element type chosen. For solid elements the defaults for these directions are the global spatial directions. For shell and membrane elements the defaults for the 1- and 2-directions are local directions in the surface of the shell or membrane, as defined in *About the Element Library*. In both cases the 1-, 2-, and 3-directions can be changed as described in *Orientations*.

For geometrically nonlinear analysis with solid elements, the default (global) directions do not rotate with the material. However, user-defined orientations do rotate with the material.

Abaqus/Explicit stores the stress and strain components internally in a different order: σ_{11}, σ_{22}, σ_{33}, τ_{12}, τ_{23}, τ_{13}. For geometrically nonlinear analysis, the internally stored components rotate with the material, regardless of whether or not a user-defined orientation is used. This distinction is important when a user subroutine (such as *VUMAT*) is used.

Nonisotropic Material Behavior

When nonisotropic material behavior is defined in continuum elements, a user-defined orientation is necessary for the anisotropic behavior to be associated with material directions. See *State storage* for a description of how material directions rotate.

Zero-Valued Stress Components

Stress components that are always zero are omitted from storage. For example, in plane stress Abaqus stores only the two direct components and one shear component of stress and strain in the plane where the stress values are nonzero.

Shear Strains

Abaqus always reports shear strain as engineering shear strain, γ:

$$\gamma_{ij} = \varepsilon_{ij} + \varepsilon_{ji}.$$

Stress and Strain Measures

The stress measure used in Abaqus is Cauchy or "true" stress, which corresponds to the force per current area. See *Stress measures* and *Stress rates* for more details on stress measures.

For geometrically nonlinear analysis, a large number of different strain measures exist. Unlike "true" stress, there is no clearly preferred "true" strain. For the same physical deformation different strain measures report different values in large-strain analysis. The optimal choice of strain measure depends on analysis type, material behavior, and (to some degree) personal preference. See *Strain measures* for more details on strain measures.

By default, the strain output in Abaqus/Standard is the "integrated" total strain (output variable E). For large-strain shells, membranes, and solid elements in Abaqus/Standard two other measures of total strain can be requested: logarithmic strain (output variable LE) and nominal strain (output variable NE).

Logarithmic strain (output variable LE) is the default strain output in Abaqus/Explicit; nominal strain (output variable NE) can be requested as well. The "integrated" total strain is not available in Abaqus/Explicit.

Total (Integrated) Strain

The default "integrated" strain measure, E, output by Abaqus/Standard to the data (`.dat`) and results (`.fil`) files for all elements that can handle finite strain is obtained by integrating the strain rate numerically in a material frame of reference:

$$\boldsymbol{\varepsilon}^{n+1} = \Delta\mathbf{R} \cdot \boldsymbol{\varepsilon}^{n} \cdot \Delta\mathbf{R}^{T} + \Delta\boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon}^{n+1}$ and $\boldsymbol{\varepsilon}^{n}$ are the total strains at increments $n+1$ and n, respectively; $\Delta\mathbf{R}$ is the incremental rotation tensor; and $\Delta\boldsymbol{\varepsilon}$ is the total strain increment from increment n to $n+1$. For elements that use a corotational coordinate system (finite-strain shells, membranes, and solid elements with user-defined orientations), the above equation simplifies to

$$\boldsymbol{\varepsilon}^{n+1} = \boldsymbol{\varepsilon}^{n} + \Delta\boldsymbol{\varepsilon}.$$

The strain increment is obtained by integration of the rate of deformation $\mathbf{D}$ over the time increment:

$$\Delta\boldsymbol{\varepsilon} = \int_{t^n}^{t^{n+1}} \mathbf{D}\, dt.$$

This strain measure is appropriate for elastic-(visco)plastic or elastic-creeping materials, because the plastic strains and creep strains are obtained by the same integration procedure. In such materials the elastic strains are small (because the yield stress is small compared to the elastic modulus), and the total strains can be compared directly with the plastic strains and creep strains.

If the principal directions of straining rotate with respect to the material axes, the resulting strain measure cannot be related to the total deformation, regardless whether a spatial or corotational coordinate system is used. If the principal directions remain fixed in the material axes, the strain is the integration of the rate of deformation,

$$\boldsymbol{\varepsilon}^{n+1} = \int_0^{t^{n+1}} \mathbf{D}\, dt,$$

which is equivalent to the logarithmic strain discussed later.

Green's Strain

For small-strain shells and beams in Abaqus/Standard, the default strain measure, E, is Green's strain:

$$\boldsymbol{\varepsilon}^{G} = \tfrac{1}{2}\left(\mathbf{F}^{T} \cdot \mathbf{F} - \mathbf{I}\right),$$

where $\mathbf{F}$ is the deformation gradient and $\mathbf{I}$ is the identity tensor. This strain measure is appropriate for the small-strain, large-rotation approximation used in these elements. The components of $\boldsymbol{\varepsilon}^{G}$ represent strain along directions in the original configuration. The small-strain shells and beams should not be used in finite-strain analysis with either elastic-plastic or hyperelastic material behavior, since incorrect analysis results may be obtained or program failure may occur.

Nominal Strain

The nominal strain, NE, is

$$\boldsymbol{\varepsilon}^{N} = \mathbf{V} - \mathbf{I} = \sum_{i=1}^{3} (\lambda_i - 1)\, \mathbf{n}_i\, \mathbf{n}_i{}^{T},$$

where $\mathbf{V} = \sqrt{\mathbf{F} \cdot \mathbf{F}^{T}}$ is the left stretch tensor, $\boldsymbol{\lambda_i}$ are the principal stretches, and $\mathbf{n_i}$ are the principal stretch directions in the current configuration. The principal values of nominal strain are, therefore, the ratios of change in length to length in the reference configuration in the principal directions, thus giving a direct measure of deformation.

Logarithmic Strain

The logarithmic strain, LE, is

$$\boldsymbol{\varepsilon}^{L} = \ln \mathbf{V} = \sum_{i=1}^{3} \ln \lambda_i \;\; \mathbf{n}_i\, \mathbf{n}_i{}^{T},$$

where the variables are as defined earlier for nominal strain. This is also the strain output for hyperelastic materials. For a hyperviscoelastic material, the logarithmic elastic strain EE is computed from the current (relaxed) stress state, and the viscoelastic strain CE is computed as LE − EE.

Stress Invariants

Many of the constitutive models in Abaqus are formulated in terms of stress invariants. These invariants are defined as the equivalent pressure stress,

$$p = -\tfrac{1}{3}\,\mathrm{trace}\;(\boldsymbol{\sigma});$$

the Mises equivalent stress,

$$q = \sqrt{\tfrac{3}{2}(\mathbf{S} : \mathbf{S})};$$

and the third invariant of deviatoric stress,

$$r = \left(\tfrac{9}{2}\mathbf{S} \cdot \mathbf{S} : \mathbf{S}\right)^{\frac{1}{3}};$$

where $\mathbf{S}$ is the deviatoric stress, defined as

$$\mathbf{S} = \boldsymbol{\sigma} + p\mathbf{I}.$$

Finite Rotations

The following convention is used for finite rotations in space: Define ϕ_x, ϕ_y, ϕ_z as "rotations" about the global *X*, *Y*, and *Z*-axes (that is, degrees of freedom 4, 5, and 6 at a node). Then define

$$p_x = \phi_x/\phi\,, \quad p_y = \phi_y/\phi\,, \quad p_z = \phi_z/\phi,$$

where

$$\phi = \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}.$$

The direction $\mathbf{p}$ is then the axis of rotation, and ϕ is the angular rotation (in radians) about the axis $\mathbf{p}$ according to the right-hand rule (see *Figure 2*).

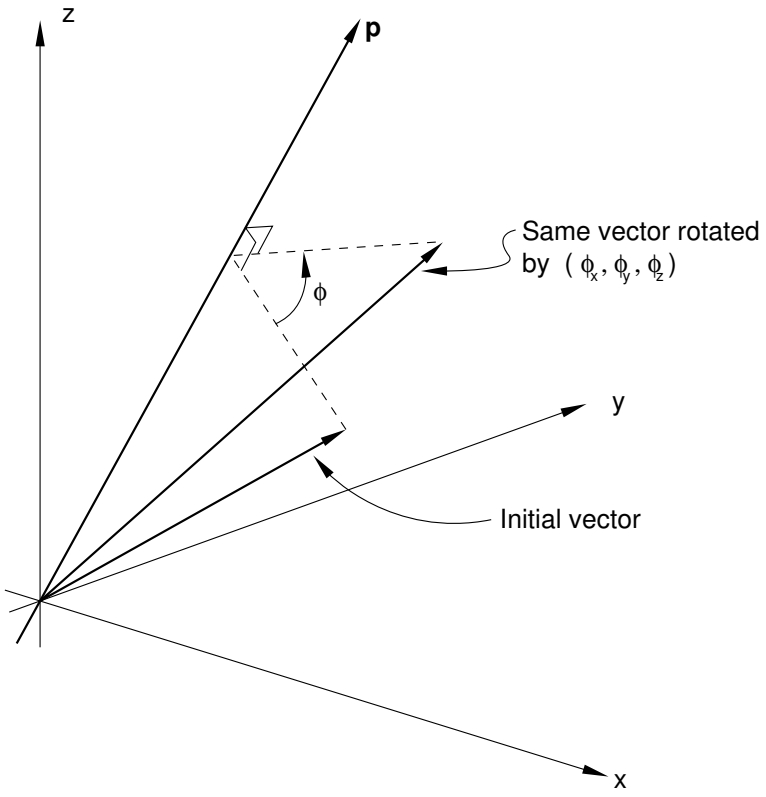


Figure 2: Definition of finite rotation.

The value of $\boldsymbol{\phi}$ is not uniquely determined. In large-rotation problems where the overall rotation exceeds $\mathbf{2\pi}$, any multiple of $\mathbf{2\pi}$ can be added or subtracted, which may lead to discontinuous output values for the rotation components. If rotations larger than $\mathbf{2\pi}$ about one axis occur in the positive (negative) direction in Abaqus/Standard, the rotation output varies discontinuously between *0* and $\mathbf{2\pi}$ ($\mathbf{-2\pi}$). In Abaqus/Explicit the rotation output varies in all cases between $\mathbf{-2\pi}$ and $\mathbf{2\pi}$.

This convention provides straightforward input of kinematic boundary conditions and moments in most cases and simple interpretation of the output. The rotations output by Abaqus represent a single rotation from the reference configuration to the current configuration about a fixed axis. The output does not follow the history of rotation at a node. In addition, this convention reduces to the usual convention for small rotations, even in the case of small rotations superposed on an initial finite rotation (such as might be considered in the study of small vibrations about a predeformed state).

Compound Rotations

Because finite rotations are not additive, the way they must be specified is a bit different from the way other boundary conditions are specified: the increment in rotation specified over a step must be the rotation needed to rotate the node from the configuration at the beginning of the step to that desired at the end of the step. It is not enough to rotate the node over this step to a total rotation vector that would have taken the node into its final configuration if applied on the node in some other initial reference configuration. If an increment of rotation $(\Delta\phi_x, \Delta\phi_y, \Delta\phi_z)$ is needed to rotate from the rotation boundary condition $(\phi_x, \phi_y, \phi_z)^t$ at the beginning of the step (and at the end of the previous step) to its final position at the end of the step, the boundary condition must be specified such that the rotation vector is $(\phi_x^t + \Delta\phi_x, \phi_y^t + \Delta\phi_y, \phi_z^t + \Delta\phi_z)$ at the end of the step. If the direction of the rotation vector is constant, this method of specifying rotation boundary conditions and the total rotation vector will be the same.

Example

As an example of how to specify compound finite rotations and to interpret finite rotation output, consider the following example of the rotation of a beam.

The beam initially lies along the *x*-axis. We want to perform the compound rotation, where (Step 1) the beam is rotated by 60° about the *z*-axis, followed by (Step 2) a 90° spin of the beam about itself, followed by (Step 3) a 90° rotation of the beam about an axis perpendicular to the beam in the *x*–*y* plane, such that the beam finishes on the *z*-axis.

This compound rotation is achieved in three steps with applied rotation vectors $\boldsymbol{\phi_1}$, $\boldsymbol{\phi_2}$, and $\boldsymbol{\phi_3}$, where

$$\begin{aligned}
\boldsymbol{\phi}_1 &= \phi_1 \mathbf{e}_z = \{0 \ 0 \ 1.047198\}^T, \\
\boldsymbol{\phi}_2 &= \phi_2 \left(\cos\phi_1 \mathbf{e}_x + \sin\phi_1 \mathbf{e}_y\right) = \{0.785398 \ 1.360350 \ 0\}^T, \quad \text{and} \\
\boldsymbol{\phi}_3 &= \phi_3 \left(\sin\phi_1 \mathbf{e}_x - \cos\phi_1 \mathbf{e}_y\right) = \{1.360350 \ -0.785398 \ 0\}^T.
\end{aligned}$$

For this example $\boldsymbol{\phi_1 = \pi/3}$, $\boldsymbol{\phi_2 = \pi/2}$, and $\boldsymbol{\phi_3 = \pi/2}$. Here $\boldsymbol{\phi_i}$ represents the magnitude of each finite rotation about the (unit length) rotation axis. The rotation vectors above are applied in each of the three steps on the configuration at the beginning of that step. It is most straightforward to prescribe these rotations with velocity-type boundary conditions. For convenience, the default amplitude reference in Abaqus for a velocity-type boundary condition is a constant value of one.

A typical Abaqus step definition for this example, where node 1 is pinned at the origin and the rotation is applied to node 2, is as follows:

```
*STEP, NLGEOM
Step 1: Rotate 60 degrees about the z-axis
```

```
*STATIC
*BOUNDARY, TYPE=VELOCITY
 2, 4, 5
 2, 6, 6, 1.047198
*END STEP
**
*STEP, NLGEOM
Step 2: Rotate 90 degrees about the beam axis
*STATIC
*BOUNDARY, TYPE=VELOCITY
 2, 4, 4, 0.785398
 2, 5, 5, 1.36035
 2, 6, 6
*END STEP
**
*STEP, NLGEOM
Step 3: Rotate beam onto z-axis
*STATIC
*BOUNDARY, TYPE=VELOCITY
 2, 4, 4, 1.36035
 2, 5, 5, -0.785398
 2, 6, 6
*END STEP
```

The above method for applying finite-rotation boundary conditions (using a velocity-type boundary condition with the default constant amplitude definition) is strongly recommended. However, if the rotation boundary conditions are applied as displacement-type boundary conditions, the input syntax would change.

The Abaqus/Standard convention for boundary condition specification within a step is to specify the total or final boundary state. In such a case the specified boundary conditions from all of the previous steps must be added to the incremental rotation vector components. The Abaqus/Standard step definitions from above would change to:

```
*STEP, NLGEOM
Step 1: Rotate 60 degrees about the z-axis
*STATIC
*BOUNDARY
 2, 4, 5
 2, 6, 6, 1.047198
*END STEP
**
*STEP, NLGEOM
Step 2: Rotate 90 degrees about the beam axis
*STATIC
*BOUNDARY
 2, 4, 4, 0.785398
 2, 5, 5, 1.36035
 2, 6, 6, 1.047198
*END STEP
**
*STEP, NLGEOM
Step 3: Rotate beam onto z-axis
*STATIC
*BOUNDARY
 2, 4, 4, 2.145748
 2, 5, 5, 0.574952
 2, 6, 6, 1.047198
*END STEP
```

The boundary conditions in Steps 2 and 3 are the sum of the incremental rotation components plus the rotation boundary conditions specified in the previous steps.

In Abaqus/Explicit references to amplitude definitions should be used such that there are no jumps in displacement across the steps. It is often convenient to use amplitude definitions given in terms of total time for this purpose. The displacement boundary conditions will be applied incrementally based on the increment in the value of amplitude curve over the time increment. Therefore, any sudden jumps in displacement at the beginning of a step introduced either without the amplitude curves or with two amplitude curves is ignored (see *Boundary Conditions*). The Abaqus/Explicit step definitions for the above example would change to:

```
*AMPLITUDE, TIME=TOTAL TIME, NAME=RAMPUR1
 0., 0., 0.001, 0., 0.002, 0.785398, 0.003, 2.145748
*AMPLITUDE, TIME=TOTAL TIME, NAME=RAMPUR2
 0., 0., 0.001, 0., 0.002, 1.36035, 0.003, 0.574952
*AMPLITUDE, TIME=TOTAL TIME, NAME=RAMPUR3
 0., 0., 0.001, 1.047198, 0.002, 1.047198, 0.003, 1.047198
*STEP
Step 1: Rotate 60 degrees about the z-axis
*DYNAMIC, EXPLICIT
 , 0.001
*BOUNDARY, AMP=RAMPUR1
 2, 4, 4, 1.0
*BOUNDARY, AMP=RAMPUR2
 2, 5, 5, 1.0
*BOUNDARY, AMP=RAMPUR3
 2, 6, 6, 1.0
*END STEP
**
*STEP
Step 2: Rotate 90 degrees about the beam axis
*DYNAMIC, EXPLICIT
 , 0.001
*END STEP
**
*STEP
Step 3: Rotate beam onto z-axis
*DYNAMIC, EXPLICIT
 , 0.001
*END STEP
```

The boundary conditions in Steps 2 and 3 are the sum of the incremental rotation components plus the rotation boundary conditions specified in the previous steps.

The Abaqus output of the rotation field at the end of Step 3 is

$$\mathtt{UR1} = 1.927\ ,\quad \mathtt{UR2} = -0.5163\ ,\quad \mathtt{UR3} = 1.927\,.$$

We see that none of the individual components of the specified boundary conditions appears in the final rotation output. The final rotation output represents the rotation vector required to obtain the final orientation in a single step.

Suppose that in Step 3 of the previous example we want to apply the rotation vector $\boldsymbol{\phi}_3$ at node 1 instead of at node 2. If the rotation is applied incrementally, the Abaqus/Standard step definition is as follows:

```
*STEP, NLGEOM
Step 3: Rotate beam onto z-axis
*STATIC
*BOUNDARY, TYPE=VELOCITY, OP=NEW
 1, 1, 3
 1, 4, 4, 1.36035
 1, 5, 5, -0.785398
 1, 6, 6
*END STEP
```

and the Abaqus/Explicit step definition is similar. It is necessary to remove the rotation boundary conditions that are in effect at node 2.

As mentioned previously, using velocity-type boundary conditions is the preferred method for applying finite-rotation boundary conditions. If the rotation boundary condition is to be applied as a displacement-type boundary condition, we must first retrieve the rotation field at node 1 at the end of Step 2. The Abaqus output of this rotation field is

$$\mathtt{UR1} = 1.412\ ,\ \mathtt{UR2} = 0.8155\ ,\ \mathtt{UR3} = 0.8155\,.$$

These rotation vector components must then be added to the incremental rotation vector components we want to prescribe in Step 3. The Abaqus/Standard step definition would change to

```
*STEP
Step 3: Rotate beam onto z-axis
*STATIC
*BOUNDARY, OP=NEW
 1, 1, 3
 1, 4, 4, 2.772
 1, 5, 5, 0.0301
 1, 6, 6, 0.8155
*END STEP
```

and the Abaqus/Explicit step definition would change to:

```
*STEP
Step 3: Rotate beam onto z-axis
*DYNAMIC, EXPLICIT
 , 0.001
*AMPLITUDE, TIME=STEP TIME, NAME=NODE1UR1
 0., 1.412, 0.001, 2.772
*AMPLITUDE, TIME=STEP TIME, NAME=NODE1UR2
 0., 0.8155, 0.001, 0.0301
*AMPLITUDE, TIME=STEP TIME, NAME=NODE1UR3
 0., 0.8155, 0.001, 0.8155
*BOUNDARY, OP=NEW
 1, 1, 3
*BOUNDARY, OP=NEW, AMP=NODE1UR1
 1, 4, 4, 1.
*BOUNDARY, OP=NEW, AMP=NODE1UR2
 1, 5, 5, 1.
*BOUNDARY, OP=NEW, AMP=NODE1UR3
 1, 6, 6, 1.
*END STEP
```

The boundary conditions are again specified in the Abaqus/Explicit input using amplitude curves to avoid any sudden jump in their values at the beginning of the step. As stated above and in *Boundary Conditions*, any jumps in the displacement values are ignored and the boundary is maintained at the previous values.

As this last procedure clearly demonstrates, it is simpler to apply finite-rotation boundary conditions as velocity-type boundary conditions rather than as displacement-type boundary conditions. The recommended method of specifying finite-rotation boundary conditions is also described in *Boundary Conditions*. For further discussion of how finite rotations are accumulated, see *Rotation variables*.

Abaqus Model Definition

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Input Syntax Rules*
- *Importing and Transferring Results*
- *Output to the Output Database*
- *Abaqus Keywords Guide*
- *Abaqus/CAE User's Guide*

Overview

An analysis in Abaqus is defined by an *input file*, which

- contains *keyword lines* and *data lines*; and
- is divided into *model* data and *history* data.

The Input File

An Abaqus input file is an ASCII data file. It can be created by using a text editor or by using a graphical preprocessor such as Abaqus/CAE. The input file consists of a series of lines containing Abaqus options (keyword lines) and data (data lines). The input syntax for keyword and data lines is described in *Input Syntax Rules*.

Most input files have the same basic structure. The following portions of the input file are specified to define a finite element model:

1. An input file often begins with the **HEADING* option, which is used to define a title for the analysis. Any number of data lines can be used to give the title; they will appear at the beginning of the output files (*About Output*). The first heading line will appear as a heading at the top of each page of the output.

   While including a title can be helpful for users examining your input file, the **HEADING* option is not required.
2. After the heading the input file usually contains a *model* data section to define nodes, elements, materials, initial conditions, etc. The model data section is explained below.
3. If the model is organized into an assembly of part instances, the model data are further categorized and must fall within the proper level: part, assembly, instance, or model. Models defined in terms of an assembly of part instances are discussed in *Assembly Definition*.
4. Finally, the input file contains *history* data to define the analysis type, loading, output requests, etc. Step definitions divide the model data from the history data in an input file: everything appearing before the first step definition is model data, and everything appearing within and following the first step definition is history data. The history data section is explained below.

The input file is processed by the "analysis input file processor" prior to executing the appropriate analysis product, Abaqus/Standard or Abaqus/Explicit. The functions of the analysis input file processor are to interpret the Abaqus options, to perform the necessary consistency checking, and to prepare the data for the analysis products.

Most computational mechanics modeling options (element types, loading types, etc.) are available in both Abaqus/Standard and Abaqus/Explicit, although some options are available in only one analysis product or the other. All of the step procedure types used in an input file must be from the same analysis product; however, it is possible to import a solution from Abaqus/Standard into Abaqus/Explicit and vice versa (see *Importing and Transferring Results*), which allows each analysis product to be used at the various stages of an analysis for which it is best suited (for example, a static preloading in Abaqus/Standard followed by a dynamic analysis in Abaqus/Explicit).

Model Data

Model data define the nodes, elements, materials, initial conditions, etc.

Required Model Data

The following model data must be included in an input file to define a finite element model:

Geometry

The geometry of a model is described by elements and their nodes. The rules and methods for defining nodes and elements are described in *Node Definition*, *Element Definition*, and *Assembly Definition*. Cross-sections for structural elements (such as beams) must be defined. Special features can be defined with special elements such as springs, dashpots, point masses, etc. The element types available for modeling are described in *About the Element Library*, along with explanations of how to define the elements. You can view the initial mesh or the configuration after adjustment for initial overclosure in the Visualization module of Abaqus/CAE after a data check run (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Material definitions

A material type must be associated with most portions of the geometry. The material library is described in *About the Material Library*. Special elements such as springs or dashpots do not have an associated material, but their properties must be defined.

Optional Model Data

The following model data can be included as necessary:

Parts and an assembly

The geometry of a model can be defined by organizing it into parts, which are positioned relative to one another in an assembly (*Assembly Definition*).

Initial conditions

Nonzero initial conditions such as initial stresses, temperatures, or velocities can be specified (*Initial Conditions*).

Boundary conditions

Zero-valued boundary conditions (including symmetry conditions) can be imposed on individual solution variables such as displacements or rotations (*Boundary Conditions*).

Kinematic constraints

Equations involving several of the fundamental solution variables in the model (*Linear Constraint Equations*) or multi-point constraints (*General Multi-Point Constraints*) can be defined.

Interactions

Contact and other interactions between parts can be defined (*About Contact Interactions*).

Amplitude definitions

Amplitude curves can be defined for later use in specifying time-dependent loading or boundary conditions (*Amplitude Curves*).

Output control

You can control model definition output to the data file (*About Output*).

Environment properties

Environment properties, such as the attributes of a fluid surrounding the model, may have to be defined.

Analysis continuation

It is possible to write restart data or to use the results from a previous analysis and continue the analysis with new model or history data (*Restarting an Analysis*), with a new mesh (*About Submodeling*, *Mesh-to-Mesh Solution Mapping*, and *Symmetric Model Generation*), or with the same or a different Abaqus program (*About Transferring Results between Abaqus Analyses*).

History Data

The purpose of an analysis is to predict the response of a model to some form of external loading or to some nonequilibrium initial conditions. An Abaqus analysis is based on the concept of *steps*, which are described in the history data portion of the input file. (For more information on steps, see *Defining an Analysis*.) The history input data are combined within a step as needed to define the history of the analysis.

Multiple steps can be defined in an analysis. Steps can be introduced simply to change the output requests or to change the loads, boundary conditions, analysis procedure, etc. There is no limit on the number of steps in an analysis.

There are two kinds of steps in Abaqus: general response analysis steps, which can be linear or nonlinear; and, in Abaqus/Standard, linear perturbation steps (see *General and Perturbation Procedures*). A general analysis step contributes to the response history of the system; a linear perturbation step allows the investigation of the perturbation response of the system with respect to a base state at any stage during the response history. The solution from the perturbation response is not carried over to subsequent steps and, therefore, does not contribute to the response history.

The state at the end of a general step provides the initial conditions for the next step, making it easy to simulate consecutive loadings of a model, such as a dynamic response following a static preload or the loading of a product during its usage following a simulation of the manufacturing process.

The optional history data described below prescribing the loading; boundary conditions; output controls; auxiliary controls; and, in Abaqus/Explicit, contact conditions are continued from one general analysis step to the next general analysis step unless modified. For example, the solution controls prescribed in a general analysis step in Abaqus/Standard (see *About Convergence and Time Integration Criteria*) will remain in effect for all subsequent general analysis steps until they are modified or reset. For linear perturbation steps only the output controls are continued from one linear perturbation step to the next if there are no intermediate general analysis steps and the output controls are not redefined (see *About Output*).

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

**STEP*

Use the following option to end a step definition:

**END STEP*

Required History Data

The following history data must be included in an input file to define an analysis procedure:

Response type

An option to define the analysis procedure type must appear immediately after the beginning of the step definition.

Abaqus can perform many types of analyses—linear or nonlinear, static or dynamic, etc. (see *Defining an Analysis*). The type of analysis can be changed from step to step. For example, in Abaqus/Standard a static preload can be analyzed first, then the response type can be changed to transient dynamic. In this way a linear or nonlinear dynamic analysis can be performed based on the conditions at the end of the static solution.

Optional History Data

The following history data can be included as necessary:

Loading

Usually some form of external loading is defined. For example, concentrated or distributed loads can be applied (*About Loads*), temperature changes leading to thermal expansion can be prescribed (*Thermal Expansion*), or contact conditions can be used to apply loads (*About Contact Interactions*).

The loading can be prescribed as a function of time (*Amplitude Curves*). This feature can be used to prescribe loadings such as the ground motion during a seismic event, known accelerations, or the temperature and pressure history during a transient in an engine. If an amplitude curve is not defined, Abaqus assumes either that the loading varies linearly over the step or that the load is applied instantaneously at the beginning of the step, depending on the chosen response type (see *Defining an Analysis*).

Boundary conditions

Boundary conditions can be added, modified, or removed during an analysis (*Boundary Conditions*).

Output control

Quantities such as stress, strain, reaction force, temperature, and energy are available as output. The output options are described in *Output to the Data and Results Files* and *Output to the Output Database*, and all of the output variables are listed in *Output Variables*. The available output files are described in *About Output*.

Contact

Contact surfaces and contact interactions can be added, modified, or removed as step-dependent history data during an Abaqus/Explicit analysis (see *About Contact Interactions*).

Auxiliary controls

You can overwrite the solution controls that are built into Abaqus. In some procedures these values are given in the procedure definition. More generally in Abaqus/Standard they are given by defining solution controls (*Commonly Used Control Parameters*). Solution controls for contact problems (*Generally Applicable Contact Controls in Abaqus/Standard*, *Common Difficulties Associated with Contact Modeling Using Contact Pairs in Abaqus/Explicit*, or *Contact Controls for General Contact in Abaqus/Explicit*) can also be defined.

Element and surface removal/reactivation

In Abaqus/Standard portions of the model can be removed or reactivated from step to step. See *Element and Contact Pair Removal and Reactivation*.

Co-simulation

The steps in the Abaqus model must be defined such that the co-simulation fits entirely within a single Abaqus step. Further, there can be only one co-simulation in the Abaqus job.

Including Model or History Data from an External File

You can specify an external file that contains a portion of the Abaqus input file. This file can include model and history definition data, comment lines, and other references to external files. When a reference to an external file is encountered, Abaqus will immediately process the data within the specified file. When the end-of-file is reached, Abaqus will return to processing the original file.

A maximum of five levels of nested external file references can be used. Linux environment variables can be used to specify the file names.

Input File Usage: **INCLUDE*, INPUT=*file_name*

Including an Encrypted Data File

You can include an encrypted file by reference in an Abaqus input file or in another data file. When you refer to the encrypted file, you must also provide the file's password. If the password is correct, Abaqus processes the data within the specified file as it would for an unencrypted external file. Material and connector behavior definitions within an encrypted input file are not written to the output database. In addition, all material and

connector behavior definitions output to the data file are suppressed if an encrypted file is used as input for any portion of the model. See *Encrypting and Decrypting Abaqus Input Data* for details about the encryption utility.

Some encrypted files are eligible for inclusion only by users with a license for a particular Abaqus feature (such as Abaqus/Explicit) or to users at a particular site. If you attempt to include an encrypted file for which you do not have the proper privileges, Abaqus issues an error message.

You cannot include encrypted input files that contain parametric input.

Input File Usage: **INCLUDE*, INPUT=*file_name*, PASSWORD=*password*

Parametric Modeling

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Scripting Parametric Studies*
- *Parametric Shape Variation*
- **PARAMETER*
- **PARAMETER DEPENDENCE*
- **PARAMETER SHAPE VARIATION*
- *Introduction to Python*

Overview

The parametric input capability allows you to create an Abaqus input file in which:

- Any number of input parameters is defined by assigning a value to each one of them.
- The parameters defined in the input file are used in place of input quantities.
- The parameters are evaluated according to their definition and are substituted for the parametrized input quantities before an analysis is run.

Parametric input allows greater flexibility in building and manipulating models. The different kinds of parameters and the different ways of parametrizing the Abaqus input quantities are discussed in this section.

Introduction

You must define all the parameters you wish to use in an analysis by assigning a value to them. The Python language (*Lutz, 1999*) is used to perform parameter evaluation and substitution; hence, parameter definitions are required to follow the Python syntax rules discussed later in this section. These parameters can then be used in place of input quantities.

Input File Usage: Use the following option to define parameters:

**PARAMETER*

You must include the **PARAMETER* option in any file where you define parameters.

Use these parameters in place of input quantities by delimiting them with < >.

For example, the following input defines the two parameters `width` and `height`, which are then used to define beam section properties:

```
*PARAMETER
width = 2.5
```

height = width*2
**BEAM SECTION*, SECTION=RECT, ELSET=*name*,
MATERIAL=*name*
<width>, <height>

In this simple example models with beams of different cross-sections can be obtained simply by changing the values of the parameters.

Parameters

Parameters are user-named variables to which you assign values. When a parameter is used instead of a value, the value of that parameter is substituted. There are two basic types of parameters: independent parameters and dependent parameters.

Independent Parameters

Independent parameters are those that do not depend on any other parameters. The following are examples of independent parameters:

```
thickness = 10.0
area = 5.0**2
length = 3.0*sin(45*pi/180.0) # convert degrees to radians
```

Python expressions using numbers and numerical operations (such as addition, multiplication, and exponentiation) can be used to define independent parameters. Arithmetic support in Python is discussed later in this section.

Dependent Parameters

Dependent parameters are those that depend on other parameters (dependent or independent). Dependent parameters can be defined in one of two ways: using a mathematical expression or using a tabular dependence.

Expressional Dependence

Python parametric expressions involving operations between numbers and parameters are used to define expressional-dependent parameters. In the following example `area` and `mom_inertia` are dependent parameters:

```
width = 2.0
height = 5.0
area = width*height
mom_inertia = area*height**2/12.0
```

Tabular Dependence

Tabular dependence between parameters is defined by specifying the dependent and independent parameters as well as a dependence table. The table that defines the dependence between the parameters must have as many values per line as the number of dependent parameters plus the number of independent parameters for which it is going to be used. The table must contain only real values; dependent parameter values are given first, followed by independent parameter values. Parameter names and character strings cannot be used in a table.

The evaluation of tabularly dependent parameters by interpolation between values in a table will result in these parameters being assigned real values. If it is necessary that the tabularly dependent parameters be integer numbers, the real numbers must be converted to integer numbers as described later in the Python language section.

When the tabularly dependent parameters are functions of only one independent parameter, the tabular data must be given in order of increasing values of the independent parameter. Abaqus then interpolates linearly for values between those given. The dependent parameters are assumed to be constant outside the range of the independent parameters used in a table. When the tabularly dependent parameters depend on several independent parameters, the variation of the dependent parameters with respect to the first independent parameter must be given at fixed values of the other independent parameters, in ascending values of the second independent parameter, then of the third independent parameter, and so on. The table lines must always be ordered so that the independent parameters are given increasing values. This process ensures that the value of each dependent parameter is completely and uniquely defined for all values of the independent parameters.

The fact that the definition of the dependence table is separate from the assignment of the dependence to particular parameters means that the same table can be used for multiple sets of dependent/independent parameters. This is useful when there are different instances of the same kind of input data; for example, multiple material definitions that use the same dependence but different sets of parameters.

Because the evaluation of parameters is procedural (see *Parameter Evaluation* below), a parameter dependence table must always be defined before it is used to specify tabular parameter dependencies.

Independent parameters in tabular dependence definitions are treated as independent for the purpose of defining this dependency; however, these "independent" parameters can be defined to depend on other parameters in a preceding parameter definition.

Input File Usage: Use the following option to define a parameter dependence table:

**PARAMETER DEPENDENCE*, TABLE=*name*, NUMBER VALUES=*n*
table with n values per line

Use the following option to define the dependent and independent parameters that are used in the dependence table:

**PARAMETER*, TABLE=*name*, DEPENDENT=*(parList)*, INDEPENDENT=*(parList)*

Rules for Parameters

Some general rules apply to all parameters used in Abaqus input files. These rules are described in the following subsections.

Parameter Evaluation

Parameters are evaluated by ordered execution of the parameter definitions as they appear in the input file. For example, the input

```
*PARAMETER
x = 2
y = x + 3
x = 4
```

gives x=4 and y=5, not x=4 and y=7. The input

```
*PARAMETER
y = x + 3
x = 4
```

is flagged as an error because y cannot be evaluated by ordered execution of the input. In other words, there is no deferred execution of the parameter definitions.

It is possible to define parameters anywhere in the input file, even after parameters have been used in place of input quantities, since the parameter definitions are always processed before any other input options are processed.

Parameters can also be defined and used in place of input quantities in an input file used for a restart analysis. However, parameters defined in the input file for the original analysis (from which the restart run is continued) are not available in the restart analysis.

Parameter Substitution

When the parameterized data are processed, Abaqus assigns the parameter values as determined at the end of parameter evaluation. An error is reported if a parameter used in place of input quantities has not been assigned a value. Later, the analysis input file processor performs its usual checks on the validity of the parameter values with respect to the options in which they are being used.

Data given to define a parameter, a parameter dependence table, or a parameter shape variation cannot be parameterized. For example, the input

```
*PARAMETER SHAPE VARIATION
<x>
```

is not valid; however, the analysis input file processor will not report an error for this input.

Data Types

The data type of a parameter is deduced from its definition. An integer parameter results from assigning an integer literal value to the parameter. Similarly, a real parameter arises from assigning a real literal value to the parameter. The result of operations between integers and reals will be real. A character string parameter results from assigning a character string literal value to the parameter.

The input option context in which the parameter is used dictates the data type that the parameter must have. Parameters of real data type should be used in place of real Abaqus input quantities. Parameters of integer (or character string) type should be used in place of integer (or character string) type input quantities, respectively. In some instances, mismatches between the input context and the type of the substituted parameter will cause the analysis input file processor to flag these instances as input errors. For example, the input

```
*PARAMETER
int_pts = 5.0
*SHELL SECTION
10.0, <int_pts>
```

will cause the analysis input file processor to report an error because the number of integration points specified for a shell section must be an integer. However, the input

```
*PARAMETER
thick = 5//4
*SHELL SECTION
<thick>,
```

will be accepted by the analysis input file processor without a warning being flagged; as a result of doing integer division, this input gives a shell thickness of 1 (not 1.25). In conclusion, you can rely on the analysis input file processor to catch only some data type errors.

Continuous and Discrete Parameters

From the point of view of design activities (sensitivity analysis, parametric studies, etc.) parameters can be continuous valued or discrete valued. A continuous-valued parameter is differentiable and can, thus, be used for design sensitivity analysis purposes. A discrete-valued parameter is not differentiable and can, thus, not be used for design sensitivity analysis purposes; however, it can be used for parametric studies. Examples of continuous-valued parameters may be a shell thickness or a material property. Examples of discrete-valued parameters may be the number of integration points through the thickness of a shell, or an element type. Continuous-valued parameters generally coincide with physical (design) input quantities, while discrete-valued parameters generally coincide with finite element (numerical approximation) input quantities.

Auxiliary Input Files

Parameters can be defined in **INCLUDE* input files but not in any other auxiliary input files. If you define parameters in an **INCLUDE* input file, you must also include the **PARAMETER* option in the **INCLUDE* input file. For information on how the parameters are interpreted, see *Execution of Parametrized Input*. Names of auxiliary input files can be parameterized, except those used in the **INCLUDE* option.

Parametrization of Input Quantities

Abaqus treats parametrization of "size" and "shape" quantities somewhat differently. Parametrization of shape input quantities is discussed in a separate section (see *Parametric Shape Variation*).

Size input quantities are understood to include all Abaqus input quantities except those that relate to shape. Size input quantities include section properties, material properties, orientation properties, prescribed conditions, interaction definitions and properties, and analysis procedure data.

Parametrizing Individual Input Quantities

The following example shows the parametrization of shell section input using three independent parameters of differing data types:

```
*ELSET, ELSET=<shell_set>, GEN
1, 111, 10
*PARAMETER
shell_set = 'lining'
shell_thick = 1.E2
num_int_pts = 5
*SHELL SECTION, ELSET=<shell_set>, MATERIAL=name
<shell_thick>, <num_int_pts>
```

Parametrizing Groups of Input Quantities (Expressional Dependence)

The following example shows the parametrization of a three-layer composite shell section using expressional-dependent parameters. In this example the `thickness` parameter can be used to change the thickness of the layers of the composite section uniformly.

```
*PARAMETER
thickness = 10.
layer1_thick = 0.15*thickness
layer2_thick = 0.6*thickness
layer3_thick = 0.25*thickness
*SHELL SECTION, ELSET=, COMPOSITE
<layer1_thick>,num int pts, material name, orientation
<layer2_thick>,num int pts, material name, orientation
<layer3_thick>,num int pts, material name, orientation
```

This parametrization requires that dependent parameters be created for the three input quantities (`layer1_thick`, `layer2_thick`, `layer3_thick`) that each depend on the independent parameter (`thickness`).

Parametrizing Groups of Input Quantities (Tabular Dependence)

The following example shows the parametrization of the section properties of a box beam. The height and wall thicknesses of the beam section are parameters that depend tabularly on the section width.

```
*PARAMETER
a = 60.
*PARAMETER DEPENDENCE, TABLE=sectprop, NUMBER VALUES=6
25.0, 1.04,  1.04, 1.04, 1.04, 50.0
50.0, 4.17,  3.13, 2.08, 2.50, 100.0
75.0, 9.38,  6.24, 3.13, 4.90, 150.0
*PARAMETER, TABLE=sectprop, DEPENDENT=(b, t1, t2, t3, t4),
INDEPENDENT=(a)
*BEAM SECTION, SECTION=BOX, ELSET=beams, MATERIAL=steel
<a>, <b>, <t1>, <t2>, <t3>, <t4>
```

The above parametrization creates dependent parameters (`b`, `t1`, `t2`, `t3`, `t4`) that each depend on the independent parameter (`a`). Usage of tabular dependence allows the definition of the dependencies of input quantities on parameters to be confined to the parameter definitions; i.e., separate from the options where parametrization of input quantities is done. An advantage of this method of parametrization is that the same parameter dependence table can be used for different parameters in different input options. For example, you may wish to use beams of different cross-section dimensions in different parts of the structure being modeled. The parameter dependence table can be reused with new dependent (`bb`, `tt1`, `tt2`, `tt3`, `tt4`) and independent (`aa`) parameters.

```
*PARAMETER
aa = 65.
*PARAMETER, TABLE=sectprop, DEPENDENT=(bb, tt1, tt2, tt3, tt4),
INDEPENDENT=(aa)
*BEAM SECTION, SECTION=BOX, ELSET=columns, MATERIAL=steel
<aa>, <bb>, <tt1>, <tt2>, <tt3>, <tt4>
```

In options where predefined field variable dependence is supported, this method of parametrization provides a clear separation between predefined field variable dependence and parameter dependence; therefore, field variable and parameter dependence can never be confused. Consider, for example, the case of perfect plasticity properties for a metal where the yield stress depends on a field variable and is also parametrized to depend tabularly on the carbon content of the metal alloy.

```
*PARAMETER
carbon = 0.01
*PARAMETER DEPENDENCE, TABLE=yield_data, NUMBER=4
ys_fv1 val 1, ys_fv2 val 1, ys_fv3 val 1, carbon val 1
ys_fv1 val 2, ys_fv2 val 2, ys_fv3 val 2, carbon val 2
ys_fv1 val 3, ys_fv2 val 3, ys_fv3 val 3, carbon val 3
ys_fv1 val 4, ys_fv2 val 4, ys_fv3 val 4, carbon val 4
*PARAMETER, TABLE=yield_data, DEPENDENT=(ys_fv1, ys_fv2, ys_fv3),
INDEPENDENT=(carbon)
*MATERIAL, NAME=alloy
*PLASTIC, DEPENDENCIES=1
<ys_fv1>, , , fv val 1
<ys_fv2>, , , fv val 2
<ys_fv3>, , , fv val 3
```

Consider, for example, the case of metal creep properties where the creep material data are parameters that depend tabularly on the carbon content of the metal alloy. In addition, one of the creep parameters, `A`, also depends on a predefined field variable.

```
*PARAMETER
carbon = 0.01
*PARAMETER DEPENDENCE, TABLE=creepdata, NUMBER=6
A_fv1 val 1, A_fv2 val 1, A_fv3 val 1, n val 1, m val 1, carbon val 1
```

```
A_fv1 val 2, A_fv2 val 2, A_fv3 val 2, n val 2, m val 2, carbon val 2
A_fv1 val 3, A_fv2 val 3, A_fv3 val 3, n val 3, m val 3, carbon val 3
A_fv1 val 4, A_fv2 val 4, A_fv3 val 4, n val 4, m val 4, carbon val 4
*PARAMETER, TABLE=creepdata, DEPENDENT=(A_fv1, A_fv2, A_fv3,
n, m), INDEPENDENT=(carbon)
*MATERIAL, NAME=alloy
*CREEP, DEPENDENCIES=1
<A_fv1>, <n>, <m>, , fv val 1
<A_fv2>, <n>, <m>, , fv val 2
<A_fv3>, <n>, <m>, , fv val 3
```

This example shows that any combination of dependencies on predefined field variables and/or dependent parameters can be defined.

Python Language

Parameter statements in parameter definitions are required to follow the syntax and semantics of version 3 of the Python language (note that the parameter dependence table and parameter shape variation definitions follow the usual Abaqus input syntax rules). The subset of the Python language that is endorsed is documented here.

Statement Length and Continuation Lines

Python statements in parameter definitions can be continued over multiple lines by terminating each line with a backslash character (\). The **PARAMETER*PARAMETER*keyword lines can be continued onto the following line using a trailing comma since they are treated like other Abaqus keyword lines.

Comments

Comments in a parameter definition start with the number character (#) and continue to the end of the line. However, comments in a parameter dependence table or parameter shape variation definition are indicated by the usual Abaqus input syntax convention (**).

Parameter Names

Parameter names must begin with a letter and can contain the underscore character (_) and numbers. Parameter names are case sensitive.

Data Types

Data types are limited to character strings, integers, and reals.

Strings are delimited with single or double quotation marks (' ' or ” ”). Backward single quotation marks (` `) are not permitted. Character strings should not contain the backslash character (\).

Integers are created by assignment to integer literals (for example, *aInt* = 2).

Reals are created by assignment to real literals (for example, *aReal* = 1.0). Real numbers can be given with or without an exponent. Any exponent must be preceded by `E` or `e`. The following line shows five acceptable ways of entering the same real number:

```
-12.345, -1234.5E-2, -0.12345E+2, -0.12345E2, -0.12345e2
```

The syntax

```
-0.12345D+2
```

(allowed elsewhere in the Abaqus input file) is not valid in Python.

Type Conversion

If integers and reals are mixed in expressions, the result of the expression will be a real. Explicit type conversion can be obtained using:

int(*aReal*)

aReal converted to integer type

float(*anInt*)

anInt converted to real type (float is the same as real)

str(*anIntOrReal*)

anIntOrReal converted to character string type

Numeric Operators

Standard support for operators is provided:

– x

x negated

+ x

x unchanged

x + y

sum of x and y

x – y

difference of x and y

x * y

product of x and y

x / y

quotient of x and y, always resulting in a real

x // y

floor of quotient of x and y

x**y

x to the power y

Functions

The following utility functions are supported:

abs(x)

absolute value of x

acos(x)

arc cosine of x (result is in radians)

asin(x)

arc sine of x (result is in radians)

atan(x)

arc tangent of x (result is in radians)

cos(x)

cosine of x (x is in radians)

log(x)

natural logarithm of x

log10(x)

base 10 logarithm of x

pow(x,y)

x to the power y (equivalent to x**y)

sin(x)

sine of x (x is in radians)

sqrt(x)

square root of x

tan(x)

tangent of x (x is in radians)

Character String Operators

'abc' + 'def'

concatenation of character string 'abc' and character string 'def'

Execution of Parametrized Input

Jobs with parametrized input files are submitted to Abaqus in the usual way; for example,

```
abaqus job=job-name input=input-file
```

where it is assumed that an input file named *input-file*`.inp` exists.

Abaqus searches *input-file*`.inp` and any **INCLUDE* input files for parameter, parameter dependence table, and parameter shape variation (*Parametric Shape Variation*) definitions, as well as parameter names inside < > that may have been used in place of input quantities. You can define parameters in both the *input-file*`.inp` and **INCLUDE* input files. You must include the **PARAMETER* option in any file where you define parameters. If any of the above are found, Abaqus interprets the parametrized input file and performs the tasks of parameter evaluation and substitution.

As a result, a modified input file that is free of parameter and parameter dependence table definitions and <*parameter*> instances is produced. This file is named *job-name*`.pes` and is subsequently submitted for execution of an analysis. The execution procedure of a parametrized input file, except for the additional processing of parameter shape variation definitions in the analysis input file processor, does not differ from that of a nonparametrized input file. All the files generated by the parametrized input job are named *job-name* with the appropriate extension appended to it.

Parameter Check Jobs

You can specify an execution mode in which only parameter processing (evaluation and substitution) is carried out. The parameter check execution mode is mutually exclusive of other execution modes, such as complete analysis, data check, continuation of a data check, conversion of results, or recovery (see *Abaqus/Standard and Abaqus/Explicit Execution*).

A parameter check run is useful in situations where you have defined complex parametrization in the input. In these cases you may want to study the results of parameter evaluation and substitution before proceeding further.

A parameter check run does not permit continuation of the execution in a subsequent run; the job must be rerun from the beginning.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **input=***input-file* **parametercheck**

Display of Parametric Input

Display of the results of parameter evaluation and substitution in the data file is described in this section. Visualization of parameter shape variations is described in *Parametric Shape Variation*.

Data File Display

The data (`.dat`) file contains information about the model definition generated by the analysis input file processor. You can control the amount of output generated by the analysis input file processor; see *Controlling the Amount of analysis input file processor Information Written to the Data File* for details. In particular, you can specify whether or not the original input (`.inp`) file is echoed to the data file (by default, it is not).

In the case of parametric input this file will generally contain a number of parameter, parameter dependence table, and parameter shape variation definitions, as well as a number of <*parameter*> instances. To verify the definition of parametric input, you can create a modified version of the original input file showing the parameters and their values (this file is named *job-name*`.par`). You can also create the *job-name*`.pes` file, which is the modified version of the original input file that is free of parameter and parameter dependence table definitions, as well as <*parameter*> instances.

Input File Usage: Use the following option to print the contents of the *job-name*`.par` file to the data file:

**PREPRINT*, PARVALUES=YES

Use the following option to print the contents of the *job-name*`.pes` file to the data file:

**PREPRINT*, PARSUBSTITUTION=YES

References

- Lutz, M., , and D. Ascher, Learning Python, O'Reilly & Associates, Inc. , 1999.

Spatial Modeling

In this section:

- *Node Definition*
- *Element Definition*
- *Surface Definition*
- *Rigid Body Definition*
- *Integrated Output Section Definition*
- *Mass Adjustment*
- *Nonstructural Mass Definition*
- *Distribution Definition*
- *Display Body Definition*
- *Assembly Definition*
- *Probability Density Function*

Node Definition

In this section:

- *Node Definition*
- *Parametric Shape Variation*
- *Nodal Thicknesses*
- *Normal Definitions at Nodes*
- *Transformed Coordinate Systems*
- *Adjusting Nodal Coordinates*

Node Definition

Products: Abaqus/Standard Abaqus/Explicit

References:

- **NCOPY*
- **NFILL*
- **NGEN*
- **NMAP*
- **NODE*
- **NSET*
- **SYSTEM*

Overview

This section describes the methods for defining nodes in an Abaqus input file. In a preprocessor such as Abaqus/CAE, you define the model geometry rather than the nodes and elements; when you mesh the geometry, the preprocessor automatically creates the nodes and elements needed for analysis. Although the concepts discussed in this section apply in general to the node definitions in the input file that is created by Abaqus/CAE, the methods and techniques described here apply only if you are creating the input file manually.

Node definition consists of:

- assigning a node number to the node;
- optionally specifying a local coordinate system in which to define nodes;
- defining individual nodes by specifying their coordinates;
- grouping nodes into node sets;
- creating nodes from existing nodes by generating them incrementally, by copying existing nodes, or by filling in nodes between the bounds of a region; and
- mapping a set of nodes from one coordinate system to another.

If any node is specified more than once, the last specification given is used.

Abaqus will eliminate all unnecessary nodes before proceeding with the analysis. This feature is useful because it allows points to be defined as nodes for mesh generation purposes only.

Assigning a Node Number to the Node

Each individual node must have a numeric label called the node number, which is assigned when the node is defined. The node number must be a positive integer, and the maximum node number allowed is 999999999 (for information on integer input, see *Input Syntax Rules*). The nodes do not need to be numbered continuously.

An Abaqus model can be defined in terms of an assembly of part instances (see *Assembly Definition*). In such a model all nodes must belong to either a part, part instance, or, in the case of reference nodes, to the assembly. Node numbers must be unique within a part, part instance, or the assembly; but they can be repeated in different parts or part instances.

Specifying a Local Coordinate System in Which to Define Nodes

Sometimes it is convenient to define nodal coordinates in a local coordinate system and then transform these coordinates to the global coordinate system. You can define a nodal coordinate system; Abaqus will translate and rotate the local (X^1, Y^1, Z^1) coordinate values into the global coordinate system. The transformation is done immediately after input and will be applied to all nodal coordinates entered or generated after the nodal coordinate system is defined.

The transformation affects only the input of nodal coordinates in node definitions. Nodal coordinate system definitions cannot be used:

- for applying loads and boundary conditions—see *Transformed Coordinate Systems* instead; or
- for output of components of stress, strain, and element section forces—see *Orientations* instead.

In addition to defining nodal coordinate systems, you can define individual nodes or node sets in local rectangular, cylindrical, or spherical systems (see *Specifying a Local Coordinate System for the Nodal Coordinates*). If a nodal coordinate system is in effect and you specify a local coordinate system for a particular node or node set definition, the input coordinates are first transformed according to the local system specified in the node definition and then according to the nodal coordinate system.

Defining the Nodal Coordinate System

You set up the coordinate system specification by specifying the global coordinates of three points in the local system: the origin of the local system (point *a* in *Figure 1*), a point on the local X^1-axis (point *b* in *Figure 1*), and a point in the (X^1, Y^1) plane of the local system on (or near) the local Y^1-axis (point *c* in *Figure 1*).

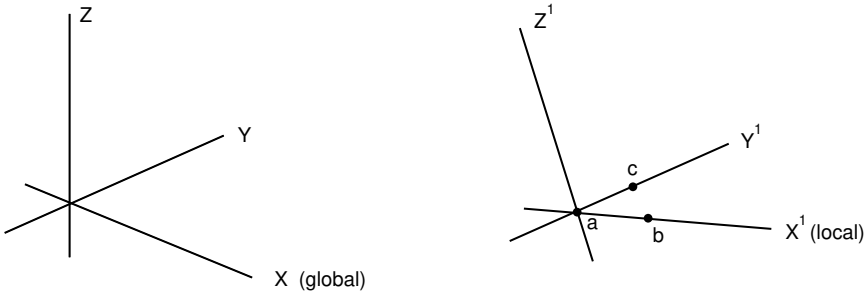


Figure 1: Nodal coordinate system.

If only one point (the origin) is given, Abaqus assumes that you need a translation only. If only two points are given, the direction of the Z^1-axis will be the same as that of the *Z*-axis; that is, the X^1-axis will be projected onto the (X, Y) plane.

To change the nodal coordinate system that is in effect, define another nodal coordinate system; to revert to input in the global coordinate system, use a nodal coordinate system definition without any associated data.

Input File Usage: Use the following option to define a nodal coordinate system:

**SYSTEM*
$X_a, Y_a, Z_a, X_b, Y_b, Z_b$
X_c, Y_c, Z_c

For example, in the following input, nodes 1 through 3 are defined in the first nodal coordinate system, nodes 4 and 5 are defined in the second nodal coordinate system, and nodes 6 and 7 are defined in the global coordinate system:

```
*SYSTEM
0, 0, 0, 5, 5, 5
*NODE
1, 0, 0, 1
2, 0, 0, 2
3, 0, 1, 2
*SYSTEM
2, 3, 4
*NODE
4, 0, 0, 1
5, 1, 4, 0
*SYSTEM
*NODE
6, 1, 0, 1
7, 0, 4, 2
```

Defining a Nodal Coordinate System within Part Definitions

When you define a nodal coordinate system within a part (or part instance) definition, it is in effect only within that part (or part instance) definition. Nodes defined in other parts are not affected.

You specify the local (X^1, Y^1, Z^1) coordinate values relative to the part coordinate system, which subsequently may be translated and/or rotated according to the positioning data given for the instance (see *Assembly Definition*).

Defining Individual Nodes by Specifying Their Coordinates

You can define individual nodes by specifying the node number and the coordinates that define the node. Abaqus uses a right-handed, rectangular Cartesian coordinate system for all nodes except for axisymmetric models, when the coordinates of the nodes must be given as the radial and axial positions. For more information about direction definitions, see *Conventions*.

In a model defined in terms of an assembly of part instances, give nodal coordinates in the local coordinate system of the part (or part instance). See *Assembly Definition*.

Input File Usage: **NODE*

Reading Node Definitions from a File

Node definitions can be read into Abaqus from an alternate file. The syntax of such file names is described in *Input Syntax Rules*.

Input File Usage: **NODE*, INPUT=*file_name*

Specifying a Local Coordinate System for the Nodal Coordinates

You can specify that a local rectangular Cartesian, cylindrical, or spherical coordinate system be used for a particular node definition. These coordinate systems are shown in *Figure 2*.

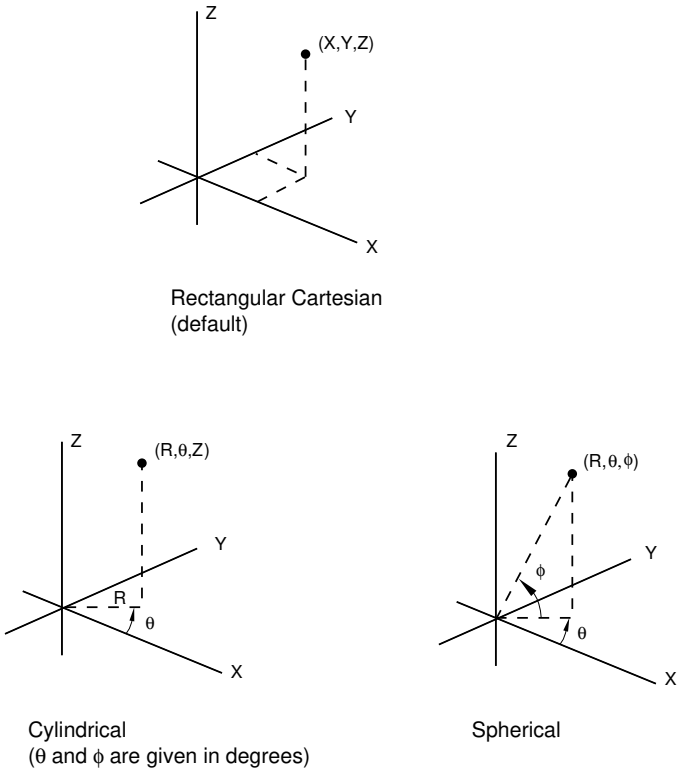


Figure 2: Coordinate systems.

This coordinate system specification is entirely local to the node definition. As the nodal data are read, the coordinates are transformed to rectangular Cartesian coordinates immediately. If a nodal coordinate system is also in effect (see *Specifying a Local Coordinate System in Which to Define Nodes*), these are local rectangular Cartesian coordinates as defined by the nodal coordinate system, which are subsequently transformed to global Cartesian coordinates.

Input File Usage:

Use the following option to specify the nodal coordinates in a rectangular Cartesian system (this is the default):

**NODE*, SYSTEM=R

Use the following option to specify the nodal coordinates in a cylindrical system:

**NODE*, SYSTEM=C

Use the following option to specify the nodal coordinates in a spherical system:

**NODE*, SYSTEM=S

For example, the following lines define node number 1 with coordinates (10cos20°, 10sin20°, 5.) in a local cylindrical system (R, θ, Z):

**NODE*, NSET=DISC, SYSTEM=C
1, 10., 20., 5.

If the following lines appeared in the input file before the above node definition, the coordinates of node 1 would be transformed first to rectangular Cartesian coordinates

in the nodal coordinate system defined by the **SYSTEM* option and then to coordinates in the global system:

```
*SYSTEM
2, 0, 2
```

Grouping Nodes into Node Sets

Node sets are used as convenient cross-references when defining loads, constraints, properties, etc. Node sets are the fundamental references of the model and should be used to assist the input definition. The members of a node set can be individual nodes or other node sets. An individual node can belong to several node sets.

Nodes can be grouped into node sets when they are created or after they have already been defined. In either case each node set is assigned a name. Node set names can be up to 80 characters long.

The same name can be used for a node set and for an element set.

By default, the nodes within a node set will be arranged in ascending order, and duplicate nodes will be removed. Such a set is called a sorted node set. You may choose to create an unsorted node set as described later, which is often useful for features that match two or more node sets. For example, if you define multi-point constraints (*General Multi-Point Constraints*) between two node sets, a constraint will be created between the first node in Set 1 and the first node in Set 2, then between the second node in Set 1 and the second node in Set 2, etc. It is important to ensure that the nodes are combined in the desired way. Therefore, it is sometimes better to specify that a node set be stored in unsorted order.

Once nodes are assigned to a node set, additional nodes can be added to the same node set; however, nodes cannot be removed from a node set.

Creating an Unsorted Node Set

You can choose to assign nodes to a new node set (or to add nodes to an existing node set) in the order in which they are given. The node numbers will not be rearranged, and duplicates will not be removed.

This unsorted node set will affect node copies, node fills, linear constraint equations, multi-point constraints, and substructure nodes associated with retained degrees of freedom. An unsorted node set can be created only by directly defining an unsorted node set as described here or by copying an unsorted node set. Any additions or modifications to a node set using other means will result in a sorted node set.

Input File Usage: **NSET*, NSET=*name*, UNSORTED

Assigning Nodes to a Node Set as They Are Created

There are several ways that nodes can be assigned to node sets as they are created.

Input File Usage: Use any of the following options:

**NODE*, NSET=*name*
**NCOPY*, NEW SET=*name*
**NFILL*, NSET=*name*
**NGEN*, NSET=*name*
**NMAP*, NSET=*name*

Assigning Previously Defined Nodes to a Node Set

You can assign nodes that you have defined previously (by specifying their coordinates, by filling in nodes between two bounds, or by generating them incrementally) to a node set by listing the nodes forming the set directly, by generating the node set, or by generating a node set from an element set.

Listing the Nodes That Define the Set Directly

You can list the nodes that form a node set directly. Previously defined node sets, as well as individual nodes, can be assigned to node sets.

Input File Usage: **NSET*, NSET=*name*

For example, the following lines add nodes 1, 3, 10, 11, and all the nodes in set `A11` to set `A12`:

```
*NSET, NSET=A12
1, 3
10, 11,
A11
```

Node set `A11` can be assigned to node set `A12` only if the definition of `A11` occurs before the definition of `A12`.

All the nodes in node set `A12` will be sorted into ascending numerical order. If the UNSORTED parameter were included on the **NSET* option, node set `A12` would contain the nodes in the order in which they are specified on the data lines.

Generating the Node Set

To generate a node set, you must specify a first node, n_1; a last node, n_2; and the increment in node numbers between these nodes, i. All nodes going from n_1 to n_2 in increments of i will be added to the set. Therefore, i must be an integer such that $(n_2 - n_1)/i$ is a whole number (not a fraction). The default is $i = 1$.

Input File Usage: **NSET*, NSET=*name*, GENERATE

For example, the following lines add all nodes from 100 to 120 in increments of 10 to set `A13`:

```
*NSET, NSET=A13, GENERATE
100, 120, 10
```

Generating a Node Set from an Element Set

You can specify the name of a previously defined element set (*Element Definition*), in which case the nodes that define the elements contained in this element set will be assigned to the specified node set. This method can be used only to define sorted node sets.

Input File Usage: **NSET*, NSET=*name*, ELSET=*name*

For example, the following lines add all nodes that define elements 50 and 100 (nodes 1, 2, 3, and 4) to node set `A14`:

*ELEMENT, TYPE=B21
50, 1, 2
100, 3, 4
*ELSET, ELSET=B1
50, 100
*NSET, NSET=A14, ELSET=B1

Element set `B1` can be assigned to node set `A14` since the definition of `B1` occurs before the definition of `A14`.

Generating a Node Set from a Surface

You can specify the name of a previously defined surface (*About Surfaces*), in which case the nodes included in the surface will be assigned to the specified node set. This method can be used only to define sorted node sets.

Input File Usage: **NSET*, NSET=*name*, SURFACE=*name*

For example, the following lines add all nodes that are included in the surface named `surf_top` to node set `A15`:

*SURFACE, NAME=surf_top
*NSET, NSET=A15, SURFACE=surf_top

The surface named `surf_top` can be assigned to node set `A15` since the definition of `surf_top` occurs before the definition of `A15`.

Limitation on Updating Node Sets That Are Used to Define Other Node Sets

If a node set is constructed from previously defined node sets, subsequent updates to these sets are not taken into account.

Input File Usage: **NSET*, NSET=*name*

For example, the following lines add nodes 1 and 2, but not 3, to the set `SET-AB` while adding nodes 1 and 3 to set `SET-A`:

*NSET, NSET=SET-A
1,
*NSET, NSET=SET-B
2,
*NSET, NSET=SET-AB
SET-A, SET-B
*NSET, NSET=SET-A
3,

Defining Part and Assembly Sets

In a model defined in terms of an assembly of part instances, all node sets must be defined within a part, part instance, or the assembly definition. If a node set is defined within a part (or part instance) definition, you can refer to the node numbers directly. To define an assembly-level node set, you must identify the nodes to be added to the set by prefixing each node number with the part instance name and a "." (as explained in *Assembly Definition*). An assembly-level node set can have the same name as a part-level node set.

Example

The following input defines a node set, `set1`, that belongs to part `PartA` and will be inherited by every instance of `PartA`:

```
*PART, NAME=PartA
   ...
   *NSET, NSET=set1
    1,3,26,500
*END PART
```

A node set with the same name is defined at the assembly level as follows:

```
*ASSEMBLY, NAME=Assembly-1
   *INSTANCE, NAME=PartA-1, PART=PartA
    ...
   *END INSTANCE
   *INSTANCE, NAME=PartA-2, PART=PartA
    ...
   *END INSTANCE
   *NSET, NSET=set1
    PartA-1.1, PartA-1.3, PartA-1.26, PartA-1.500
    PartA-2.1, PartA-2.3, PartA-2.26, PartA-2.500
*END ASSEMBLY
```

Assembly-level node set `set1` contains all the nodes from node sets `set1` belonging to part instances `PartA-1` and `PartA-2`. Therefore, the nodes are assigned to two separate node sets: one at the part instance level and one at the assembly level. An assembly-level node set called `set1` could be created with entirely different nodes than those that belong to the part set; part- and assembly-level node sets are independent. However, since in this example the same nodes are assigned to both the part- and assembly-level node sets `set1`, the assembly-level set could alternatively be defined by

```
*ASSEMBLY, NAME=Assembly-1
   *INSTANCE, NAME=PartA-1, PART=PartA
    ...
   *END INSTANCE
   *INSTANCE, NAME=PartA-2, PART=PartA
    ...
   *END INSTANCE
   *NSET, NSET=set1
    PartA-1.set1, PartA-2.set1
*END ASSEMBLY
```

This node set definition is equivalent to the previous example, where the nodes are listed individually.

Alternate Method for Defining Assembly-Level Node Sets

Sometimes it is not convenient to define an assembly-level node set by referring to part-level node sets. In such cases a set definition containing many nodes can get quite lengthy. Therefore, an alternate method is provided.

Input File Usage: **NSET*, NSET=*NsetName*, INSTANCE=*InstanceName*

The following example shows two equivalent ways to define an assembly-level node set; once by prefixing each node number with a part instance name (as shown above) and once using the more compact INSTANCE notation:

```
*ASSEMBLY, NAME=Assembly-1
  *INSTANCE, NAME=PartA-1, PART=PartA
   ...
  *END INSTANCE
  *INSTANCE, NAME=PartA-2, PART=PartA
```

```
 ...
*END INSTANCE
*NSET, NSET=set2
 PartA-1.11, PartA-1.12, PartA-1.13, PartA-1.14,
 PartA-2.21, PartA-2.22, PartA-2.23, PartA-2.24
*NSET, NSET=set3, INSTANCE=PartA-1
 11, 12, 13, 14
*NSET, NSET=set3, INSTANCE=PartA-2
 21, 22, 23, 24
*END ASSEMBLY
```

When the **NSET* option is used more than once with the same name, as it is with `set3`, the nodes in the second use of **NSET* are appended to the set created by the first use of **NSET*.

Internal Node Sets Created by Abaqus/CAE

In Abaqus/CAE many modeling operations are performed by picking geometry with the mouse. For example, a concentrated load can be applied by picking a point on a geometric part instance. Since the **CLOAD* option refers to a node set, this "picked" geometry must be translated into a node set in the input file. Such sets are assigned a name by Abaqus/CAE and marked as internal. You can view these internal sets using display groups in the Visualization module of Abaqus/CAE (see *Using display groups to display subsets of your model*).

Input File Usage: **NSET*, NSET=*NsetName*, INTERNAL

Transferring of Node Sets

If the results of an Abaqus/Explicit analysis are imported into an Abaqus/Standard analysis (or vice versa) or results from an Abaqus/Standard analysis are imported into another Abaqus/Standard analysis (see *About Transferring Results between Abaqus Analyses*), all node set definitions in the original analysis are imported by default. Alternatively, you can import only selected node set definitions; see *Importing Element Set, Node Set, and Surface Definitions One Time* for details.

If a three-dimensional model is generated from a symmetric model (see *Symmetric Model Generation*), all node sets in the original model will be used (and expanded) in the generated model.

Creating Nodes from Existing Nodes by Generating Them Incrementally

You can generate nodes incrementally from existing nodes. All of the nodes along a straight or curved line can be generated by giving the coordinates of the two end nodes and defining the type of curve.

The two end nodes must already be defined, usually by specifying their coordinates, but it is also possible to have them defined by an earlier generation.

Defining a Straight Line between the Two End Nodes

To define a straight line between the two end nodes, specify the number of the first end node, n_1; the number of the last end node, n_2; and the increment in node numbers between each node along the line, i. Therefore, i must be an integer such that $(n_2 - n_1)/i$ is a whole number (not a fraction). The default is $i = 1$.

Input File Usage: **NGEN*

For example, in the following input node number 1 with coordinates (0., 0., 0.) and node number 6 with coordinates (10., 0., 0.) are defined and nodes 2, 3, 4, and 5 with coordinates (2., 0., 0.), (4., 0., 0.), (6., 0., 0.), and (8., 0., 0.), respectively, are generated automatically:

```
*NODE
1, 0., 0., 0.
6, 10., 0., 0.
*NGEN
1, 6, 1
```

Defining a Circular Arc between the Two End Nodes

To define a circular arc between the two end nodes, specify the number of the first end node, n_1; the number of the last end node, n_2; and the increment in node numbers between each node along the arc, i. Therefore, i must be an integer such that $(n_2 - n_1)/i$ is a whole number (not a fraction). The default is $i = 1$.

In addition, you must specify the coordinates of one extra point, the center of the circle, either by giving the node number of a node that has already been defined or by giving the nodal coordinates directly. If both are supplied, the node number will take precedence over the coordinates.

If the coordinates are defined directly, they can be specified in a local coordinate system as described later.

The coordinates of the end nodes will be adjusted radially if the circle cannot be passed through both points. An arc of a circle of 180° through 360° will require more extensive definition. For this case you must define the plane of the circular disc by giving the normal to the disc; the nodes will then be numbered according to the right-hand rule about this normal.

Input File Usage: **NGEN*, LINE=C

Defining a Parabola between the Two End Nodes

To define a parabola between the two end nodes, specify the number of the first end node, n_1; the number of the last end node, n_2; and the increment in node numbers between each node along the parabola, i. Therefore, i must be an integer such that $(n_2 - n_1)/i$ is a whole number (not a fraction). The default is $i = 1$.

In addition, you must specify the coordinates of one extra point, the midpoint on the arc between the two end points, either by giving the node number of a node that has already been defined or by giving the nodal coordinates directly. If both are supplied, the node number will take precedence over the coordinates.

If the coordinates are defined directly, they can be specified in a local coordinate system as described later.

Input File Usage: **NGEN*, LINE=P

Defining the Extra Point and the Normal Direction in a Local Coordinate System

You can specify the coordinates of the extra point that is required for a circle or a parabola in a local rectangular Cartesian system, a cylindrical system, or a spherical system. These coordinate systems are shown in *Figure 2*.

If a nodal coordinate system is in effect (see *Specifying a Local Coordinate System in Which to Define Nodes*), the coordinates and normal direction specified in the node definition are assumed to be in the nodal coordinate system. If a nodal coordinate system is in effect and you specify the extra point for a circle or parabola in a local coordinate system, the input is first transformed according to the local system specified in the node definition and subsequently according to the nodal coordinate system.

Input File Usage: Use the following option to specify the extra point in a rectangular Cartesian system (this is the default):

**NGEN*, SYSTEM=RC

Use the following option to specify the extra point in a cylindrical system:

**NGEN*, SYSTEM=C

Use the following option to specify the extra point in a spherical system:

**NGEN*, SYSTEM=S

Creating Nodes by Copying Existing Nodes

You can create new nodes by copying existing nodes. The coordinates of the new nodes can be translated and rotated, reflected from the nodes being copied, or projected from the nodes being copied by using a polar projection with respect to a pole node.

You must identify the existing node set to copy and specify an integer constant, *n*, that will be added to the node numbers of existing nodes to define node numbers for the nodes being created.

You can assign the newly created nodes to a node set. If you do not specify a node set name for the newly created nodes, they are not assigned to a node set.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, NEW SET=*new_name*

Translating and Rotating the Coordinates of the Old Nodes

You can create new nodes by translating and/or rotating the nodes in the old node set (see *Figure 3*). You specify the value of the translation in the *X*-, *Y*-, and *Z*-directions.

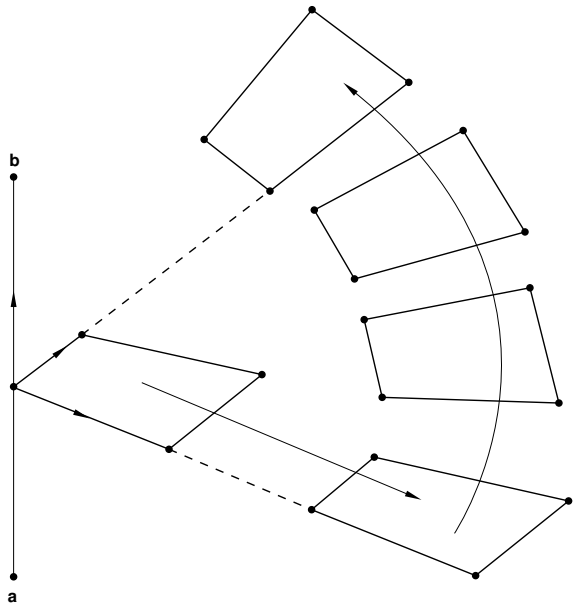


Figure 3: Translation and rotation of existing nodes.

In addition, you specify the coordinates of the first point defining the rotation axis (point *a* in *Figure 3*), the coordinates of the second point defining the rotation axis (point *b* in *Figure 3*), and the angle of rotation (in degrees) about the *a*–*b* axis. The rotation can be applied multiple times as described later.

If you specify both translation and rotation, the translation is applied once before the rotation.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, SHIFT

Applying the Rotation Multiple Times

You can specify the number of times the rotation should be applied, *m*. For example, if nodes are to be created at angles of 30°, 60°, and 90°, set *m*=3. The identifiers of the nodes created are incremented sequentially by the value of *n*, as described above.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, SHIFT, MULTIPLE=*m*

Reflecting the Coordinates of the Old Nodes

You can create new nodes by reflecting the coordinates of the old nodes through a line, a plane, or a point.

Reflecting the Coordinates through a Line

To reflect the old nodal coordinates through a line, you specify the coordinates of points *a* and *b* (see *Figure 4*).

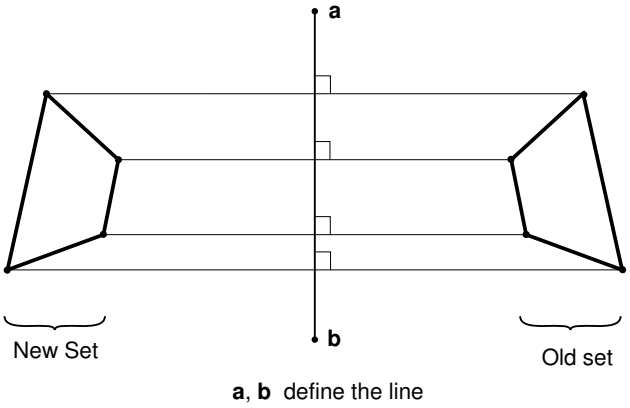


Figure 4: Reflection of coordinates through a line.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, REFLECT=LINE

Reflecting the Coordinates through a Plane

To reflect the old nodal coordinates through a plane, you specify the coordinates of points *a*, *b*, and *c* (see *Figure 5*).

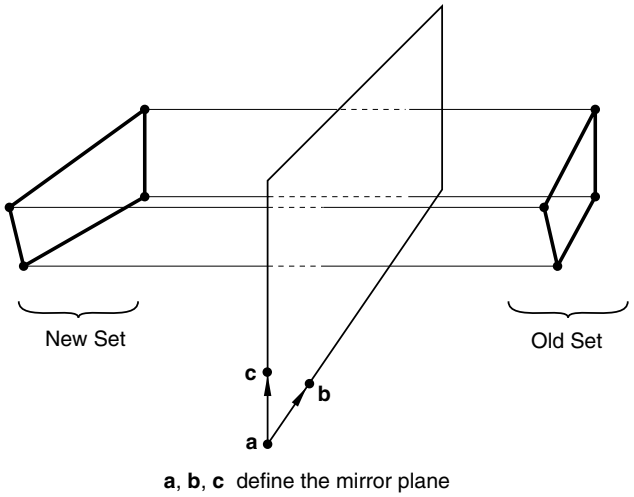


Figure 5: Reflection of coordinates through a plane.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, REFLECT=MIRROR

Reflecting the Coordinates through a Point

To reflect the old nodal coordinates through a point, you specify the coordinates of point *a* (see *Figure 6*).

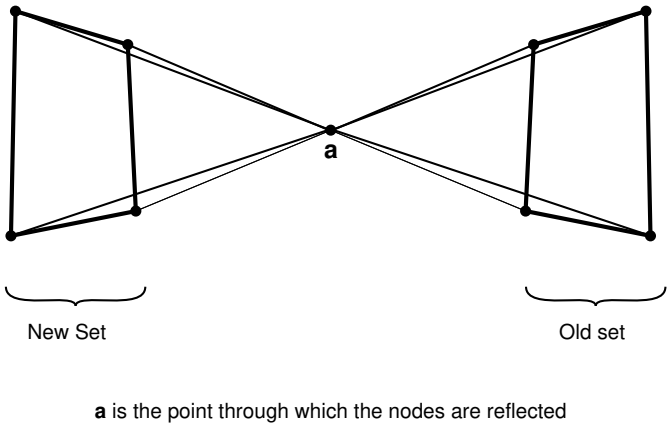


Figure 6: Reflection of coordinates through a point.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, REFLECT=POINT

Projecting the Nodes in the Old Set from a Pole Node

You can create new nodes by projecting the nodes in the old set from a pole node. Each new node will be located such that the corresponding old node is equidistant between the pole node and the new node. The pole node (see *Figure 7*) is identified by giving its number or, alternatively, its coordinates.

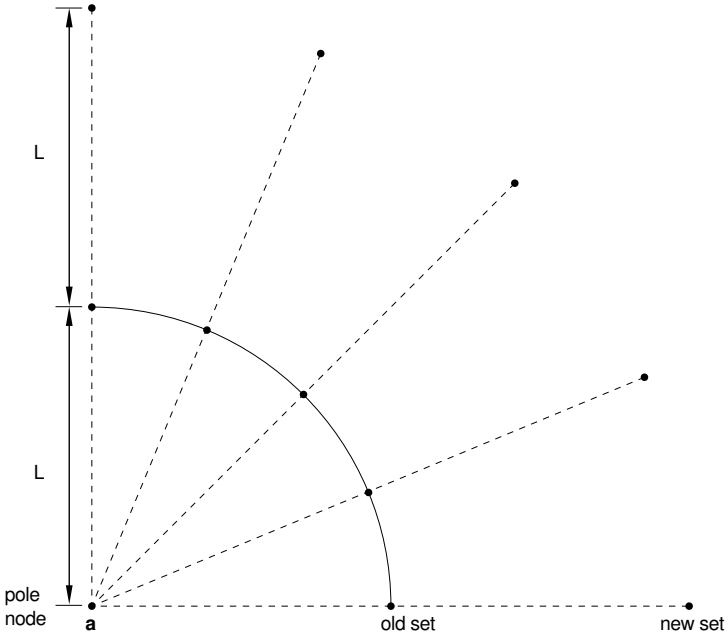


Figure 7: Projection of existing nodes from a pole node.

This method is particularly useful for creating nodes that are associated with infinite elements (*Infinite Elements*). In this case the pole node should be located at the center of the far-field solution.

Input File Usage: **NCOPY*, OLD SET=*name*, CHANGE NUMBER=*n*, POLE

Creating Nodes by Filling in Nodes between Two Bounds

You can create nodes by filling in nodes between two bounds. In this case you specify the two node sets whose members form the bounds, the number of intervals along each line between the bounding nodes, and the increment in node numbers from the node number at the first bound set end.

Let l equal the number of lines of nodes to be created between the two bounding node sets; the number of intervals along each line between the bounding nodes is then given by $l + 1$.

Let n equal the increment in node numbers from the node number at the first bound set end; for each node (n_{A_i}) in the first bounding node set, the corresponding node in the other bounding node set (n_{B_i}) must be numbered such that $(n_{B_i} - n_{A_i})/n$ is a whole number.

The node sets that define the bounds of the region are used as they exist at the time the node fill definition appears in the input file: only those nodes that have been added to the sets prior to the node fill definition are used. Both sorted and unsorted node sets can be used. Nodes that have not yet been given coordinates are assumed to be at the origin, (0.,0.,0.).

The nodes created by this method lie on straight lines between corresponding nodes in the two sets. If the sets do not have the same number of nodes, the extra nodes in the longer set are ignored. By default, the spacing between nodes along the lines is uniform.

Input File Usage: **NFILL*

Example

For example, *Figure 8* shows a simple quarter-cylinder model.

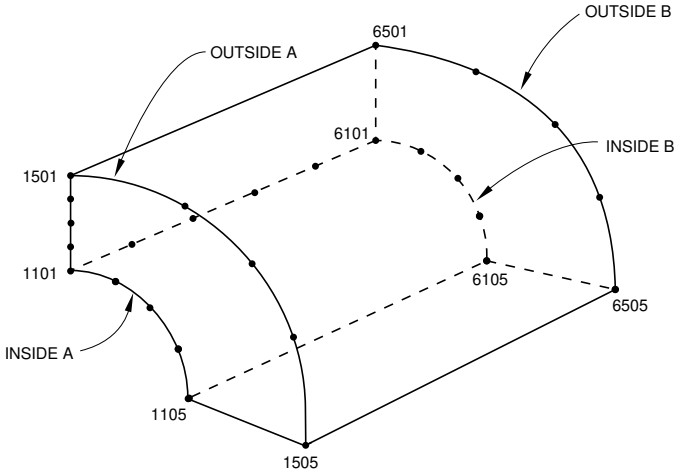


Figure 8: Filling a three-dimensional region.

The quarter circles `INSIDEA` (nodes 1101–1105), `OUTSIDEA` (nodes 1501–1505), `INSIDEB` (nodes 6101–6105), and `OUTSIDEB` (6501–6505) have already been defined by specifying their coordinates directly or generating them incrementally. The region is filled by first filling the end planes and placing the nodes on those planes into sets `A` and `B` and then filling between those sets with the following options:

```
*NFILL, NSET=A
INSIDEA, OUTSIDEA, 4, 100
*NFILL, NSET=B
INSIDEB, OUTSIDEB, 4, 100
*NFILL
A, B, 5, 1000
```

Concentrating the Nodes toward One Bound or the Other

You can concentrate the nodes toward one bound or the other by specifying b, the ratio of adjacent distances between nodes along each line of nodes generated as the nodes go from the first bounding node set to the second.

Thus, if b is less than one, the nodes are concentrated toward the first bounding node set; if b is greater than one, the nodes are concentrated toward the second bounding set. The value of b must be positive.

The bias intervals along the line from the first bounding node are L, L/b, L/b^2, L/b^3, L/b^4, L/b^5, … (where L is the length of the first interval). In Abaqus/Standard the bias value can be applied at every interval along the line or at every second interval along the line as described later.

Input File Usage: **NFILL*, BIAS=*b*

Example

For example, suppose the lines of nodes shown in *Figure 9* have already been generated by other methods and placed into node sets `INSIDE` and `OUTSIDE`.

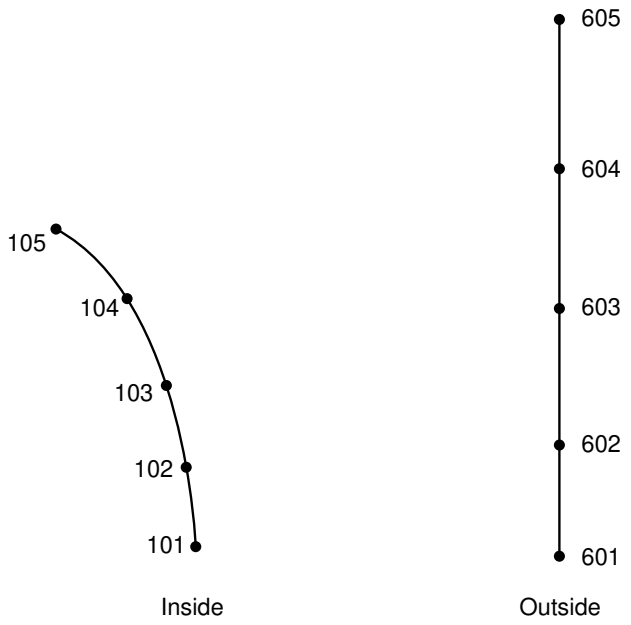


Figure 9: Node sets defining bias example.

The following option will fill the region as shown in *Figure 10*:

```
*NFILL, BIAS=0.6
INSIDE, OUTSIDE, 5, 100
```

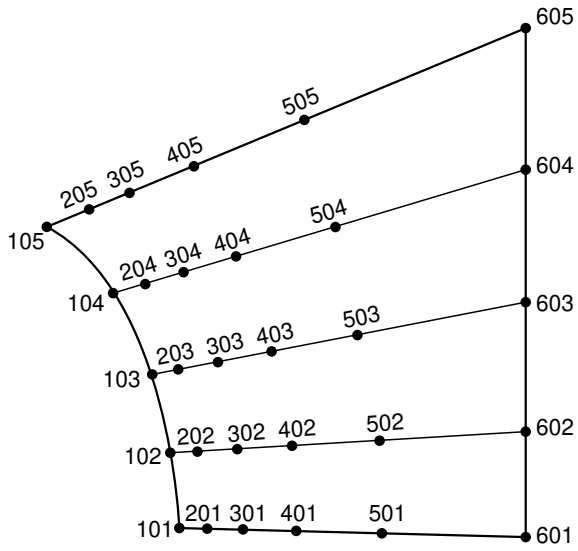


Figure 10: Result of bias example.

Applying the Bias Value at Every Second Interval along the Line

In Abaqus/Standard you can apply the bias value at every second interval along the line. In this case the nodes will be positioned along the line correctly for use with second-order elements, so that the midside nodes are at the middle of the interval between the corner nodes of the elements.

The bias intervals along the line from the first bounding node are L, L, L/b, L/b, L/b^2, L/b^2, … (where L is the length of the first interval).

Input File Usage: **NFILL*, BIAS=*b*, TWO STEP

Creating Quarter-Point Spacing

In Abaqus/Standard you can create quarter-point spacing for fracture mechanics calculations with second-order isoparametric elements (*About Fracture Mechanics*). This spacing gives a square root singularity in the strain field at the crack tip by placing the first node away from that point at one-quarter of the distance to the second point. The remaining nodes on each line are spaced so that the size of the elements will grow as the square of the distance from the singularity, with the midside nodes exactly at the midsides of the elements. This spacing produces a reasonable mesh gradation for this type of problem; however, better results can be obtained for crude meshes by making the size of the crack element smaller than the quarter-point spacing technique does.

Input File Usage: **NFILL*, SINGULAR

Example

Figure 11 shows a simple fracture mechanics example.

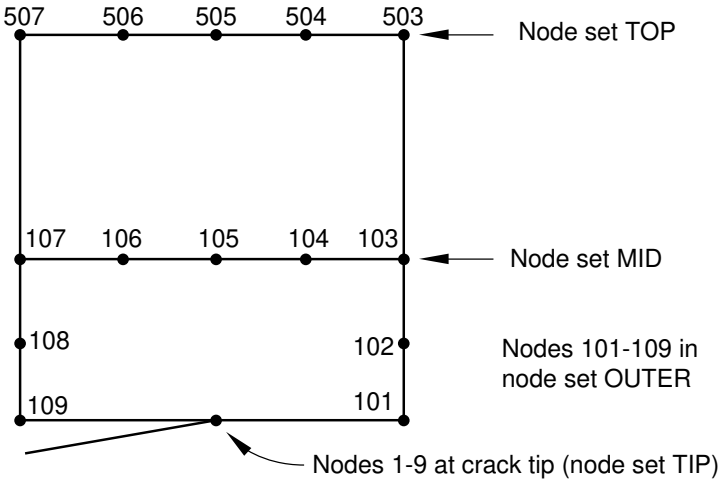


Figure 11: Node fill used in a singular problem.

(The mesh shown is very coarse, and a finer mesh would probably be used in an actual case.) The nodes on the top edge have been placed in node set `TOP`, those on the horizontal line at the upper end of the focused region are in node set `MID`, all the nodes around the focused region are in node set `OUTER`, and there are multiple nodes at the crack tip in node set `TIP`. The following options are used to fill in the region as shown in *Figure 12* (note the quarter-point nodes adjacent to the crack tip):

```
*NFILL, BIAS=0.8
MID, TOP, 4, 100
*NFILL, SINGULAR=1
TIP, OUTER, 5, 20
```

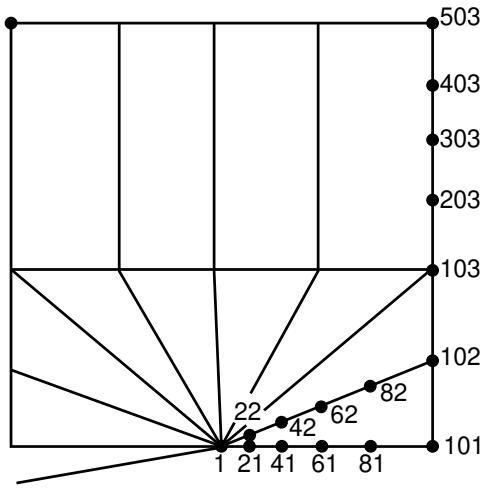


Figure 12: Node fill used in a singular problem.

Mapping a Set of Nodes from One Coordinate System to Another

You can map a set of nodes from one coordinate system to another. You can also rotate, translate, or scale the nodes in a set using a more direct method instead of coordinate system mapping. These capabilities are useful for many geometric situations: a mesh can be generated quite easily in a local coordinate system (for example, on the surface of a cylinder) using other methods and then can be mapped into the global (X, Y, Z) system. In other cases some parts of your model need to be translated or rotated along a given axis or scaled with respect to one point.

The mapping capability cannot be used in a model defined in terms of an assembly of part instances.

The following different mappings are provided: a simple scaling; a simple shift and/or rotation; skewed Cartesian; cylindrical; spherical; toroidal; and, in Abaqus/Standard only, blended quadratic. The first five of these mappings are shown in *Figure 13*.

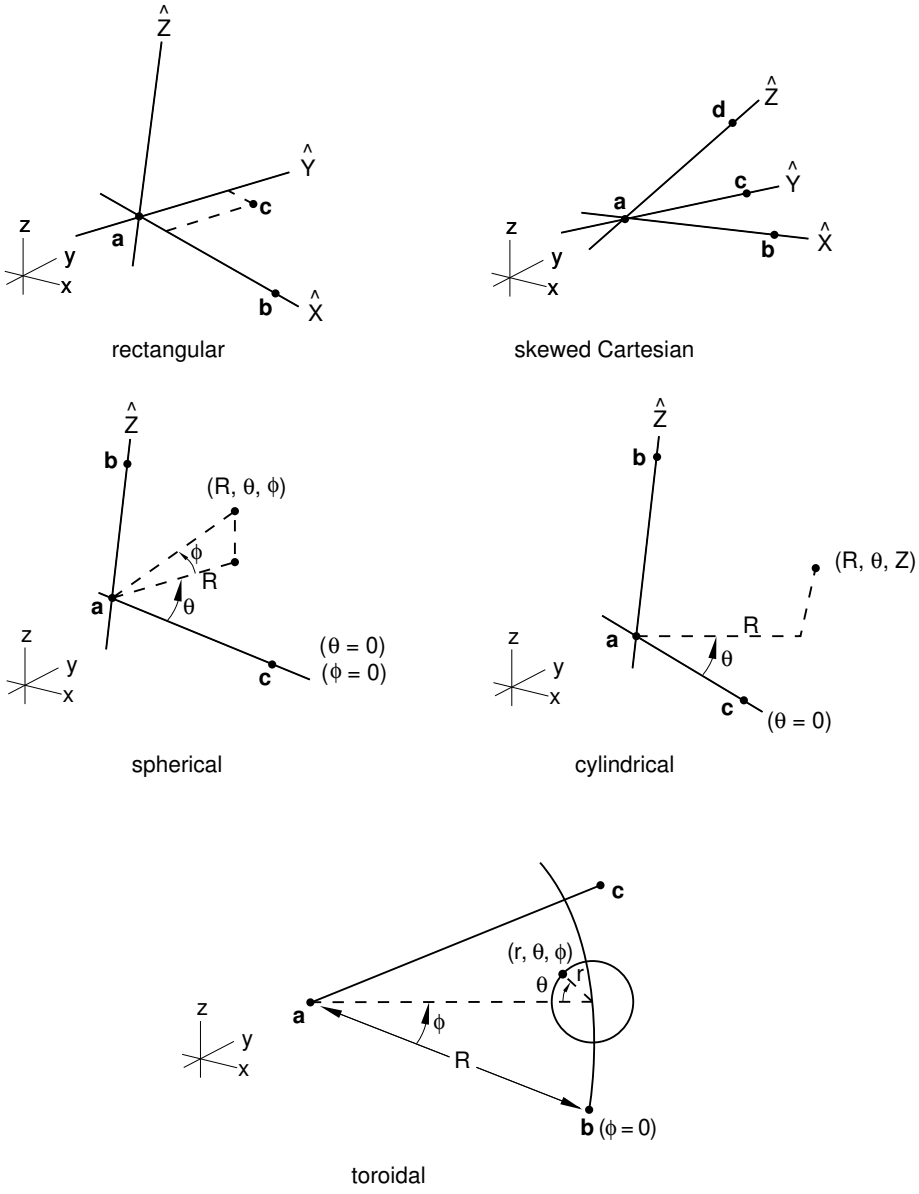


Figure 13: Coordinate systems; angles are in degrees.

Blended quadratic mapping is shown in *Figure 14*.

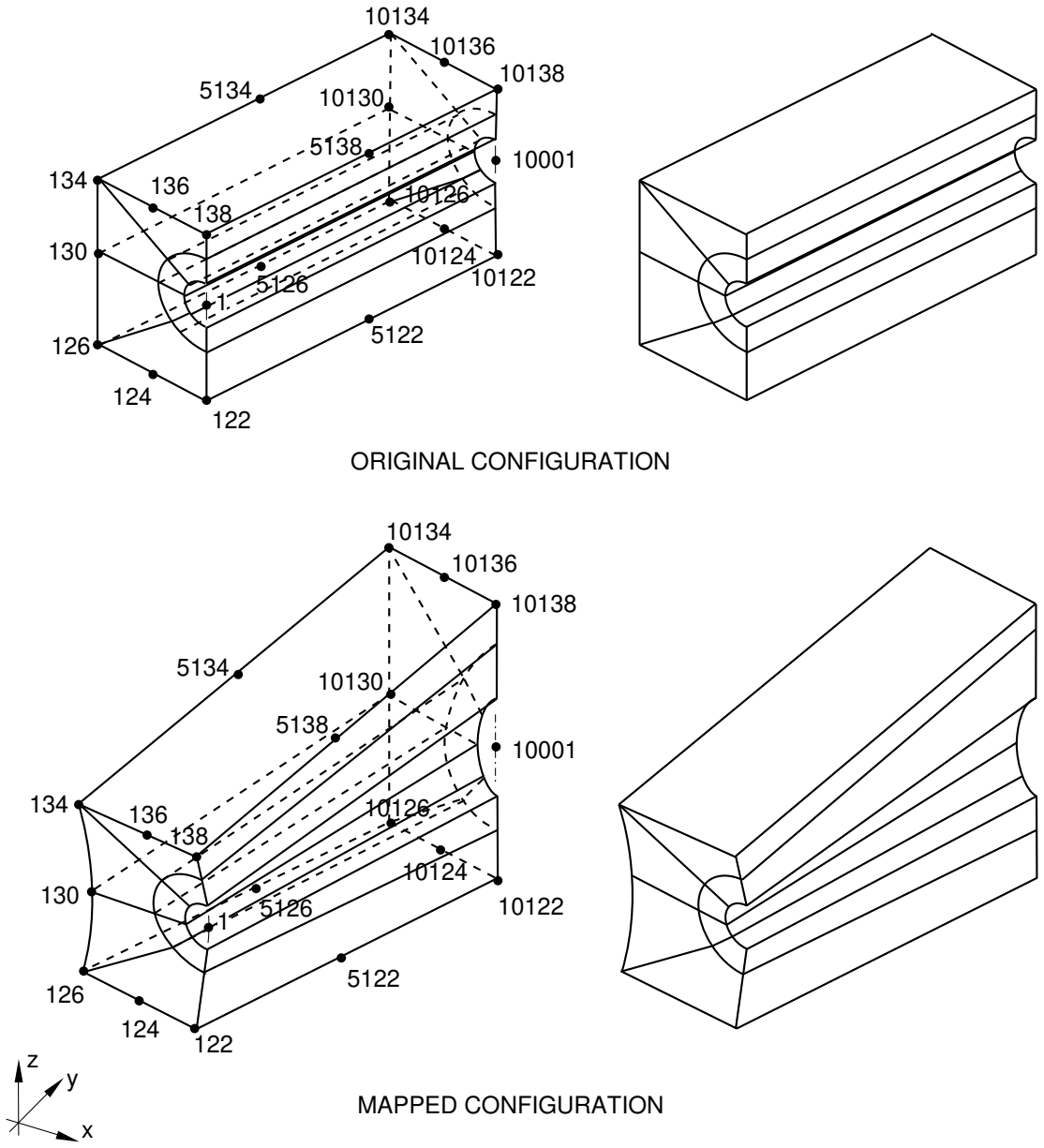


Figure 14: Use of blended quadratic mapping to develop a solid mesh onto a curved block.

In all cases the coordinates of the nodes in the set are assumed to be defined in the local system: these local coordinates at each node are replaced with the global Cartesian (X, Y, Z) coordinates defined by the mapping. All angular coordinates should be given in degrees.

You can use either coordinates or node numbers to define the new coordinate system, the axis of rotation and translation, or the reference point used for scaling.

The mapping capability can be used several times in succession on the same nodes, if required.

Scaling the Local Coordinates before They Are Mapped

For all mappings except the blended quadratic mapping, you can specify a scaling factor to be applied to the local coordinates before they are mapped.

This facility is useful for "stretching" some of the coordinates that are given. For example, in cases where the local system uses some angular coordinates and some distance coordinates (cylindrical, spherical, etc.), it may be preferable to generate the mesh in a system that uses distance measures in the angular directions and then scale onto the angular coordinate system for the mapping.

Two different scaling methods are available.

Specifying the Scaling Factors Directly

A first method of scaling the nodes with respect to the origin of the local system is to specify the scale factors directly. In this case the scaling is done at the same time as the mapping from one coordinate system to another.

Input File Usage: **NMAP*, NSET=*name*
first data line
second data line
scale factor for first local coord, scale factor for second local coord,
scale factor for third local coord

Specifying the Scaling with Respect to a Reference Point

Alternatively, you can scale with respect to a point other than the origin. The reference point with respect to which the scaling is done can be defined by using either its coordinates or the user node number.

Input File Usage: Use the following option to define the scaling reference point by using its coordinates (default):

**NMAP*, TYPE=SCALE, DEFINITION=COORDINATES
X-coordinate of reference point, Y-coordinate of reference point,
Z-coordinate of reference point
scale factor for first local coord, scale factor for second local coord,
scale factor for third local coord

Use the following option to define the scaling reference point by using its node number:

**NMAP*, TYPE=SCALE, DEFINITION=NODES
Local node number of the reference point
scale factor for first local coord, scale factor for second local coord,
scale factor for third local coord

Introducing a Simple Shift and/or Rotation by Mapping from One Coordinate System to Another

In the case of a simple shift and/or rotation, point *a* in *Figure 13* defines the origin of the local rectangular coordinate system defining the map. The local $\hat{x}$-axis is defined by the line joining points *a* and *b*. The local $\hat{x}$–$\hat{y}$ plane is defined by the plane passing through points *a*, *b*, and *c*.

Input File Usage: **NMAP*, NSET=*name*, TYPE=RECTANGULAR

Introducing a Pure Shift by Specifying the Axis and Magnitude of the Translation

You can define a pure translation (or shift) to move a set of nodes by a prescribed value along a desired axis. You must specify the axis of translation by providing either the coordinates or the two node numbers defining this axis, and you must prescribe the magnitude of the translation.

Input File Usage: Use the following option to specify the axis of translation using coordinates (default):

**NMAP*, NSET=*name*, TYPE=TRANSLATION,
DEFINITION=COORDINATES

Use the following option to specify the axis of translation using node numbers:

**NMAP*, NSET=*name*, TYPE=TRANSLATION, DEFINITION=NODES

Introducing a Pure Rotation by Specifying the Axis, Origin, and Angle of the Rotation

You can define a rotation of a set of nodes by providing the axis of rotation, the origin of rotation, and the magnitude of the rotation. You must specify the axis of rotation by providing either the coordinates or the two node numbers defining this axis. You must specify the origin of the rotation by providing either the coordinates or the node number at the origin of rotation. Finally, you must specify the angle of the rotation in degrees.

Input File Usage: Use the following option to specify the axis of rotation using coordinates (default):

**NMAP*, NSET=*name*, TYPE=ROTATION,
DEFINITION=COORDINATES

Use the following option to specify the axis of rotation using node numbers:

**NMAP*, NSET=*name*, TYPE=ROTATION, DEFINITION=NODES

Mapping from Cylindrical Coordinates

For mapping from cylindrical coordinates, point *a* in *Figure 13* defines the origin of the local cylindrical coordinate system defining the map. The line going through point *a* and point *b* defines the $\hat{\boldsymbol{z}}$-axis of the local cylindrical coordinate system. The local $\hat{\boldsymbol{r}}$–$\hat{\boldsymbol{z}}$ plane for $\boldsymbol{\theta} = \mathbf{0}$ is defined by the plane passing through points *a*, *b*, and *c*.

Input File Usage: **NMAP*, NSET=*name*, TYPE=CYLINDRICAL

Mapping from Skewed Cartesian Coordinates

For mapping from skewed Cartesian coordinates, point *a* in *Figure 13* defines the origin of the local diamond coordinate system defining the map. The line going through point *a* and point *b* defines the $\hat{\boldsymbol{x}}$-axis of the local coordinate system. The line going through point *a* and point *c* defines the $\hat{\boldsymbol{y}}$-axis of the local coordinate system. The line going through point *a* and point *d* defines the $\hat{\boldsymbol{z}}$-axis of the local coordinate system.

Input File Usage: **NMAP*, NSET=*name*, TYPE=DIAMOND

Mapping from Spherical Coordinates

For mapping from spherical coordinates, point *a* in *Figure 13* defines the origin of the local spherical coordinate system defining the map. The line going through point *a* and point *b* defines the polar axis of the local spherical coordinate system. The plane passing through point *a* and perpendicular to the polar axis defines the $\boldsymbol{\phi} = \mathbf{0}$ plane. The plane passing through points *a*, *b*, and *c* defines the local $\boldsymbol{\theta} = \mathbf{0}$ plane.

Input File Usage: **NMAP*, NSET=*name*, TYPE=SPHERICAL

Mapping from Toroidal Coordinates

For mapping from toroidal coordinates, point *a* in *Figure 13* defines the origin of the local toroidal coordinate system defining the map. The axis of the local toroidal system lies in the plane defined by points *a*, *b*, and *c*. The *R*-coordinate of the toroidal system is defined by the distance between points *a* and *b*. The line between points *a* and *b* defines the $\phi = 0$ position. For every value of ϕ the θ-coordinate is defined in a plane perpendicular to the plane defined by the points *a*, *b*, and *c* and perpendicular to the axis of the toroidal system. $\theta = 0$ lies in the plane defined by the points *a*, *b*, and *c*.

Input File Usage: **NMAP*, NSET=*name*, TYPE=TOROIDAL

Mapping by Means of Blended Quadratics

To map by means of blended quadratics in Abaqus/Standard, you define the new (mapped) coordinates of up to 20 “control nodes”: these are the corner and midedge nodes of the block of nodes being mapped. The mapping in this case is like that of a 20-node brick isoparametric element. Any of the midedge nodes can be omitted, thus allowing linear interpolation along that edge of the block. Abaqus/Standard does not check whether the nodes in the set lie within the physical space of the block defined by the corner and midedge nodes: these control nodes simply define mapping functions that are then applied to all the nodes in the set.

The control nodes should define a “well”-shaped block; for example, midedge nodes should be close to the midpoint of the edge. Otherwise, the mapping can be very distorted. For example, the nodes of a crack-tip 20-node element with midside nodes at the quarter points will not map correctly and, therefore, should not be used as the control nodes.

Blended mapping is available only for three-dimensional analyses.

Input File Usage: **NMAP*, NSET=*name*, TYPE=BLENDED

Parametric Shape Variation

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Parametric Modeling*
- **PARAMETER SHAPE VARIATION*

Overview

Shape parametrization can be accomplished in an Abaqus input file by:

- parametrizing nodal coordinates; or
- relating nodal coordinates to shape parameters using shape variations.

The different approaches to shape parametrization are described in this section.

Parametrization of Nodal Coordinates

Any individual nodal coordinates can be parametrized directly. This is usually of limited value because it often leads to designs with irregular shape that cannot be manufactured easily. In addition, parametrization of individual nodal coordinates generally requires an excessive number of parameters to define the parametrized shape.

Parametrization of nodal coordinates used in conjunction with node generation in Abaqus provides a more practical method of shape parametrization. However, this method is still of somewhat limited practical use because the simple node generation capabilities available in Abaqus cannot describe complex shapes.

Direct Parametrization of Individual Nodal Coordinates

The simplest form of parametrization of nodal coordinates is to define individual parameters and use them in place of the nodal coordinates to be parametrized, as described in *Parametric Modeling*. For example,

```
*PARAMETER
x_coord_node_1 = 10.
y_coord_node_1 = 20.
*NODE
1, <x_coord_node_1>, <y_coord_node_1>
```

Parametrization of Nodal Coordinates Using Node Generation

Shape parametrization can be accomplished by parametrizing the coordinates of some nodes, then using these nodes to generate other nodes and their coordinates. For example:

```
*PARAMETER
x_coord_node_1 = 10.
x_coord_node_11 = 20.
*NODE
1, <x_coord_node_1>, 50.
11, <x_coord_node_11>, 50.
*NGEN
1, 11, 1
```

This method of shape parametrization reduces the number of user-defined parameters necessary for shape parametrization by implicitly making the nodal coordinates of the generated nodes dependent on the shape parameters.

Shape Change by Linear Combination of Shape Variations

The definition of shape in Abaqus includes a basic shape plus any number of additional shape variations that are added to the basic shape using a linear combination. Mathematically, we can express the nodal coordinates, $\boldsymbol{x}$, as

$$\boldsymbol{x} = \boldsymbol{x}_0 + \sum_{i=1}^{N} \boldsymbol{s}_i \ \boldsymbol{p}_i,$$

where $\boldsymbol{x}_0$ is the basic shape, $\boldsymbol{s}_i$ is the $\boldsymbol{i}^{th}$ shape variation, and $\boldsymbol{p}_i$ is the value of the $\boldsymbol{i}^{th}$ shape parameter. This calculation is always done in the global rectangular Cartesian coordinate system. Although it is not necessarily so, it is frequently the case that the input to define a shape variation is simply the gradient of the basic shape $\boldsymbol{x}_0$ taken with respect to the corresponding shape parameter.

You specify the basic shape of a model in the Abaqus input file by providing nodal definitions either directly or through node generation; see *Node Definition*.

You can specify shape variations and associated shape parameters, as described here.

In addition, you can specify perturbations of the shape as a linear combination of other shapes (for example, buckling mode shapes); see *Introducing a Geometric Imperfection into a Model*.

The definition of the nodal coordinates for a model in the Abaqus input file is then possible using a combination of four types of methods:

- You can directly define individual nodes and their respective coordinates; these coordinates are part of the definition of the basic shape, $\boldsymbol{x}_0$, and can be parametrized.
- Node generation can be used to create nodes and their coordinates according to geometrically simple mappings that rely on existing node definitions; these generated coordinates are also part of the definition of the basic shape, $\boldsymbol{x}_0$. If necessary, the node generation input can be parametrized.
- Parameter shape variations can be used to vary the coordinates of nodes defined using the above methods.
- Geometric imperfections can be used to perturb nodal coordinates previously defined using any combination of the above three types of methods.

Shape Parametrization Using Shape Variations

Instead of parametrizing nodal coordinates directly, you can specify shape variations. Each shape variation must be associated with a single shape parameter. The names of the parameters associated with the shape variations must be chosen such that the names remain unique when interpreted in a case-insensitive manner. The values of the shape parameters are assigned using parameter definitions.

A parameter shape variation can be defined more than once for the same parameter so that different parts of a shape variation can be specified separately. In these cases if the same node is specified in multiple parameter shape variation definitions, the last definition for the node prevails.

A node that is specified under a parameter shape variation definition that has not also been defined directly or through node generation will be ignored.

You can specify shape variations using a combination of three possibilities: directly specifying them, reading them from an alternate input file, and reading them from the results files of auxiliary analyses. These methods are described in the following sections.

Defining Shape Variations Directly or Reading Them from an Alternate Input File

You can define the shape variation data directly by specifying the node number and corresponding variations of coordinate components. Alternatively, the data can be given in an ASCII file.

Input File Usage: Use the following option to specify the shape variation data directly:

**PARAMETER SHAPE VARIATION*, PARAMETER=*name*

Use the following option to specify the shape variation data in an alternate input file:

**PARAMETER SHAPE VARIATION*, PARAMETER=*name*, INPUT=*input file*

Defining Shape Variations in Alternative Coordinate Systems

By default, the shape variation data are interpreted in the global rectangular Cartesian coordinate system. You can specify the shape variation data (either directly or in an alternate input file) in cylindrical or spherical coordinate systems. In such cases the computation of the shape variation is done as follows. The nodal coordinate components that define the basic shape are first transformed from the global rectangular Cartesian coordinate system in which they are stored to the specified coordinate system. The shape variation coordinate components are then added to give updated coordinate components, which are transformed back to the global rectangular Cartesian coordinate system. Finally, the shape variation is taken as the difference between the updated coordinate components and the original coordinate components, using the components expressed in the global rectangular Cartesian coordinate system. The value of the shape parameter associated with the shape variation is not used at any point in the calculation of the shape variation.

Input File Usage: Use the following option to specify the shape variation data in a rectangular coordinate system (the default):

**PARAMETER SHAPE VARIATION*, PARAMETER=*name*, SYSTEM=R

Use the following option to specify the shape variation data in a cylindrical coordinate system:

**PARAMETER SHAPE VARIATION*, PARAMETER=*name*, SYSTEM=C

Use the following option to specify the shape variation data in a spherical coordinate system:

**PARAMETER SHAPE VARIATION*, PARAMETER=*name*, SYSTEM=S

Using Auxiliary Analyses to Generate Shape Variations

Auxiliary models are additional finite element models that are used to generate shape variations for a primary model. Rather than defining shape variations directly on a node-by-node basis, auxiliary models can be used to simplify this process. Auxiliary analyses are finite element analyses of these auxiliary models.

An auxiliary model usually has the same geometry, element connectivity, and material type as the primary model. However, the boundary conditions are usually different. Applying loading to an auxiliary model results in sets of displacements that we may interpret as shape variations. For example, we may be interested in studying the sensitivity of the nonlinear buckling behavior of a structure with respect to imperfections in the structure. In this case we could perform an auxiliary eigenvalue linear buckling analysis and then use the resulting mode shapes as shape variations to be added to the basic geometry of the primary model. (This particular problem could also be addressed by using a geometric imperfection.)

Abaqus reads the shape variation data from auxiliary analyses through the user node labels. Abaqus does not check model compatibility between both analysis runs. Shape variation data cannot be read from the results file for models defined in terms of an assembly of part instances (*Assembly Definition*).

Reading Shape Variations from a Static Analysis Results File

To define a shape variation based on the deformed geometry of a previous static analysis, specify the results file and step from a previous static analysis. Optionally, you can specify the increment number from which displacement data are read. (By default, Abaqus will read data from the last increment available for the specified step on the results file.) In addition, you can read shape variation data for a specified node set.

Input File Usage: **PARAMETER SHAPE VARIATION*, PARAMETER=*name*,
FILE=*results file*, STEP=*step*, INC=*inc*, NSET=*name*

Reading Shape Variations from an Eigenvalue Analysis Results File

To define a shape variation based on a mode shape from a previous eigenvalue analysis, specify the results file and step from a previous eigenfrequency extraction or eigenvalue buckling prediction analysis. Optionally, you can specify the mode number from which eigenvector data are read. (By default, Abaqus will read data from the first eigenvector available for the specified step on the results file.) In addition, you can read eigenmode data for a specified node set.

Input File Usage: **PARAMETER SHAPE VARIATION*, PARAMETER=*name*,
FILE=*results file*, STEP=*step*, MODE=*mode*, NSET=*name*

Shape Parametrization and Design Sensitivity Analysis

For the purpose of design sensitivity analysis with Abaqus/Design (*Direct Design Sensitivity Analysis*) if the parameter specified for a parameter shape variation is also specified as a design parameter, the shape variation is used to define the design gradient of the nodal coordinates and nodal normals with respect to the design parameter. If you wish to perform design sensitivity analysis for the basic shape, all shape parameters must be given a value of zero. In addition, if *any* parameter specified in a parameter shape variation definition is also specified as a design parameter, the parameters of *all* parameter shape variations must be specified as design parameters.

In DSA calculations for shell and beam elements Abaqus always computes the design gradients of nodal normals using the design gradients of nodal coordinates. To overwrite the gradients computed by Abaqus, you must provide the nodal normal as part of the node definition and design gradients of the normals using a parameter shape variation. To prescribe a design-independent normal, you must provide a zero design gradient explicitly. For shape variations read from the results file, Abaqus computes the gradients of the normals based on the displacements and ignores the nodal rotations.

For beam elements Abaqus computes the design gradients for the $\mathbf{n}_1$-direction of the beam cross-section using the gradients of the node coordinates and the gradients for the $\mathbf{n}_2$-direction specified using a parameter shape variation. You cannot provide the shape variation for the $\mathbf{n}_1$-direction. Abaqus ignores any such design gradients implicitly provided in either the beam section definition or as an extra node in the beam element connectivity.

In cases where the data defining a shape variation are given in a cylindrical or spherical coordinate system it is important that you understand how the shape variation is calculated from the data. This calculation is described in the previous section.

Visualization of Shape Variations

Shape variations can be visualized only after the parametrized input file has been processed by the analysis input file processor. Therefore, at least a data check run must be executed before parameter shape variations can be visualized using Abaqus/CAE.

The shape variations associated with each individual shape parameter can be visualized as displaced shape plots at step zero of the analysis. The basic shape is interpreted as the undeformed shape, and the shape generated by adding the i^{th} shape variation to the basic shape is interpreted as the i^{th} displaced shape.

The combination of all shape variations added to the basic shape represents the true undeformed shape of the analysis.

Using Abaqus/CAE to Compute Shape Variations

A capability for computing shape variations is provided by the Abaqus Scripting Interface command `_computeShapeVariations( )`. Using the command requires some familiarity with the Abaqus Scripting Interface and the execution of scripts in Abaqus/CAE. The procedure that must be followed is described and illustrated in *Using Abaqus/CAE to compute shape variations*.

Nodal Thicknesses

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Membrane Elements*
- *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*
- *Using a General Shell Section to Define the Section Behavior*
- **NODAL THICKNESS*
- **MEMBRANE SECTION*
- **RIGID BODY*
- **SHELL GENERAL SECTION*
- **SHELL SECTION*

Overview

Nodal thicknesses are used to define continuously varying thicknesses for:

- shell structures;
- membrane structures; or
- in Abaqus/Explicit rigid elements.

Defining Nodal Thicknesses

You can specify the thickness of a shell, membrane, or rigid element at a particular node or node set.

Input File Usage: **NODAL THICKNESS*
node_number or node_set_name, *thickness*

Abaqus/CAE Usage: Use the following option for a conventional shell composite layup:

Property module: composite layup editor: **Shell Parameters**: **Nodal distribution**: select an analytical field or a node-based discrete field

Use the following option for a homogeneous shell section:

Property module: shell section editor: **Basic**: **Nodal distribution**: select an analytical field or a node-based discrete field

Use the following option for a composite shell section:

Property module: shell section editor: **Advanced**: **Nodal distribution**: select an analytical field or a node-based discrete field

Reading Nodal Thicknesses from an Alternate File

The nodal thickness data can be stored in a separate file and read from there at the start of the analysis. For details on the syntax of such file names, see *Input Syntax Rules*.

Input File Usage: **NODAL THICKNESS*, INPUT=*file_name*

Abaqus/CAE Usage: Reading nodal thicknesses from an alternate file is not supported in Abaqus/CAE.

Generating Continuously Varying Thicknesses between Two Nodes or Node Sets

Abaqus can linearly interpolate the thickness between two bounding nodes or node sets. The thicknesses at the bounding nodes must first be defined.

Input File Usage: Use the following options:

**NODAL THICKNESS*
first bounding node or node set, *thickness*
second bounding node or node set, *thickness*
**NODAL THICKNESS*, GENERATE
first bounding node or node set, second bounding node or node set,
number of intervals, increment in node numbers

Abaqus/CAE Usage: Generating thicknesses between bounding nodes or node sets is not supported in Abaqus/CAE.

Specifying a Continuously Varying Thickness for Shell, Membrane, and Rigid Elements

You must specify that a shell or membrane element is going to have a continuously varying thickness rather than a homogeneous thickness when you define the element section. See *Membrane Elements*, *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*, and *Using a General Shell Section to Define the Section Behavior* for details.

In Abaqus/Explicit you must specify that a rigid element is going to have a continuously varying thickness when you define the rigid body to which the element belongs; see *Rigid Elements*. In Abaqus/Standard rigid elements cannot have a continuously varying thickness.

Every node that is part of a shell, membrane, or rigid element using a continuously varying thickness must have a nodal thickness defined. Abaqus will issue an error message if there is a node with no nodal thickness in an element that is using a continuously varying thickness.

Specifying a Continuously Varying Thickness for a Composite Shell

When a composite shell structure has a continuously varying thickness, the total thickness of the shell at any node is defined by the nodal thickness value. The total thickness at an integration point is interpolated from the nodal thicknesses. The layer thicknesses given in the shell section definition are used as relative thicknesses and are scaled proportionally such that the sum of the layer thicknesses equals the total thickness at the integration point.

Example

For example, if a composite shell section were defined with the following input:

```
*SHELL SECTION, COMPOSITE, NODAL THICKNESS, ELSET=name
1.5, 3, STEEL
2.5, 3, FOAM
1.0, 3, STEEL
```

and the total thickness at a point was only 1.0, the thicknesses of the individual layers at the point would be 0.3 for the first steel layer, 0.5 for the foam layer, and 0.2 for the second steel layer.

Creating a Discontinuity in the Shell, Membrane, or Rigid Element Thicknesses

You can specify only a single thickness at each node. Therefore, use separate nodes along the interface on shell, membrane, or rigid elements where there is a discontinuity in the thickness and assign the appropriate thickness to each group of nodes. For elements that are not part of a rigid body, multi-point constraints must be used to make the displacements (and rotations, for shells) the same at corresponding nodes.

Normal Definitions at Nodes

Products: Abaqus/Standard Abaqus/Explicit

References:

- **NORMAL*
- **NODE*

Overview

Normals can be defined at nodes:

- with a user-specified normal definition;
- following the nodal coordinates as part of the node definition for beam and shell elements;
- on rigid main surfaces used in contact pairs in Abaqus/Standard;
- in beam and shell elements;
- for line spring elements to give the direction normal to the flaw in the structure;
- for gasket elements to give the thickness direction of the elements; and
- for contour integral evaluation.

The normals defined at nodes do not affect the element face normals, which are defined by the element connectivity. They need not be of unit length.

Contact Surfaces in Abaqus/Standard

User-specified surface normals for contact surfaces in Abaqus/Standard are relevant only when the small-sliding contact approach is used or when the finite-sliding contact approach is used with rigid elements that make up the main surface. User-specified surface normals defined on deformable main surfaces in contact pairs are ignored when finite sliding is used.

The small-sliding contact formulation uses the surface normals at each node along the main surface to define a normal vector that varies smoothly from point to point on the surface. For a detailed discussion on how the “main plane” is constructed for each secondary node using the surface normals, see *Contact Formulations in Abaqus/Standard*.

For main surfaces composed of rigid elements Abaqus/Standard smooths any discontinuous surface normal transitions between the rigid elements. The surface normals at the nodes are used to control the surface normal interpolation. For a detailed discussion on the smoothing of such main surfaces, see *Analytical Rigid Surface Definition*.

To define the normal, specify the components of the normal in the global coordinate system.

Input File Usage: **NORMAL*, TYPE=CONTACT SURFACE

Elements

User-specified normals might be necessary for beam and shell elements, line spring elements, gasket elements, or elements involved in contour integral evaluations. In such cases, specify the components of the normal in the global coordinate system.

Input File Usage: **NORMAL*, TYPE=ELEMENT

Beam and Shell Elements

User-specified normals might be needed to define the desired normal directions at shell surface intersections or at beam intersections where the automatically determined normals might be inappropriate for the model (see *Beam Element Cross-Section Orientation* or *Defining the Initial Geometry of Conventional Shell Elements*).

The nodal normals can also be defined as part of the node definition. While you can define a single normal for all elements connected to a node as part of the node definition, a user-specified normal definition defines a normal for a particular element at a node, thus allowing you to define separate normals for each element connected to a node. User-specified normal definitions supersede normals defined as part of a node definition.

Input File Usage: **NODE*
Specify the normals in the fifth, sixth, and seventh positions on the data line.

For example, the following lines define some normals as part of node definitions; the normal to be used at node 7 in element 2 is then redefined using a user-specified normal definition:

```
*NODE
6, 5., 5., , -0.5, .8
7, 10., 8., , -0.5, .8
9, 14., 4., , .6, .6
*NORMAL
2, 7, .6, .6
```

Line Spring Elements

For line spring elements user-specified normals can be used to give the direction normal to the flaw in the structure. See *Line Spring Elements for Modeling Part-through Cracks in Shells* for a description of these elements.

Gasket Elements

For gasket elements user-specified normals can be used to specify the thickness direction of the elements. The nodal thickness directions can also be defined as part of the gasket section definition. Thickness directions defined by user-specified normals supersede thickness directions defined as part of the gasket section definition. See *Defining the Gasket Element's Initial Geometry* for a description of the definition of the thickness direction for these elements.

Contour Integral Evaluation

For contour integral evaluations (*Contour Integral Evaluation*) surface normals should be specified at all surface nodes lying within the bounds of the requested contours. These nodes are printed out under the “Contour Integral” information in the data (`.dat`) file. For accurate contour integral evaluation, it is important that the virtual crack extension direction is in the plane of the surface for the following cases: when a crack front intersects the external surface of a three-dimensional solid, when the crack front intersects a surface of material discontinuity, or when

the crack is in a curved shell. If no normals are specified, Abaqus calculates the normals automatically, except for a model with tetrahedral elements. In the latter case, you must either specify the normals or request that Abaqus calculate the normals to improve the accuracy of contour integral evaluation at the free surface.

The nodal normal data specified as part of a node definition is not activated for solid elements unless a user-specified normal definition is used in the model; it suffices to include a user-specified normal definition for only one node to activate the utilization of the nodal normal data specified as part of a node definition.

The Coordinate System in Which Normals Are Defined

Abaqus models can be defined in terms of an assembly of part instances (see *Assembly Definition*). Normals at nodes defined within a part (or part instance) are defined relative to the part coordinate system. These normals are rotated according to the positioning data given for each instance of the part. Normals can be defined at reference nodes at the assembly level if necessary. Normals defined at the assembly level are defined in the global coordinate system.

For models that are not defined in terms of an assembly of part instances, normals are defined in the global coordinate system.

Transformed Coordinate Systems

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Prescribed Conditions*
- *Output to the Output Database*
- **TRANSFORM*
- *Transforming results into a new coordinate system*
- *Methods for creating a datum coordinate system*

Overview

A nodal transformation is used to define a local coordinate system for:

- the definition of concentrated forces and moments;
- the definition of displacement and rotation boundary conditions;
- the definition of linear constraint equations; and
- the output of vector-valued quantities.

A nodal transformation cannot be used to specify a local coordinate system for defining:

- nodal coordinates—see *Specifying a Local Coordinate System in Which to Define Nodes* or *Specifying a Local Coordinate System for the Nodal Coordinates* instead; or
- material properties or rebars—see *Orientations* instead.

Defining a Local Coordinate System

Normally displacement and rotation components are associated with the global, rectangular Cartesian axis system. When a transformed coordinate system is associated with a node, all input data for concentrated forces and moments and for displacement and rotation boundary conditions at the node are given in the local system. The following transformations are available:

- Rectangular Cartesian
- Cylindrical
- Spherical

The coordinate transformation defined at a node must be consistent with the degrees of freedom that exist at the node. For example, a transformed coordinate system should not be defined at a node that is connected only to a SPRING1 or SPRING2 element, since these elements have only one active degree of freedom per node.

Input File Usage: You must identify the node set for which the local transformed system is defined.

**TRANSFORM*, NSET=*name*

Abaqus/CAE Usage: In Abaqus/CAE you define a local coordinate system independent of its use and then refer to it when you apply a load or boundary condition at a node.

Any module: **Tools->Datum**: **Type**: **CSYS**

Interaction module: load or boundary condition editor: **CSYS: Edit**: select local coordinate system

Defining a Local Coordinate System in a Model That Contains an Assembly of Part Instances

In a model defined in terms of an assembly of part instances, you can define a nodal transformation at the part, part instance, or assembly level. A nodal transformation defined at the part or part instance level will be rotated according to the positioning data given for each instance of that part (or for the part instance). See *Assembly Definition*. Multiple transformation definitions are not allowed at a node, even if one of them is at the part level and another is at the assembly level.

Large-Displacement Analysis

The transformed coordinate system is always a set of fixed Cartesian axes at a node (even for cylindrical or spherical transforms). These transformed directions are fixed in space; the directions do not rotate as the node moves. Therefore, even in large-displacement analysis, the displacement components must always be given with respect to these fixed directions in space.

Defining a Rectangular Cartesian Coordinate Transformation

In a rectangular Cartesian transformation the transformed directions are parallel at all nodes of the set. The coordinates of two points must be given, as shown in *Figure 1*.

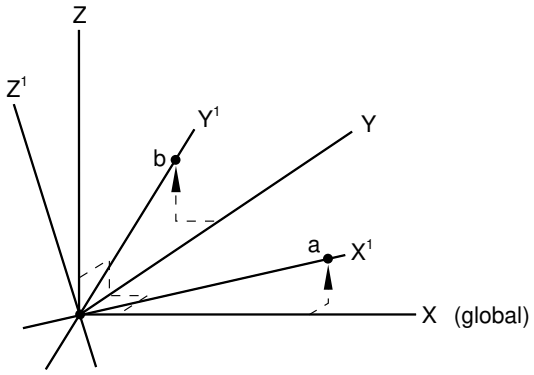


Figure 1: Cartesian transformation.

The first point, *a*, must be on a line through the global origin; this point defines the transformed X^1-direction. The second point, *b*, must be in the plane containing the global origin and the transformed X^1- and Y^1-directions. This second point should be on or near the positive Y^1-axis.

Input File Usage: **TRANSFORM*, NSET=*name*, TYPE=R (default)

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Rectangular**

Defining a Cylindrical Coordinate Transformation

The radial, tangential, and axial directions must be defined based on the original coordinates of each node in the node set for which the transformation is invoked. The global ($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$) coordinates of the two points defining the axis of the cylindrical system (points *a* and *b* as shown in *Figure 2*) must be given.

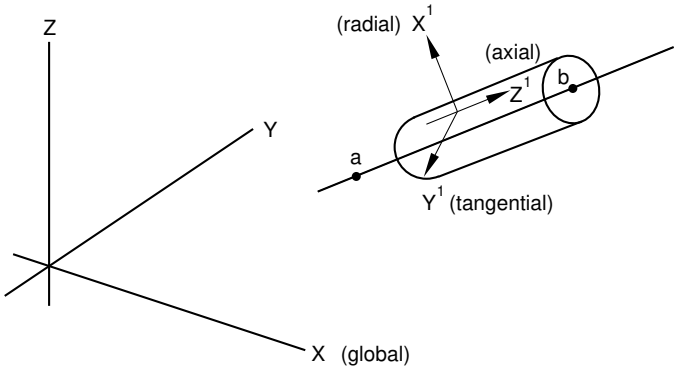


Figure 2: Cylindrical transformation.

The origin of the local coordinate system is at the node of interest. The local $\boldsymbol{X}^{\mathbf{1}}$-axis is defined by a line through the node, perpendicular to the line through points *a* and *b*. The local $\boldsymbol{Z}^{\mathbf{1}}$-axis is defined by a line that is parallel to the line through points *a* and *b*. The local $\boldsymbol{Y}^{\mathbf{1}}$-axis forms a right-handed coordinate system with $\boldsymbol{X}^{\mathbf{1}}$ and $\boldsymbol{Z}^{\mathbf{1}}$. A cylindrical coordinate system cannot be defined for a node that lies along the line joining points *a* and *b*.

Input File Usage: **TRANSFORM*, NSET=*name*, TYPE=C

Abaqus/CAE Usage: Any module: **Tools**->**Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Cylindrical**

Defining a Spherical Coordinate Transformation

The radial, circumferential, and meridional directions must be defined based on the original coordinates of each node in the node set for which the transformation is invoked. The global ($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$) coordinates of the center of the spherical system, *a*, and of a point on the polar axis, *b*, must be given as shown in *Figure 3*.

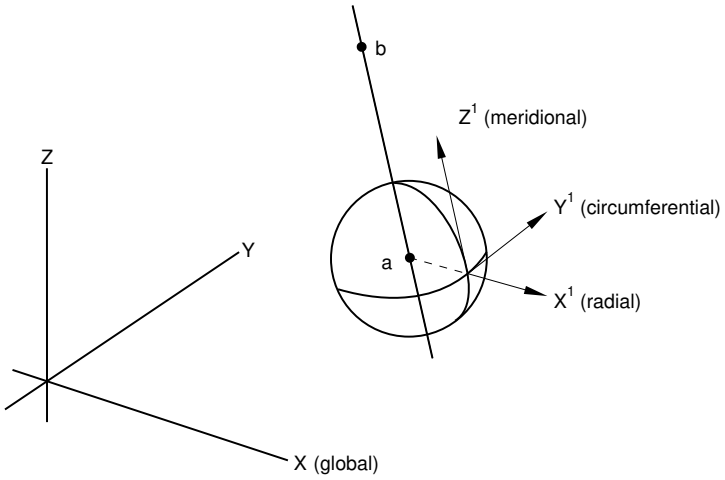


Figure 3: Spherical transformation.

The origin of the local coordinate system is at the node of interest. The local X^1-axis is defined by a line through the node and point *a*. The local Z^1-axis lies in a plane containing the polar axis (the line between points *a* and *b*) and is perpendicular to the local X^1-axis. The local Y^1-axis forms a right-handed coordinate system with X^1 and Z^1.

A spherical coordinate system cannot be defined for a node that lies along the line joining points *a* and *b*.

Input File Usage: **TRANSFORM*, NSET=*name*, TYPE=S

Abaqus/CAE Usage: Any module: **Tools**->**Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Spherical**

Output at a Node Associated with a Coordinate Transformation

Printed and file output of vector-valued quantities from Abaqus/Standard at transformed nodes can be in the local or global system (see *Specifying the Directions for Nodal Output*). By default, the values are written to the data file in the local system, whereas the values are written to the results file in the global system (since this is more convenient for postprocessing). Consequently, reaction forces printed using the default will not appear to equilibrate loads applied in the global system. However, these reaction forces and loads should equilibrate if you output them to the data file in the global system.

File output from Abaqus/Explicit is always in the global system.

Output database output of field vector-valued quantities at transformed nodes is in the global system. The local transformations are also written to the output database. You can apply these transformations to the results in the Visualization module of Abaqus/CAE to view the vector components in the transformed systems.

Output database output of history vector-valued quantities at transformed nodes can be in the local or global system (see *Output to the Output Database*). By default, the values are written in the global system (since this is more convenient for postprocessing).

Adjusting Nodal Coordinates

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- **ADJUST*
- *Defining adjust points constraints*

Overview

Nodal adjustment is used for:

- adjusting user-specified nodal coordinates so that the nodes lie on a given surface; and
- specifying the direction along which the nodes are moved.

Adjusting Nodal Coordinates

In general, user-specified nodal coordinates are not modified during input file processing. However, there are some situations where mesh coordinates are known only in a generic way and it is inconvenient to determine their coordinates for their actual usage. For example, when using fasteners the specified reference node should be positioned at its projection point on the associated surface. Since that location may be known only approximately, you can use nodal adjustment to move the reference node to that location automatically. For typical usage of the nodal adjustment feature, refer to *About assembled fasteners*.

When using this feature, the nodes are adjusted to lie on the specified surface without regard for shell thickness or shell offsets. Therefore, it is not advisable to use this feature as a way of correcting initial overclosures for contact or for tie constraints. In addition, care should be taken when choosing the nodes to be adjusted because the feature does not respect any constraints relating the relative position of the adjusted node with other nodes (e.g., rigid body definitions).

Input File Usage: Use the following option to identify the nodes to be moved and the surface onto which the nodes are to be moved:

**ADJUST*, NODE SET=*name*, SURFACE=*name*

Abaqus/CAE Usage: Use the following option to move the control point of a coupling constraint onto the coupling surface:

Interaction module: **Constraint->Create**: **Coupling**; **Adjust control point to lie on surface**

Use the following option to move any point or points onto any surface:

Interaction module: **Constraint->Create**: **Adjust points**

Specifying the Nodal Adjustment Direction

A node can be moved to the surface using a normal adjustment or a directed adjustment. By default, the node is adjusted to the closest point on the specified surface along the normal to the surface. You can specify an orientation

to move the node to the surface along a given direction rather than along the normal to the surface. The vector along the local *Z*-direction from the orientation definition is used to move the node to the surface (see *Orientations*). If no projection can be found, the nodal coordinates are left unmodified.

Input File Usage: **ADJUST*, ORIENTATION=*name*

Abaqus/CAE Usage: The orientation projection option is not supported in Abaqus/CAE.

Element Definition

In this section:

- *Element Definition*
- *Element Foundations*
- *Defining Reinforcement*
- *Defining Rebar as an Element Property*
- *Orientations*

Element Definition

Products: Abaqus/Standard Abaqus/Explicit

References:

- **ELCOPY*
- **ELEMENT*
- **ELGEN*
- **ELSET*

Overview

This section describes the methods for defining elements in an Abaqus input file. In a preprocessor such as Abaqus/CAE, you define the model geometry rather than the nodes and elements; when you mesh the geometry, the preprocessor automatically creates the nodes and elements needed for analysis. Although the concepts discussed in this section apply in general to the element definitions in the input file that is created by Abaqus/CAE, the methods and techniques described here apply only if you are creating the input file manually.

Element definition consists of:

- assigning an element number to the element;
- defining individual elements by specifying their nodes;
- grouping elements into element sets; and
- creating elements from existing elements by generating them incrementally or by copying existing elements.

If any element is specified more than once, the last specification given is used.

Assigning an Element Number to the Element

Each individual element must have a numeric label called the element number, which is assigned when the element is defined. The element number must be a positive integer, and the maximum element number allowed is 999999999 (for information on integer input, see *Input Syntax Rules*). The elements do not need to be numbered continuously.

An Abaqus model can be defined in terms of an assembly of part instances (see *Assembly Definition*). In such a model almost all elements must belong to a part or part instance. The only exceptions are mass, rotary inertia, capacitance, connector, spring, and dashpot elements, which can belong to a part or to the assembly. Element numbers must be unique within a part, part instance, or the assembly; but they can be repeated in different parts or part instances.

Defining Individual Elements by Specifying Their Nodes

You can define individual elements by specifying the element number and the nodes that define the element. In addition, you must specify the element type. The element must be chosen from one of the element types specified in *About the Element Library*; or, in Abaqus/Standard, it can be a user-defined element (*User-Defined Elements*) or a substructure (*Using Substructures*).

Input File Usage: **ELEMENT*, TYPE=*name*

For example, the following lines create element number 11, which is of type C3D8R, by defining its nodes (2, 3, 9, 7, 5, 8, 12, 16):

*ELEMENT, TYPE=C3D8R
11, 2, 3, 9, 7, 5, 8, 12, 16

Using Large Node Numbers with Elements That Use Many Nodes

The following rules apply when defining elements:

- The connectivity for each element is considered a logical record, and any number of input lines can be used to specify it. Abaqus will read the first line for an element and consider the next line a continuation line if a comma ends the line and the element definition is not complete.
- Any number of continuation lines can be used.
- For elements such as C3D27 with a variable number of nodes (see *Solid (Continuum) Elements*), the last line should not end with a comma or Abaqus will interpret the next element definition as a continuation of the current element.

For example,

```
*ELEMENT, TYPE=C3D20
100001, 100001, 100002, 100003, 100004, 100005, 100006, 100007,
100008, 100009, 100010, 100011, 100012, 100013, 100014, 100015,
100016, 100017, 100018, 100019, 100020
```

Reading Element Definitions from a File

Element definitions can be read into Abaqus from an alternate file. The syntax of such file names is described in *Input Syntax Rules*.

Input File Usage: **ELEMENT*, INPUT=*file_name*

Reading Substructure Definitions from a Substructure Database

Substructure definitions can be read from the substructure database in which the substructure resides (*Using Substructures*).

Input File Usage: **ELEMENT*, FILE=*substructure_database_name*

Defining Axisymmetric Elements with Asymmetric Deformation

You can define a positive offset number that will be used to specify nodes for axisymmetric elements with asymmetric deformation (see *Choosing the Element's Dimensionality*, *Axisymmetric Solid Elements with Nonlinear, Asymmetric Deformation*, and *Axisymmetric Shell Elements with Nonlinear, Asymmetric Deformation* for more information on axisymmetric elements with asymmetric deformation; they are available only in Abaqus/Standard). The default offset is 100000.

Input File Usage: **ELEMENT*, OFFSET=*number*

Defining Gasket Elements

There are several methods for defining gasket elements. (See *About Gasket Elements*, *Including Gasket Elements in a Model*, and *Defining the Gasket Element's Initial Geometry* for more information on gasket elements; they are available only in Abaqus/Standard.)

In the first method you define individual elements by specifying the element number and the nodes that define the element.

In the second method you specify only the nodes on the bottom surface of the gasket element and a positive offset number that will be used to define the corresponding nodes for the top surface. For the 18-node gasket element you give the first eight nodes followed by the midsurface node; i.e., node 17 in the full element nodal connectivity.

Abaqus/Standard can generate the midface nodes of the 18-node gasket elements automatically if both element faces are part of contact surfaces. To invoke this feature, you enter a blank instead of the actual node numbers in either of the above input methods. Abaqus/Standard will then generate the node numbers and coordinates of the midface nodes automatically.

Input File Usage: Use the following option to specify the element number and the nodes that define the element:

**ELEMENT*, TYPE=*name*

Use the following option to specify the nodes on the bottom surface of the element and a positive offset number for the top surface:

**ELEMENT*, TYPE=*name*, OFFSET=*offset number*

Using Solid Element Connectivity to Define Gasket Elements

The node numbering scheme for gasket elements does not correspond to the node numbering scheme for continuum elements, which can be inconvenient if the mesh generator used does not support gasket elements directly or in thermal-stress analysis where continuum elements are used to model the heat conduction in the gasket. For such cases you can specify that solid element connectivity is used to define the gasket element. By default, it is assumed that the first (S1) face of the solid element coincides with the first (SNEG) face of the gasket element. If the equivalent solid element is oriented differently, specify the face number on the solid element that corresponds to the first face of the gasket element. The solid element must have the same number of nodes on each face as the corresponding gasket element; any nodes between the faces will be ignored. The 18-node gasket element is an exception. If both element faces are part of contact surfaces, the connectivity of a 20-node brick element can be used, and Abaqus/Standard will generate the node numbers and coordinates of the midface nodes automatically.

Abaqus/Standard will transform the solid element connectivity to the normal gasket element connectivity immediately upon reading the data. Hence, all output to the data (`.dat`), results (`.fil`), and output database (`.odb`) files will use the normal gasket element connectivity.

Input File Usage: Use the following option to specify solid element connectivity for a gasket element in which the first face of the solid element corresponds to the first face of the gasket element:

**ELEMENT*, TYPE=*name*, SOLID ELEMENT NUMBERING

Use the following option to specify solid element connectivity for a gasket element and the face of the solid element that corresponds to the first face of the gasket element:

**ELEMENT*, TYPE=*name*, SOLID ELEMENT NUMBERING=*face number*

Examples

The following lines create GK3D12M element number 11 that has node numbers 1, 2, 3, 4, 5, 6, 1001, 1002, 1003, 1004, 1005, and 1006:

```
*ELEMENT, TYPE=GK3D12M
11, 1, 2, 3, 4, 5, 6, 1001, 1002, 1003, 1004, 1005, 1006
```

The same element connectivity is also created by the following lines:

```
*ELEMENT, TYPE=GK3D12M, OFFSET=1000
11, 1, 2, 3, 4, 5, 6
```

The equivalent solid element would be C3D15, with the following input:

```
*ELEMENT, TYPE=GK3D12M, SOLID ELEMENT NUMBERING
11, 1, 2, 3, 1001, 1002, 1003, 4, 5, 6, 1004, 1005, 1006,
501, 502, 503
```

where nodes 501, 502, and 503 would not be used.

Defining Cohesive Elements

There are three methods for defining cohesive elements. (See *About Cohesive Elements*, *Modeling with Cohesive Elements*, and *Defining the Cohesive Element's Initial Geometry* for more information on cohesive elements.)

- In the first method you specify the element number and all of the nodes that define the element.
- In the second method you specify only the nodes on the bottom face of the cohesive element and Abaqus will create the remaining nodes, numbering them according to an offset number that you specify.
- In the third method, which is applicable only to pore pressure cohesive elements, you specify the nodes on the bottom and top faces. Abaqus will create the remaining middle-face nodes according to an offset number that you specify.

Defining a Cohesive Element by Specifying All Nodes

With this method you specify all nodes that define the cohesive element. See *Two-Dimensional Cohesive Element Library*, *Three-Dimensional Cohesive Element Library*, and *Axisymmetric Cohesive Element Library* for the element node numbering definition.

Input File Usage: Use the following option to specify the element number and the nodes that define the element:

**ELEMENT*, TYPE=*name*

For example, the following lines create COH3D8 element number 11 that has node numbers 1, 2, 3, 4, 1001, 1002, 1003, and 1004:

**ELEMENT*, TYPE=COH3D8
11, 1, 2, 3, 4, 1001, 1002, 1003, 1004

Defining a Cohesive Element by Specifying Only the Bottom Face Nodes

With this method you specify only the nodes on the bottom face of the cohesive element and a positive offset number. With displacement cohesive elements, the offset number is added to the bottom face node numbers to create the corresponding nodes on the top face. With pore pressure cohesive elements, the offset number first is added to the bottom face node numbers to create the corresponding nodes on the top face, then the offset number is added to the top face node numbers to create the corresponding nodes on the middle face.

Input File Usage: Use the following option to specify the nodes on the bottom face of the element and a positive offset number for nodes on the remaining face or faces:

**ELEMENT*, TYPE=*name*, OFFSET=*offset number*

For example, the following lines create COH3D8 element number 11 that has node numbers 1, 2, 3, 4, 1001, 1002, 1003, and 1004:

**ELEMENT*, TYPE=COH3D8, OFFSET=1000
11, 1, 2, 3, 4

and the following lines create pore pressure cohesive element COH3D8P element number 11 that has node numbers 1, 2, 3, 4, 1001, 1002, 1003, 1004, 2001, 2002, 2003, and 2004 (nodes 1, 2, 3, and 4 define the bottom face; nodes 1001, 1002, 1003, and 1004 define the top face; and nodes 2001, 2002, 2003, and 2004 define the middle face):

**ELEMENT*, TYPE=COH3D8P, OFFSET=1000
11, 1, 2, 3, 4

Defining a Pore Pressure Cohesive Element by Specifying Only the Bottom and Top Face Nodes

With this method you specify only the nodes on the bottom and top faces of the pore pressure cohesive element and a positive offset number. The offset number is added to the bottom face node numbers to create the corresponding nodes on the middle face.

Input File Usage: Use the following option to specify the nodes on the bottom and top faces of the pore pressure cohesive element and a positive offset number for the remaining middle-face nodes:

**ELEMENT*, TYPE=*name*, OFFSET=*offset number*

For example, the following lines create a pore pressure cohesive element COH3D8P element number 11 that has node numbers 1, 2, 3, 4, 1001, 1002, 1003, 1004, 2001, 2002, 2003, and 2004 (nodes 1, 2, 3, and 4 define the bottom face; nodes 1001, 1002, 1003, and 1004 define the top face; and nodes 2001, 2002, 2003, and 2004 define the middle face):

**ELEMENT*, TYPE=COH3D8P, OFFSET=2000
11, 1, 2, 3, 4, 1001, 1002, 1003, 1004

Grouping Elements into Element Sets

Element sets are used as convenient cross-references for defining loads, properties, etc. Element sets are the fundamental references of the model and should be used to assist the input definition. The members of an element set can be individual elements or other element sets. An individual element can belong to several element sets.

Elements can be grouped into element sets when they are created or after they have already been defined. In either case each element set is assigned a name. Element set names can be up to 80 characters long.

The same name can be used for a node set and for an element set.

All elements within an element set will be arranged in ascending order of their element number, and duplicates will be removed.

Once elements are assigned to an element set, additional elements can be added to the same element set; however, elements cannot be removed from an element set.

Assigning Elements to an Element Set as They Are Created

There are several ways that elements can be assigned to element sets as they are created.

Input File Usage: Use any one of the following options:

**ELEMENT*, ELSET=*name*
**ELGEN*, ELSET=*name*
**ELCOPY*, NEW SET=*name*

Assigning Previously Defined Elements to an Element Set

You can assign elements that you have defined previously (by specifying their nodes, by generating them incrementally, or by copying existing elements) to an element set by listing the elements forming the set directly or by generating the element set.

Listing the Elements That Form the Set Directly

You can list the elements that form the element set directly. Previously defined element sets, as well as individual elements, can be assigned to element sets.

Input File Usage: **ELSET*, ELSET=*name*

For example, the following lines add elements 3, 13, and 20 to set `LEFT`:

```
*ELSET, ELSET=LEFT
20
3, 13
```

The following lines add elements 5 and 16 to the existing set `LEFT`:

```
*ELSET, ELSET=LEFT
5, 16
** The above data line is equivalent to
specifying 5, 16, LEFT
```

The following lines add elements 22, 14, and all elements in set `LEFT` to set `B`:

```
*ELSET, ELSET=B
22, 14, LEFT
```

Thus, element set `B` contains the following elements: 3, 5, 13, 14, 16, 20, and 22. Element set `LEFT` can be assigned to element set `B` since the definition of `LEFT` occurs before the definition of `B`.

Generating the Element Set

To generate an element set, you must specify a first element, e_1; a last element, e_2; and the increment in element numbers between these elements, i. All elements going from e_1 to e_2 in steps of i will be added to the set.

Therefore, i must be an integer such that $(e_2 - e_1)/i$ is a whole number (not a fraction). The default is $i = 1$.

Input File Usage: **ELSET*, ELSET=*name*, GENERATE

For example, the following lines add elements 1, 3, 5, …, 19, 21 and elements 39, 49, 59, …, 129, 139 to set `UP`:

**ELSET*, ELSET=UP, GENERATE
1, 21, 2
39, 139, 10

Limitation on Updating Element Sets That Are Used to Define Other Element Sets

If an element set is constructed from previously defined element sets, subsequent updates to these sets are not taken into account.

Input File Usage: **ELSET*, ELSET=*name*

For example, the following lines add elements 1 and 2, but not 3, to the set `SET-AB` while adding elements 1 and 3 to set `SET-A`:

**ELSET*, ELSET=SET-A
1,
**ELSET*, ELSET=SET-B
2,
**ELSET*, ELSET=SET-AB
SET-A, SET-B
**ELSET*, ELSET=SET-A
3,

Defining Part and Assembly Sets

In a model defined in terms of an assembly of part instances, all element sets must be defined within a part, part instance, or the assembly definition. If an element set is defined within a part (or part instance), you can refer to the element numbers directly. To define an assembly-level element set, you must identify the elements to be added to the set by prefixing each element number with the part instance name and a "." (as explained in *Assembly Definition*). An assembly-level element set can have the same name as a part-level element set.

Example

The following input defines an element set, `set1`, that belongs to part `PartA` and will be inherited by every instance of `PartA`:

```
*PART, NAME=PartA
   ...
   *ELSET, ELSET=set1
    1,3,26,500
*END PART
```

An element set with the same name is defined at the assembly level as follows:

```
*ASSEMBLY, NAME=Assembly-1
   *INSTANCE, NAME=PartA-1, PART=PartA
    ...
   *END INSTANCE
   *INSTANCE, NAME=PartA-2, PART=PartA
    ...
   *END INSTANCE
   *ELSET, ELSET=set1
```

```
    PartA-1.1, PartA-1.3, PartA-1.26, PartA-1.500
    PartA-2.1, PartA-2.3, PartA-2.26, PartA-2.500
*END ASSEMBLY
```

Assembly-level element set `set1` contains all the elements from element sets `set1` belonging to part instances `PartA-1` and `PartA-2`. Therefore, the elements are assigned to two separate element sets: one at the part instance level and one at the assembly level. An assembly-level element set called `set1` could be created with entirely different elements than those that belong to the part set; part- and assembly-level element sets are independent. However, since in this example the same elements are assigned to both the part- and assembly-level element sets `set1`, the assembly-level set could alternatively be defined by

```
*ASSEMBLY, NAME=Assembly-1
   *INSTANCE, NAME=PartA-1, PART=PartA
    ...
   *END INSTANCE
   *INSTANCE, NAME=PartA-2, PART=PartA
    ...
   *END INSTANCE
   *ELSET, ELSET=set1
    PartA-1.set1, PartA-2.set1
*END ASSEMBLY
```

This element set definition is equivalent to the previous example, where the elements are listed individually.

Alternate Method for Defining Assembly-Level Element Sets

Sometimes it is not convenient to define an assembly-level element set by referring to part-level element sets. In such cases a set definition containing many elements can get quite lengthy. Therefore, an alternate method is provided.

Input File Usage: **ELSET*, ELSET=*ElsetName*, INSTANCE=*InstanceName*

The following example shows two equivalent ways to define an assembly-level element set; once by prefixing each element number with a part instance name (as shown above) and once using the more compact INSTANCE notation:

```
*ASSEMBLY, NAME=Assembly-1
  *INSTANCE, NAME=PartA-1, PART=PartA
   ...
  *END INSTANCE
  *INSTANCE, NAME=PartA-2, PART=PartA
   ...
  *END INSTANCE
  *ELSET, ELSET=set2
   PartA-1.11, PartA-1.12, PartA-1.13, PartA-1.14,
   PartA-2.21, PartA-2.22, PartA-2.23, PartA-2.24
  *ELSET, ELSET=set3, INSTANCE=PartA-1
   11, 12, 13, 14
  *ELSET, ELSET=set3, INSTANCE=PartA-2
   21, 22, 23, 24
*END ASSEMBLY
```

When the **ELSET* option is used more than once with the same name, as it is with `set3`, the elements in the second use of **ELSET* are appended to the set created by the first use of **ELSET*.

Internal Element Sets Created by Abaqus/CAE

In Abaqus/CAE many modeling operations are performed by picking geometry with the mouse. For example, a surface can be created by picking a face on a geometric part instance. Since the **SURFACE* option refers to an element set, this "picked" geometry must be translated into an element set in the input file. Such sets are assigned a name by Abaqus/CAE and marked as internal. You can view these internal sets using display groups in the Visualization module of Abaqus/CAE (see *Using display groups to display subsets of your model*).

Input File Usage: **ELSET*, ELSET=*ElsetName*, INTERNAL

Transferring of Element Sets

If the results of an Abaqus/Explicit analysis are imported into an Abaqus/Standard analysis (or vice versa) or results from an Abaqus/Standard analysis are imported into another Abaqus/Standard analysis (see *About Transferring Results between Abaqus Analyses*), all element set definitions in the original analysis are imported by default. Alternatively, you can import only selected element set definitions; see *Importing Element Set, Node Set, and Surface Definitions One Time* for details.

If a three-dimensional model is generated from a symmetric model (see *Symmetric Model Generation*), all element sets in the original model will be used (and expanded) in the generated model.

Creating Elements from Existing Elements by Generating Them Incrementally

You can generate elements incrementally from existing elements. The newly created elements are always the same element type as that of the main element.

Abaqus first generates a row of elements by copying the node pattern of a given element with prescribed increments in the node and element numbers. This row can then be repeated to form a layer, which can also be repeated to form a block.

To generate a row of elements, you must specify the following information:

- The main element number. The main element must exist at the time that the generation is specified, although it can be an element that has just been defined in this same element generation.
- The number of elements to be defined in the first row generated, including the main element.
- The increment in node numbers of corresponding nodes from element to element in the row. The default is 1. All element node numbers (except special-purpose nodes, discussed later) will increase by the same value.
- The increment in element numbers in the row. The default is 1.

To copy this newly created main row to create a layer of elements, you must specify the following additional information:

- The number of rows to be defined, including the main row.
- The increment in node numbers of corresponding nodes from row to row.
- The increment in element numbers of corresponding elements from row to row.

To copy this newly created main layer to create a block of elements, you must specify the following additional information:

- The number of layers to be defined, including the main layer.
- The increment in node numbers of corresponding nodes from layer to layer.
- The increment in element numbers of corresponding elements from layer to layer.

Input File Usage: **ELGEN*

For example, the elements forming the quarter cylinder shown in *Figure 1* can be generated by the following lines:

```
*ELGEN
1, 3, 1, 1, 5, 10, 10, 6, 100, 100
```

Incrementing Special-Purpose Nodes

By default, the following nodes are not incremented:

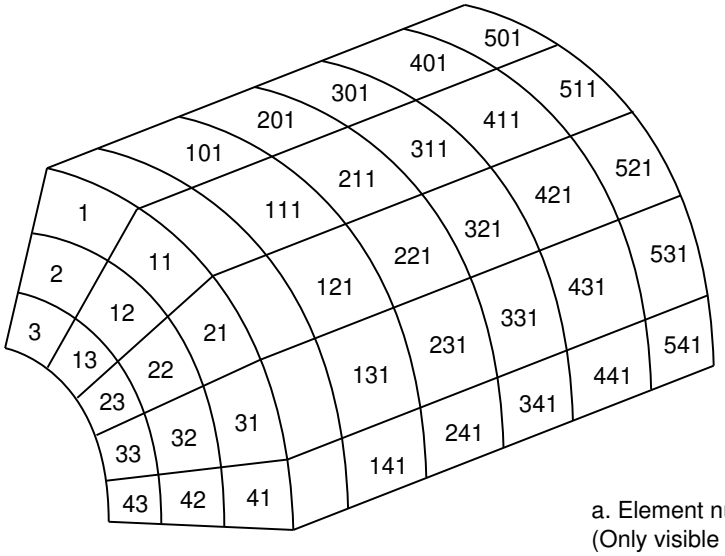
- rigid body reference nodes for IRS-type and drag chain elements; and
- nodes used to define the direction of the first cross-section axis for beams or frames in space.

You can specify that all nodes should be incremented. You define the increment between node numbers as described above. Usually the incrementation of all nodes is needed only for nodes used to define the direction of the first cross-section axis for beams in space.

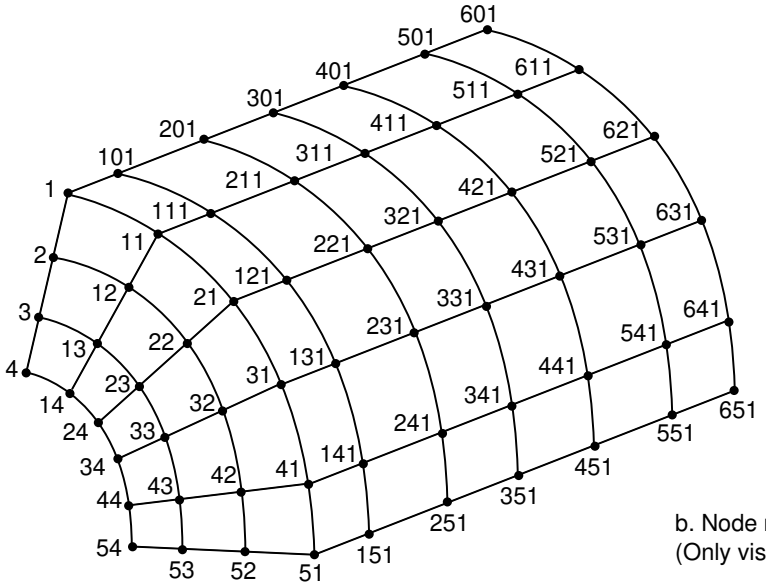
Input File Usage: **ELGEN*, ALL NODES

Creating Elements by Copying Existing Elements

You can create new elements by copying existing elements. You must identify the existing element set to copy and specify an integer constant that will be added to the node numbers of the existing elements to define the node numbers of the new elements. Likewise, you must specify an integer constant that will be added to the element numbers of existing elements to define element numbers for the elements being created.



a. Element numbers
(Only visible elements shown).



b. Node numbers
(Only visible nodes shown).

Figure 1: Element generation example.

You can assign the newly created elements to an element set. If you do not specify an element set name for the newly created elements, they are not assigned to an element set.

Input File Usage: **ELCOPY*, OLD SET=*name*, NEW SET=*new_name*,
SHIFT NODES=*number*, ELEMENT SHIFT=*number*

For example, the following data lines will generate new elements in set `B` that are copies of all elements in set `A` at the time this option is processed, with 1000 added to each element number and to each node number in the definitions of the new elements. The members of set `A` at the time the line is processed are those elements defined to be in set `A` by all element generation and element set definition lines that appear in the input file prior to this **ELCOPY* option.

*ELCOPY, OLD SET=A, NEW SET=B, ELEMENT SHIFT=1000,
SHIFT NODES=1000

Special Considerations for Continuum Elements

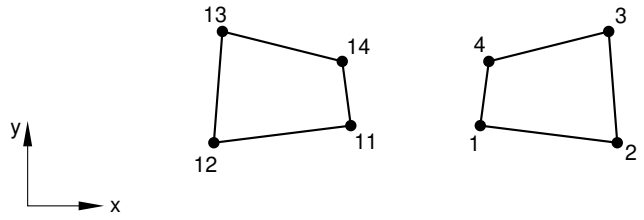
When copying existing elements, you can choose to modify the node numbering sequence for the elements being created to avoid creating continuum elements that violate the Abaqus convention for counterclockwise element numbering. This modification is normally required when the nodes have been generated by copying existing nodes (*Creating Nodes by Copying Existing Nodes*).

Input File Usage: *ELCOPY, REFLECT

For example, assume element 1 is in element set A and is defined by nodes 1, 2, 3, 4. The following data line will generate element number 11, also in set A, with nodes 11, 14, 13, and 12:

*ELCOPY, OLD SET=A, NEW SET=A, ELEMENT SHIFT=10, SHIFT NODES=10,
REFLECT

If the REFLECT parameter is not used, the new element will be defined by the node sequence 11, 12, 13, 14 and will violate the counterclockwise element numbering convention used with continuum elements, as shown below.



Element Foundations

Products: Abaqus/Standard Abaqus/CAE

References:

- **FOUNDATION*
- *Defining foundations*

Overview

Elastic element foundations:

- can be defined for stress/displacement elements in Abaqus/Standard according to the load identifiers described in *About the Element Library*;
- act like springs to ground; and
- are a simple way of including the stiffness effects of a support (such as the soil under a building) without modeling the details of the support.

Defining Element Foundation Behavior

Foundation pressures act normal to the element faces on which they are applied. In large-displacement analysis the direction of action of the foundation is based on the deformed configuration; foundations rotate with the element sides.

Convergence difficulties might arise with large-deformation problems since no corresponding foundation load stiffness terms are included in the element stiffness matrices.

To define the foundation behavior, you specify the foundation stiffness per unit area.

Input File Usage: Use the following option in the model definition portion of the input file:

**FOUNDATION*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Step: Initial**, **Elastic foundation**

Defining elastic foundation behavior for beams is not supported in Abaqus/CAE.

Defining Reinforcement

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- **BEAM SECTION*
- **EMBEDDED ELEMENT*
- **MEMBRANE SECTION*
- **PRESTRESS HOLD*
- **REBAR*
- **REBAR LAYER*
- **REBAR LINE*
- **SHELL SECTION*
- **SURFACE SECTION*

Overview

Rebar:

- are used to define layers of uniaxial reinforcement in membrane, shell, and surface elements (such layers are treated as a smeared layer with a constant thickness equal to the area of each reinforcing bar divided by the reinforcing bar spacing);
- can be used to add lines of discrete axial reinforcement in beam elements;
- can be used to add layers of reinforcement in a solid by embedding reinforced surface or membrane elements in the "host" solid elements as described in *Embedded Elements*;
- can be used to add additional stiffness, volume, and mass to the model;
- can be used in coupled temperature-displacement analysis but do not contribute to the thermal conductivity and specific heat;
- can be used in coupled thermal-electrical-structural analysis but do not contribute to the electrical conductivity, thermal conductivity and specific heat;
- cannot be used in heat transfer or mass diffusion analysis; and
- have material properties that are distinct from those of the underlying or host element.
- do not include the mass or volume of the underlying elements.

Defining Rebar Layers in Membrane, Shell, or Surface Elements

You can specify one or multiple layers of reinforcement in membrane, shell, or surface elements. For each layer, you specify the rebar properties including the rebar layer name; the cross-sectional area of each rebar; the rebar spacing in the plane of the membrane, shell, or surface element; the position of the rebars in the thickness direction (for shell elements only), measured from the midsurface of the shell (positive in the direction of the positive normal to the shell); the rebar material name; the initial angular orientation, in degrees, measured relative to the local 1-direction; and the isoparametric direction from which the rebar angle output will be measured.

You can model rebar layers in solid (continuum) elements by embedding a set of surface or membrane elements with rebar layers defined as discussed above in a set of host continuum elements.

Input File Usage: Use the following options to define one or more rebar layers in membrane elements:

**MEMBRANE SECTION*, ELSET=*memb_set_name*
**REBAR LAYER*

Use the following options to define one or more rebar layers in shell elements:

**SHELL SECTION*, ELSET=*shell_set_name*
**REBAR LAYER*

Use the following options to define one or more rebar layers in surface elements:

**SURFACE SECTION*, ELSET=*surf_set_name*
**REBAR LAYER*

Use the following option to model rebar layers in solid (continuum) elements:

**EMBEDDED ELEMENT*, HOST ELSET=*solid_set_name*
memb_set_name or *surf_set_name*

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**

Interaction module: **Create Constraint**: **Embedded region**

Assigning a Name to the Rebar Layer

You must assign each layer of rebar in a particular element or element set a separate name. This name can be used in defining rebar prestress and output requests.

Input File Usage: **REBAR LAYER*
rebar layer name

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**: **Layer Name** *rebar layer name*

Specifying Rebar Geometry

The rebar geometry is always defined with respect to a local coordinate system. Defining an appropriate local system is described in the next section. The rebar geometry can be constant, vary as a function of radial position in a cylindrical coordinate system, or vary according to the tire “lift” equation. In each case you must specify the spacing, s, and the area, A, which are used to determine the thickness of the equivalent rebar layer, $t = A/s$, as well as the angular orientation, α, of the rebar with respect to this local system.

In addition, for shell elements you must specify the position of the rebars in the shell thickness direction measured from the midsurface of the shell (positive in the direction of the positive normal to the shell). If the shell's thickness is defined by nodal thicknesses (*Nodal Thicknesses*), this distance will be scaled by the ratio of the thickness defined by the nodal thickness to the thickness defined by the section definition. If the shell's thickness is defined with a distribution (*Distribution Definition*), this distance is scaled by the ratio of the element thickness defined by the distribution to the default thickness.

Defining Rebar with Constant Spacing

You can specify the geometry to be constant in the local rebar coordinate system. In this case the spacing, *s*, is specified as a length measure.

Input File Usage: **REBAR LAYER*, GEOMETRY=CONSTANT

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**: **Rebar geometry: Constant**

Defining Rebar Spacing as a Function of Radial Position

You can specify the spacing, *s*, in terms of angular spacing in degrees as shown in *Figure 1*.

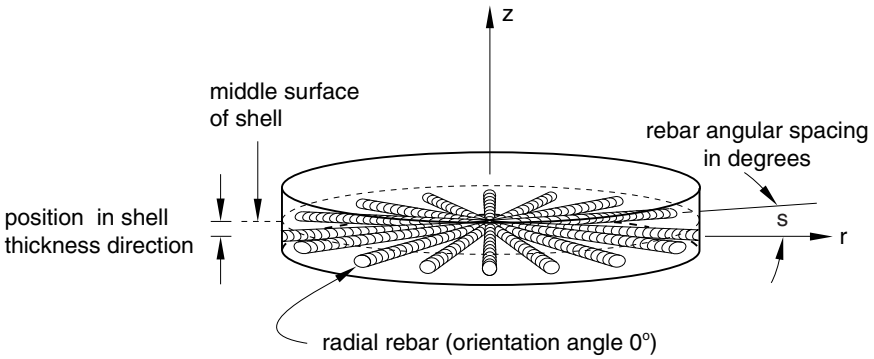


Figure 1: Example of radial rebars in axisymmetric shell elements.

Angular spacing values can also be used for non-radial rebars as well as for rebars having nonzero orientation angles from the meridional plane. In these cases the orientation angles of the rebars do not change. The angular spacing option is used only to compute the spacing between rebars in units of length by multiplying the angular spacing by the radial distance of the concerned point on the rebar from the axis of axisymmetry. A local cylindrical coordinate system must be defined for the rebar if the rebar is associated with three-dimensional elements.

Input File Usage: **REBAR LAYER*, GEOMETRY=ANGULAR

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**: **Rebar geometry: Angular**

Defining Rebar Using the Tire "Lift" Equation

Structural tire analysis is often performed using the cured tire geometry as the reference configuration for the finite element model. However, the cord geometry is more conveniently specified with respect to the "green," or uncured, tire configuration. The tire lift equation provides mapping from the uncured geometry to the cured geometry (see *Figure 2*).

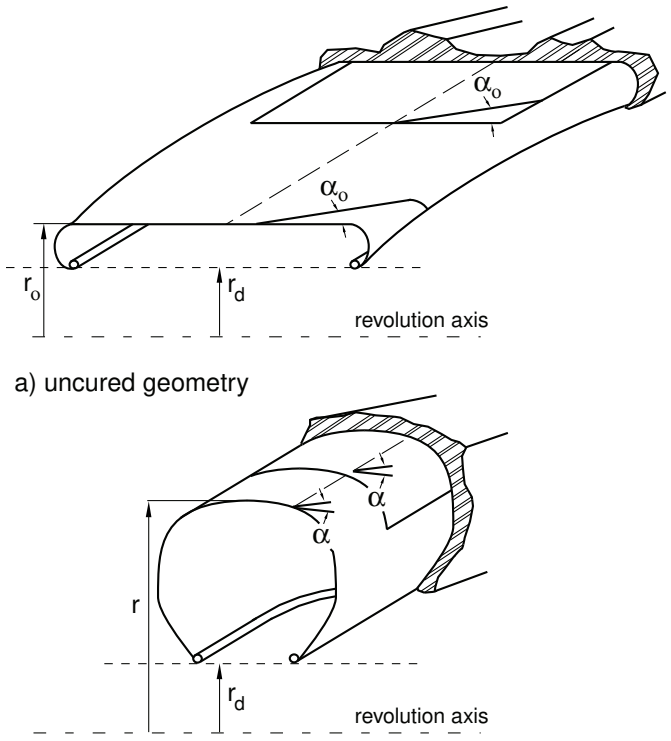


Figure 2: Mapping between uncured and cured tire rebar geometry.

You can specify the spacing and orientation of the rebar cords with respect to the uncured configuration and let Abaqus map these properties to the reference configuration of the cured tire. Using a cylindrical coordinate system, the spacing, *s*, and angular orientation, α, in the cured tire are obtained from

$$\sin\alpha = \frac{r\sin\alpha_0}{r_0(1+e)} \quad \text{and} \quad s = s_0\frac{r\cos\alpha}{r_0\cos\alpha_0},$$

where *r* is the position of the rebar along the radial direction in the cured geometry, r_0 is the position of the rebar in the uncured geometry, s_0 is the spacing in the uncured geometry, α_0 is the angle measured with respect to the projected local 1-direction in the uncured geometry, and *e* is the cord extension ratio. In a tire *e* represents the pre-strain that occurs during the curing process; *e* =1 means a 100% extension. When α_0 is equal to 90°, the rebar is assumed to have a constant spacing of s_0.

A local cylindrical coordinate system must be defined for the rebar if the rebar is associated with three-dimensional elements.

Input File Usage: **REBAR LAYER*, GEOMETRY=LIFT EQUATION

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**: **Rebar geometry: Lift equation–based**

Local Rebar Orientation System

The rebar geometry, such as rebar orientation and spacing, is defined with respect to a local orientation system. This local rebar orientation system is entirely independent from the local orientation system used for the underlying assignment.

The rebar angle is always defined with respect to the local 1-direction as shown in *Figure 3*.

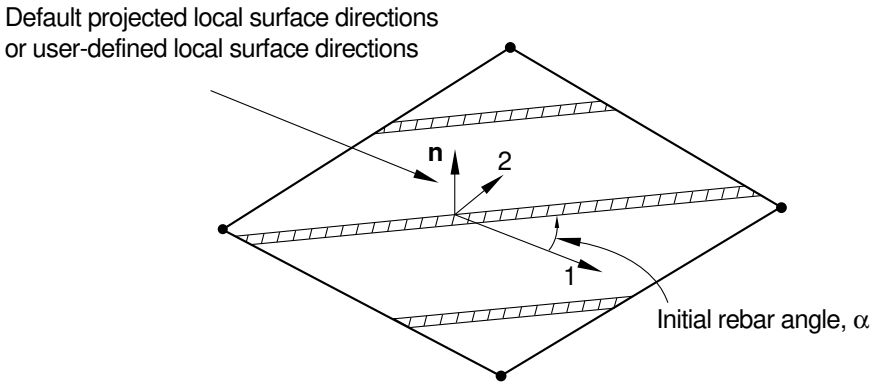


Figure 3: Rebar in a three-dimensional shell, membrane, or surface element.

Rebar defined with either angular spacing or spacing defined by the tire lift equation is specified with respect to a cylindrical orientation system. For axisymmetric analysis the global coordinate system is used as the cylindrical system. For three-dimensional analysis you must provide a user-defined cylindrical orientation definition.

Local Orientation System for Three-Dimensional Elements

You can define the local system by referring to a user-defined local coordinate system. See *Orientations* for a description of how the local coordinate system is calculated from the user-defined directions for definition of rebar in shell, membrane, and surface elements.

If you do not specify a user-defined orientation, the local 1-direction is based on the default projected local coordinate system. See *Conventions* for a definition of the default projected local directions on a surface in space.

A positive angle $\boldsymbol{\alpha}$ defines a rotation from local direction 1 to local direction 2 around the element's normal direction or the user-defined normal direction. If the shell, membrane, or surface element is curved in space, the local 1-direction will vary across the element and the initial rebar angular orientation will also vary accordingly. The orientation definition that can optionally be associated with a shell or membrane section definition has no influence on the rebar angular orientation definitions. For example, in a membrane section, shell section, or surface section, the following data would result in the rebar layer definition shown in *Figure 4*: A=0.01; s=0.1; distance of rebar from the shell midsurface=0.0; $\boldsymbol{\alpha}$=30.; and the rebar definition refers to a local rectangular orientation defined to have its X-axis go through the point (−0.7071, 0.7071, 0.0), its $\boldsymbol{X-Y}$ plane include the point (−0.7071, −0.7071, 0.0), and an additional rotation of 0.0 degrees about the 3-direction.

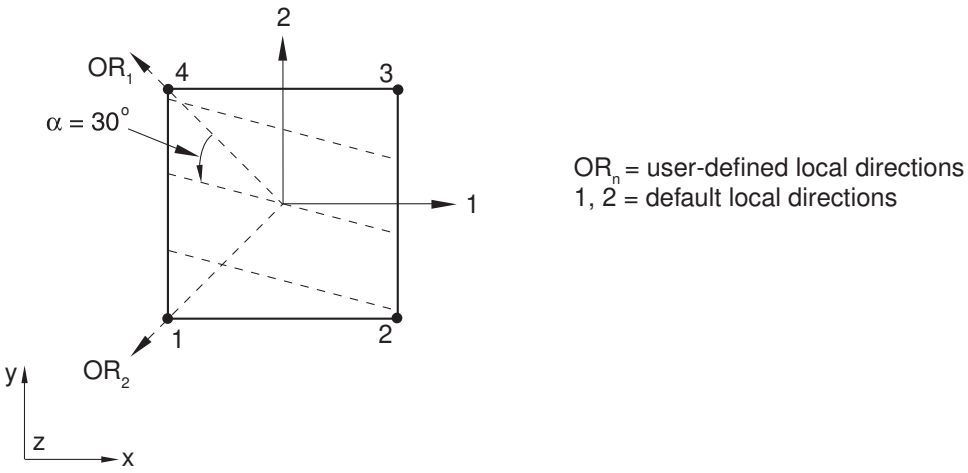


Figure 4: Rebar defined relative to user-defined local coordinate directions.

The following data would result in the rebar layer definition shown in *Figure 5*: A=0.01, s=0.1, distance of rebar from the shell midsurface=0.0, and α=45.

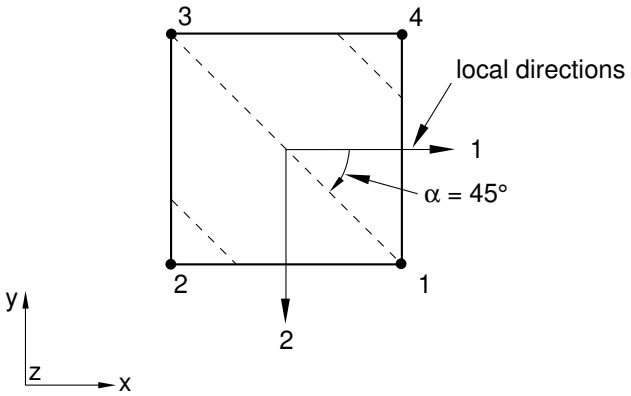


Figure 5: Rebar defined relative to default local coordinate directions.

Input File Usage: Use the following options to define the local 1-direction for a rebar layer:

**ORIENTATION*, NAME=*name*
**REBAR LAYER*, ORIENTATION=*name*

Abaqus/CAE Usage: Property module:
Tools->Datum: **Type**: **CSYS**
Assign->Rebar Reference Orientation

Local Orientation System for Axisymmetric Elements

Rebars in an axisymmetric membrane element or an axisymmetric surface element must lie in the element reference surface, whereas rebars in an axisymmetric shell can lie in the shell reference surface or can be offset from the midsurface. Rebars in axisymmetric membrane, shell, and surface elements can be defined to have any angular orientation with respect to the r–z plane. See *Figure 6* for an example of circumferential rebars and *Figure 1* for an example of radial rebars in axisymmetric shells.

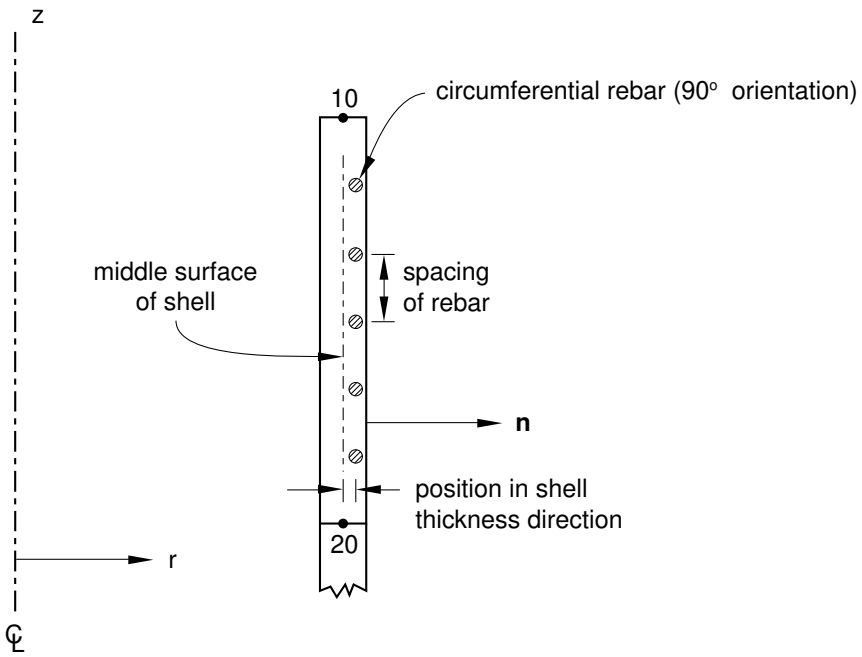


Figure 6: Example of circumferential rebars in axisymmetric shell elements.

You cannot specify a user-defined orientation for rebar layers in axisymmetric membrane, shell, and surface elements. Instead, in the rebar layer definition you specify the angular orientation of the rebar layer, in degrees, with respect to the *r*–*z* plane; this orientation is measured positive about the positive normal to the membrane, shell, or surface element.

If you specify an orientation angle other than 0° or 90° for rebar in an axisymmetric membrane without twist, axisymmetric shell, or axisymmetric surface without twist, Abaqus assumes that the rebars are balanced (i.e., half the rebar lie at the specified angle $\boldsymbol{\alpha}$ and the other half at an angle of $-\boldsymbol{\alpha}$) and internal calculations are handled accordingly. Such a rebar definition should not be used with the symmetric model generation capability (*Symmetric Model Generation*). The recommended modeling technique is to define unbalanced rebar in axisymmetric elements with twist. Balanced rebar, on the other hand, can be defined in regular axisymmetric elements or in axisymmetric elements with twist and should be defined by specifying half the rebar at the specified angle $\boldsymbol{\alpha}$ and the other half at an angle of $-\boldsymbol{\alpha}$.

Large-Displacement Considerations

In geometrically nonlinear analyses as the rebar-reinforced element deforms, the initially defined geometric properties and orientation of the rebar layer can change as a result of finite-strain effects. The deformation of the rebar layer is determined from the deformation gradient of the underlying shell, membrane, or surface element. Rebars rotate with the actual deformation and not with the average rigid body rotation of the material point in the underlying element. See *Rebar modeling in shell, membrane, and surface elements* for details.

For example, consider a plate modeled with a first-order element under large pure shear deformation as shown in *Figure 7*, where rebars are initially aligned with the element isoparametric directions.

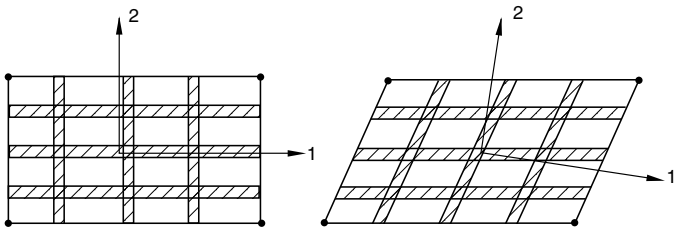


Figure 7: Rebar orientation evolves in a geometrically nonlinear analysis.

As a result of finite-strain effects, rebars rotate but remain aligned with the element isoparametric directions. If the same problem is modeled using anisotropic material properties rather than rebars and the material directions (1 and 2) are initially aligned with the element isoparametric directions, under such large shear deformation the material directions rotate and are no longer aligned with the element isoparametric directions. The material directions in this case are determined based on the average rigid body rotation of the material point. Hence, if the material is not truly a continuum, the anisotropic behavior is better modeled with rebars.

Defining Rebar Lines in Beam Elements

You can specify one or multiple rebar lines of reinforcement in beam elements. For each rebar line, you specify the rebar properties including the rebar line name, the cross-sectional area of each rebar, the positions of the rebars with respect to the local beam section axis, and the rebar material name.

Input File Usage: Use the following option to define one or more rebar lines in beam elements:

**BEAM SECTION*, ELSET=*beam_set_name*
**REBAR LINE*

Assigning a Name to the Rebar Line

You must assign each rebar in a beam section a separate name. This name can be used in defining rebar prestress and output requests.

Input File Usage: **REBAR LINE*
rebar line name

Specifying Rebar Line Geometry

You must specify the cross-sectional area of each rebar and the location of each rebar with respect to the local beam section axis.

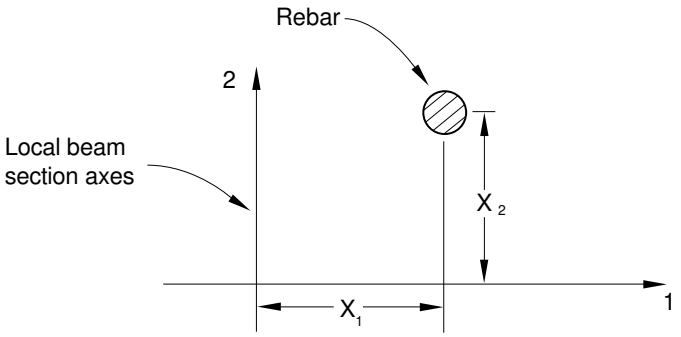


Figure 8: Rebar location in a beam section.

Defining Rebar Materials

The material properties of the rebars are distinct from those of the underlying element and are defined by a separate material definition (*Material Data Definition*). You must associate each rebar layer (or, for beam elements, each rebar line) with a set of material properties.

The following material behavior cannot be used in Abaqus/Standard to define rebar materials:

- *Porous Metal Plasticity*.

The following material behaviors cannot be used in Abaqus/Explicit to define rebar materials:

- *Defining Fully Anisotropic Elasticity*;
- *Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*;
- *Equation of State*;
- *Hill Anisotropic Yield/Creep*;
- *Porous Metal Plasticity*;
- *Extended Drucker-Prager Models*;
- *Modified Drucker-Prager/Cap Model*;
- *Crushable Foam Plasticity Models*; or
- *Cracking Model for Concrete*.

Although Abaqus/Standard will allow for a rebar material to be defined with orthotropic elasticity (*Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*) or anisotropic elasticity (*Defining Fully Anisotropic Elasticity*), $\boldsymbol{D}_{1111}$ is the only meaningful material constant in these definitions. $\boldsymbol{D}_{1111}$ is used to compute the strain in the rebar direction, $\boldsymbol{\varepsilon}_{11}$, using the corresponding stress component, $\boldsymbol{\sigma}_{11}$, as discussed in *Linear Elastic Behavior*; no other strain or stress components exist in rebars.

In Abaqus/Explicit, if a nonzero density is specified for the material in a rebar layer or line, the mass of the rebar is taken into account for the dynamic analysis as well as for gravity loads.

In Abaqus/Standard, mass is not taken into account for reinforcements in beam elements; you should adapt the density of the beam material to account for the rebar mass.

Input File Usage: **REBAR LAYER*
rebar layer name, *A*, *s*, *distance of rebar from shell midsurface*,
rebar material name

Abaqus/CAE Usage: Property module: membrane, shell, or surface section editor: **Rebar Layers**: **Material** *rebar material name*

Initial Conditions

Initial conditions (*Initial Conditions*) can be used to define prestress or solution-dependent values for rebars.

Defining Prestress in Rebar

For structures in which reinforcing is defined (such as reinforced concrete structures), you can use initial conditions to define the prestress in the rebars.

In such cases in Abaqus/Standard the structure must be brought to a state of equilibrium before it is actively loaded by means of an initial static analysis step (*Static Stress Analysis*) with no external loads applied (or, perhaps, with the "dead" loads only)—see *Initial Conditions*.

Input File Usage: **INITIAL CONDITIONS*, TYPE=STRESS, REBAR
element number or element set name, rebar name, prestress value

Abaqus/CAE Usage: Rebar prestress is not supported in Abaqus/CAE.

Holding Prestress in Rebar in Abaqus/Standard

If prestress is defined in the rebars and unless the prestress is held fixed, it will be allowed to change during an equilibrating static analysis step; this is a result of the straining of the structure as the self-equilibrating stress state establishes itself. An example is the pretension type of concrete prestressing in which reinforcing tendons are initially stretched to a desired tension before being covered by concrete. After the concrete cures and bonds to the rebar, release of the initial rebar tension transfers load to the concrete, introducing compressive stresses in the concrete. The resulting deformation in the concrete reduces the stress in the rebar.

Alternatively, you can keep the initial stress defined in some or all of the rebars constant during this initial equilibrium solution. An example is the post-tension type of concrete prestressing; the rebars are allowed to slide through the concrete (normally they are in conduits), and the prestress loading is maintained by some external source (prestressing jacks). The magnitude of the prestress in the rebar is normally part of the design requirements and must not be reduced as the concrete compresses under the loading of the prestressing. Normally, the prestress is held constant only in the first step of an analysis. This is generally the more common assumption for prestressing.

If the prestress is not held constant in analysis steps following the step in which it is held constant, the stress in the rebar will change due to additional deformation in the concrete. If there is no additional deformation, the stress in the rebar will remain at the level set by the initial conditions. If the loading history is such that no plastic deformation is induced in the concrete or rebar in steps subsequent to the steps in which the prestress is held constant, the stress in the rebar will return to the level set by the initial conditions upon removal of the loading applied in those steps.

Input File Usage: **PRESTRESS HOLD*

Abaqus/CAE Usage: Rebar prestress is not supported in Abaqus/CAE.

Defining the Initial Values of Solution-Dependent State Variables for Rebars

You can define the initial values of solution-dependent state variables for rebars within elements. See *Initial Conditions* for details.

Input File Usage: **INITIAL CONDITIONS*, TYPE=SOLUTION, REBAR

Abaqus/CAE Usage: Initial solution-dependent state variables are not supported in Abaqus/CAE.

Output

Rebar force output is available at the rebar integration locations with output variable RBFOR. The rebar force is equal to the rebar stress times the current rebar cross-sectional area. The current cross-sectional area of the rebar is calculated by assuming the rebar is made of an incompressible material, regardless of the actual material definition. For rebars in membrane, shell, or surface elements output variables RBANG and RBROT identify the current orientation of rebar within the element and the relative rotation of the rebar as a result of finite

deformation, respectively. These quantities are measured with respect to the user-specified isoparametric direction in the element, not the default local element system or the orientation-defined system. See *Rebar modeling in shell, membrane, and surface elements*.

See *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers* for information on additional output quantities such as stress and strain. For rebars in membrane, shell, surface, or beam elements with multiple integration points, output quantities are available at the integration points and at the centroid of the element.

Specifying the Direction for Rebar Angle Output

The output quantities RBANG and RBROT can be measured from either of the isoparametric directions in the plane of the membrane, shell, or surface elements. You can specify the desired isoparametric direction from which the rebar angle will be measured (1 or 2). The rebar angle is measured from the isoparametric direction to the rebar with a positive angle defined as a counterclockwise rotation around the element's normal direction. The default direction is the first isoparametric direction.

In axisymmetric shell, surface, and membrane elements the first isoparametric direction coincides with the meridional direction, and the second isoparametric direction coincides with the hoop direction. In triangular elements Abaqus defines the isoparametric directions as follows: for a 3-node triangle the first isoparametric direction is a straight line going from node 1 to the midpoint of the second element edge, and the second isoparametric direction is a straight line going from the midpoint of the first element edge to the midpoint of the third element edge; for a 6-node triangle the first isoparametric direction is a straight line going from node 1 to node 5, and the second isoparametric direction is a straight line going from node 4 to node 6 (see *About the Element Library* for the element node ordering).

Input File Usage: **REBAR LAYER*
rebar layer name, *A*, *s*, *distance of rebar from shell midsurface*,
rebar material name, *angular orientation of rebar*, *isoparametric direction*

Abaqus/CAE Usage: You cannot specify the direction for rebar angle output in Abaqus/CAE; the first isoparametric direction is always used.

Example

As an example, a user-defined local coordinate system is used to define rebar in a shell element ($\alpha = 30°$), and the output value of RBANG is 75°, as illustrated in *Figure 9*:

```
*REBAR LAYER, ORIENTATION=ORIENT
 Rbname, 0.01, 0.1, 0.0, Rbmat, 30., 2
*ORIENTATION, SYSTEM=RECTANGULAR, NAME=ORIENT
 -0.7071, 0.7071, 0.0, -0.7071, -0.7071, 0.0
 3, 0.0
```

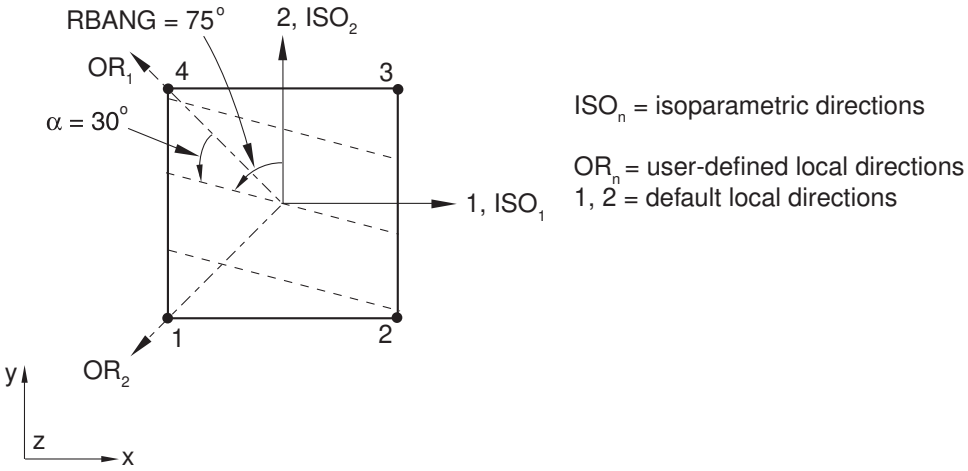


Figure 9: RBANG measurement for rebar defined relative to user-defined local coordinate directions.

The rebars are located at the midsurface of the shell. Output variable RBANG is measured from the second isoparametric direction to the rebar. If the first isoparametric direction were chosen instead, output variable RBANG would report an angle of 165°.

Visualizing Rebar Orientation and Results in Rebar

Abaqus/CAE supports visualization of results in rebar layers or rebar lines and visualization of rebar direction only in rebar layers. Plots of rebar orientation are available only if you request element output for rebars (see *Writing Element Output to the Output Database*). You can contour element variables for rebar as field output or plot them as history output in the Visualization module. Each rebar layer or rebar line will have a unique name and represents one additional section point in a membrane, shell, surface, or beam element. You can select a named rebar layer or rebar line in a membrane, shell, surface, or beam element to display its results in the Visualization module.

Defining Rebar as an Element Property

Products: Abaqus/Standard Abaqus/Explicit

References:

- **PRESTRESS HOLD*
- **REBAR*

Overview

You can define rebar in shell, membrane, continuum, and beam elements as an element property. However, this method is more cumbersome than the method described in *Defining Reinforcement* and does not allow visualization of the rebar and rebar results in Abaqus/CAE.

The preferred method for defining rebar in shell, membrane, and beam elements is defining rebar layers (rebar lines for beam elements) of reinforcement as part of the element section definition (documented in *Defining Reinforcement*). The preferred method for defining rebar in solids is embedding reinforced surface or membrane elements in "host" solid elements as described in *Embedded Elements*.

Element-based rebars:

- are used to define uniaxial reinforcement in solid, membrane, shell, and beam elements;
- can be defined as individual reinforcing bars in solid elements;
- can be defined as layers of uniformly spaced reinforcing bars in shell, membrane, and solid elements (such layers are treated as a smeared layer with a constant thickness equal to the area of each reinforcing bar divided by the reinforcing bar spacing);
- can be used with coupled temperature-displacement elements but do not contribute to the thermal conductivity and specific heat;
- can be used with coupled thermal-electrical-structural elements but do not contribute to the electrical conductivity, thermal conductivity and specific heat;
- do not contribute to the mass of the model in Abaqus/Standard;
- cannot be used in elements intended for heat transfer or mass diffusion analysis;
- cannot be used with triangular shell and membrane elements or with triangular, triangular prism, and tetrahedral solid elements; and
- have material properties that are distinct from those of the underlying element.

Assigning a Name to the Rebar Set

You must assign a name to the rebar set. This name can be used in defining rebar prestress and output requests. Each layer of rebar must be assigned a separate name in a particular element or element set.

Input File Usage: **REBAR*, ELEMENT=*elem*, MATERIAL=*mat*, NAME=*name*

Defining Rebars in Three-Dimensional Shell and Membrane Elements

Both isoparametric and skew rebars can be defined in three-dimensional shell and membrane elements. Rebars cannot be used with triangular shells or membranes.

If triangular-shaped shells or membranes are needed, collapsed quadrilateral shells or membranes can be used. The resulting rebar directions will depend on the type of rebar (isoparametric or skew) used. The rebar must be defined carefully since the element is distorted. This technique should be used only in regions of the mesh where results are not critical and stress gradients are not high.

The stiffness calculations for the rebars use the same integration points as the calculations for the underlying shell or membrane elements. See *About Shell Elements* and *Membrane Elements* for more information about shell and membrane elements.

Defining Isoparametric Rebars in Three-Dimensional Shell and Membrane Elements

Isoparametric rebars are aligned along the mapping of constant isoparametric lines in the element (see *Figure 1*).

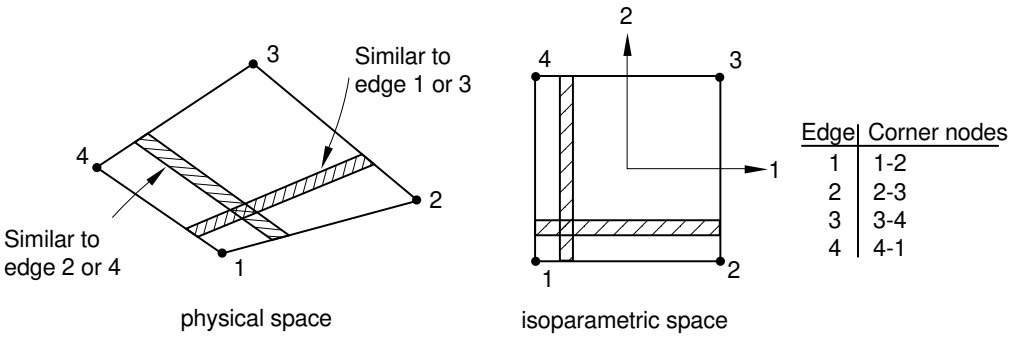


Figure 1: "Isoparametric" rebar in an undistorted three-dimensional shell or membrane element.

If opposite edges of the element containing the rebar are not parallel, the rebar directions will be different at each of the integration points within an element (see *Figure 2*).

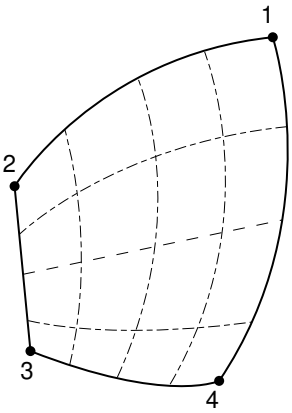


Figure 2: "Isoparametric" rebar directions in a distorted three-dimensional shell or membrane element (dashed lines indicate rebar directions).

The spacing of the rebar will be fixed in physical space. The spacing, s, and the area of the rebar, A, are used to determine the thickness of the equivalent smeared layer, $t = A/s$. If the edges of the element containing the rebar are not parallel, the number of actual rebar with this spacing passing through one edge will be different than the number passing through the opposite edge (opposite in isoparametric space).

You specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the rebar spacing in the plane of the shell, s; and the edge number to which the rebars are parallel when plotted in isoparametric space (see *Figure 1*). In addition, for shell elements you specify the position of the rebars in the shell thickness direction measured from the midsurface of the shell (positive in the direction of the positive normal to the shell). If the

shell's thickness is defined by nodal thicknesses (*Nodal Thicknesses*), this distance is scaled by the ratio of the thickness defined by the nodal thickness to the thickness defined by the section definition. If the shell's thickness is defined with a distribution (*Distribution Definition*), this distance is scaled by the ratio of the element thickness defined by the distribution to the default thickness. If the shell has a composite section whose layer thicknesses are defined with distributions (*Distribution Definition*), this distance is scaled by the ratio of the sum of the element layer thicknesses defined by the distributions to the sum of the default layer thicknesses.

Input File Usage: Use the following option to define isoparametric rebars in three-dimensional shell elements:

**REBAR*, ELEMENT=SHELL, MATERIAL=*mat*, GEOMETRY=ISOPARAMETRIC

Use the following option to define isoparametric rebars in general membrane elements:

**REBAR*, ELEMENT=MEMBRANE, MATERIAL=*mat*, GEOMETRY=ISOPARAMETRIC

Defining Skew Rebars in Three-Dimensional Shell and Membrane Elements

Skew rebars need not be similar to an element edge; they can lie at any prescribed angle from the local 1-axis. The direction of the rebars must be defined in one of two ways, as indicated in *Figure 3*:

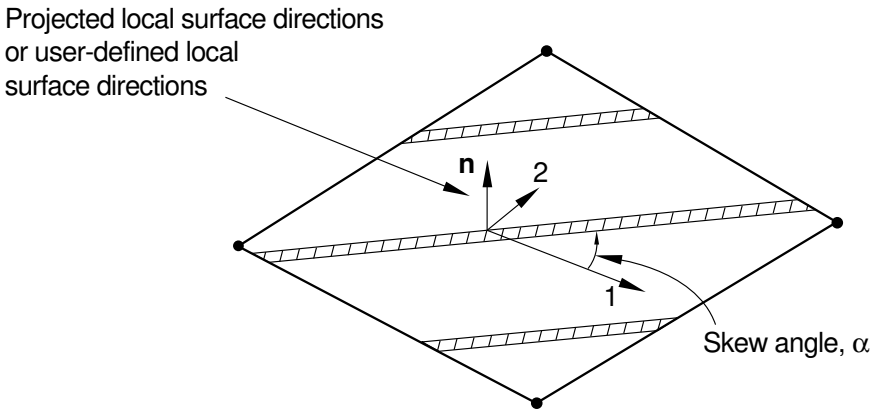


Figure 3: "Skew" rebar in a three-dimensional shell or membrane.

1. The rebars can be defined relative to the default projected local coordinate system (see *Conventions*).
2. The rebars can be defined relative to a user-defined local coordinate system (see *Orientations*).

The orientation definition that can optionally be associated with a shell or membrane section definition has no influence on the rebar angular orientation definitions. If the shell or membrane is curved in space, the local 1-direction will vary across the element and the skew rebar will also vary accordingly.

For shell elements the definition of a local coordinate system using distributions (*Distribution Definition*) has no influence on the rebar angular orientation definitions.

If the rebar cross-sectional area is A, the rebar spacing, s, should be given so that the thickness of the equivalent "smeared" layer of reinforcing is $t = A/s$.

Defining Skew Rebars Relative to the Default Projected Local Coordinate System

To define skew rebars relative to the default projected local coordinate system, you specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the rebar spacing in the plane of the shell, s; the

position of the rebars in the thickness direction (for shell elements only), measured from the midsurface of the shell (positive in the direction of the positive normal to the shell); and the angle α, in degrees, between the default local 1-direction and the rebars. See *Conventions* for a definition of the default projected local directions on a surface in space. If the shell's thickness is defined by nodal thicknesses (*Nodal Thicknesses*), the rebar position in the thickness direction will be scaled by the ratio of the thickness defined by the nodal thickness to the thickness defined by the section definition. If the shell's thickness is defined with a distribution (*Distribution Definition*), the rebar position in the thickness direction is scaled by the ratio of the element thickness defined by the distribution to the default thickness. A positive angle α defines a rotation from local direction 1 to local direction 2 around the element's normal direction. For example, in a membrane the following data would result in the rebar definition shown in *Figure 4*: A=0.05, s=0.1, and α=45.

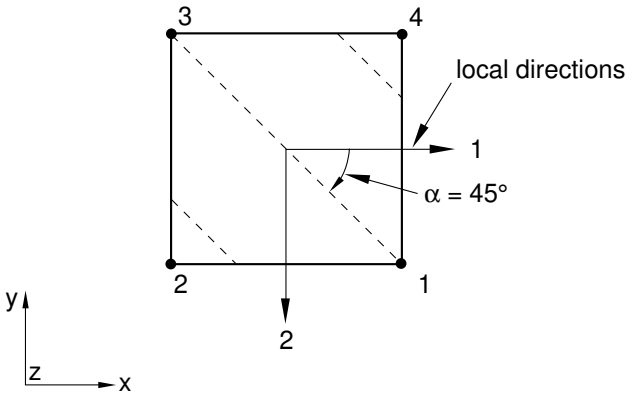


Figure 4: Skew rebar defined relative to default local coordinate directions.

When a user-defined local orientation definition is not used to define the angular orientation of the rebar and the normal to the shell is nearly parallel to the global 1-axis, the local 1-axis may change significantly within an element or from one element to the next (see *Conventions*).

Input File Usage: Use the following option to define skew rebars relative to the default projected local coordinate system in three-dimensional shell elements:

**REBAR*, ELEMENT=SHELL, MATERIAL=*mat*, GEOMETRY=SKEW

Use the following option to define skew rebars relative to the default projected local coordinate system in general membrane elements:

**REBAR*, ELEMENT=MEMBRANE, MATERIAL=*mat*, GEOMETRY=SKEW

Defining Skew Rebars Relative to a User-Defined Local Coordinate System

To define skew rebars relative to a user-defined local coordinate system, you specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the rebar spacing in the plane, s; the position of the rebars in the thickness direction (for shell elements only), measured from the midsurface of the shell (positive in the direction of the positive normal to the shell); and the angle, α, in degrees, between the user-defined 1-direction and the rebars. See *Orientations* for a description of how the local coordinate system is calculated from the user-defined directions for definition of rebar in shells and membranes. A positive angle α defines a rotation from local direction 1 to local direction 2 around the user-defined normal direction. For example, in a shell the following data would result in the skew rebar definition shown in *Figure 5*: A=0.01; s=0.1; distance of rebar from the shell midsurface=0.0; α=30.; and the rebar definition refers to a local rectangular orientation defined to have its X-axis go through the point (–0.7071, 0.7071, 0.0), its X–Y plane include the point (–0.7071, –0.7071, 0.0), and an additional rotation of 0.0 degrees about the 3-direction.

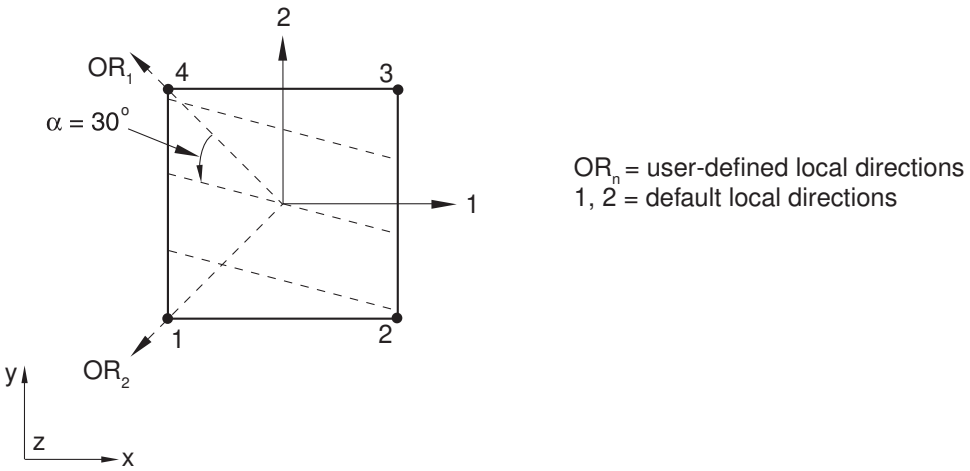


Figure 5: Skew rebar defined relative to user-defined local coordinate directions.

Input File Usage: Use the following option to define skew rebars relative to a user-defined local coordinate system in three-dimensional shell elements:

**REBAR*, ELEMENT=SHELL, MATERIAL=*mat*, GEOMETRY=SKEW, ORIENTATION=*name*

Use the following option to define skew rebars relative to a user-defined local coordinate system in general membrane elements:

**REBAR*, ELEMENT=MEMBRANE, MATERIAL=*mat*, GEOMETRY=SKEW, ORIENTATION=*name*

Defining Rebars in Axisymmetric Shell and Membrane Elements

Rebars in an axisymmetric membrane must lie in the membrane reference surface, whereas rebars in an axisymmetric shell can lie in the shell reference surface or can be offset from the midsurface. Rebars in axisymmetric shells and membranes can be defined to have any orientation with respect to the *r*–*z* plane. See *Figure 6* for an example of circumferential rebars and *Figure 7* for an example of radial rebars in axisymmetric shells.

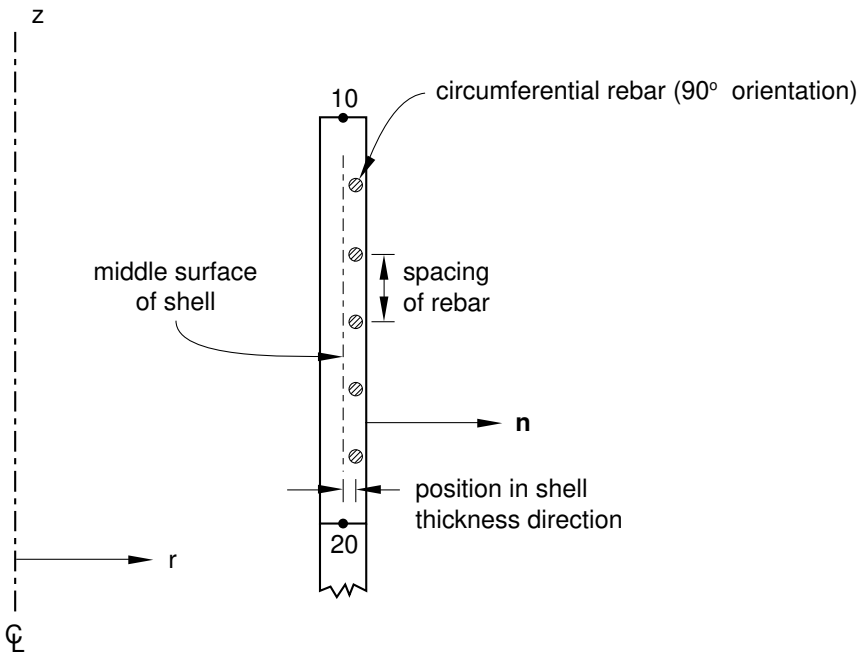


Figure 6: Example of circumferential rebars in axisymmetric shell elements.

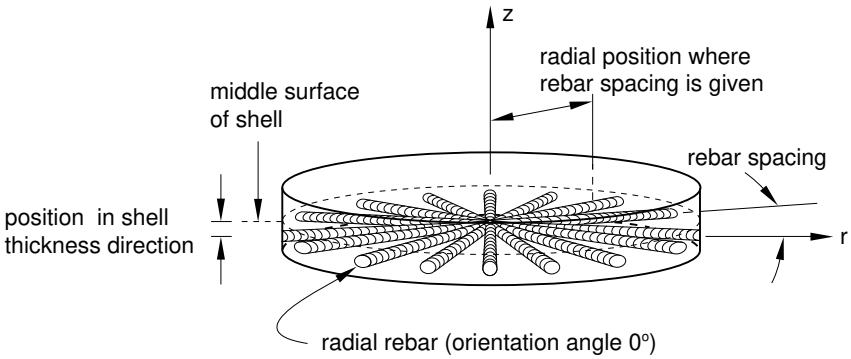


Figure 7: Example of radial rebars in axisymmetric shell elements.

You specify the cross-sectional area, A, of each rebar; the rebar spacing, s; for shell elements the position of the rebars in the shell thickness direction, measured from the midsurface of the shell (positive in the direction of the positive normal to the shell); the angular orientation with respect to the r–z plane, $\boldsymbol{\alpha}$, measured in degrees; and the radial position at which the rebar spacing is measured. The angular orientation is measured positive about the positive normal to the shell or membrane element. If the shell's thickness is defined by nodal thicknesses (*Nodal Thicknesses*), the distance from the midsurface will be scaled by the ratio of the thickness defined by the nodal thickness to the thickness defined by the section definition. If the shell's thickness is defined with a distribution (*Distribution Definition*) the distance from the midsurface will be scaled by the ratio of the element thickness defined by the distribution to the default thickness.

If an orientation angle other than 0 or 90° is specified for rebar in an axisymmetric shell or membrane without twist, Abaqus assumes that the rebars are balanced (i.e., half the rebar lie at the specified angle $\boldsymbol{\alpha}$ and the other half at an angle of $-\boldsymbol{\alpha}$) and internal calculations are handled accordingly. See *Rebar modeling in two dimensions* for details. If the symmetric model generation capability (*Symmetric Model Generation*) is used to create a three-dimensional model from an axisymmetric shell or membrane model, only balanced rebars will be translated appropriately. The definition of balanced rebars in the axisymmetric model will result in balanced rebars in the

three-dimensional model; such a translation with unbalanced rebars is not available. Unbalanced rebars in generalized axisymmetric membranes with twist will be translated properly.

If the radial position for the rebar spacing is given, the total cross-sectional area of rebar will remain constant as the radial position changes; this behavior corresponds to the number of rebar in the circumferential direction remaining constant and implies that the thickness of the smeared layer of rebar decreases and that the spacing of the rebars increases as r increases (see *Figure 7*). If the radial position for the rebar spacing is omitted (or is set to zero), Abaqus assumes that the spacing of the rebar remains constant; the thickness of the corresponding smeared layer is held fixed such that $t = A/s$.

Input File Usage: Use the following option to define rebars in an axisymmetric shell element:

**REBAR*, ELEMENT=AXISHELL, MATERIAL=*mat*

Use the following option to define rebars in an axisymmetric membrane element:

**REBAR*, ELEMENT=AXIMEMBRANE, MATERIAL=*mat*

Defining Rebars in Beam Elements

You can use element-based rebar to model discrete rebar in beam elements in Abaqus/Standard and Abaqus/Explicit. You specify the elements that contain the rebar, the cross-sectional area of each rebar, and the location of each rebar with respect to the local beam section axis (see *Figure 8*).

You must assign each individual rebar a separate name in a particular element or element set.

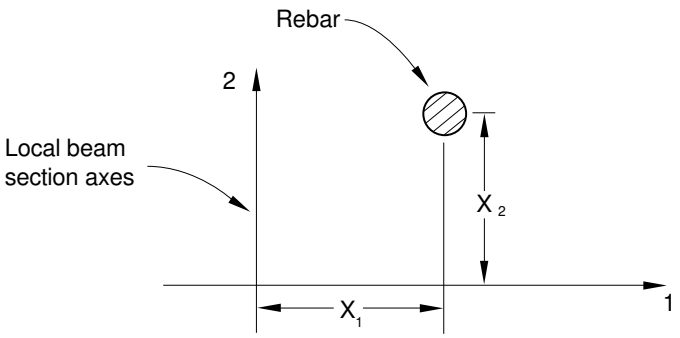


Figure 8: Rebar location in a beam section.

Input File Usage: Use the following option to define rebar in beam elements:

**REBAR*, ELEMENT=BEAM, MATERIAL=*mat*, NAME=*name*

Defining Rebars in Continuum Elements

Two- or three-dimensional continuum (solid) elements can contain rebars; rebars cannot be defined in triangular, prism, tetrahedral, or infinite elements. If triangular or wedge-shaped elements are needed, collapsed quadrilateral or brick elements can be used. Be careful when collapsing elements that contain rebar. It is important to check that the location and orientation of the rebar are correct.

Rebars are defined as single bars or in layers. In the latter case the layer is a surface in each element; you provide the rebar orientation in the surface.

Defining Layers of Rebars in Planar and Axisymmetric Continuum Elements

By default, the rebars form a layer that lies in a surface that is at right angles to the plane of the model. You define the line where this rebar surface intersects the plane of the model, as described below.

The orientation of the rebars within the rebar surface is defined by giving an angle, in degrees, between the line of intersection in the plane of the model and the rebars. This angle is measured in physical three-dimensional space, not in isoparametric space. See *Rebar modeling in two dimensions* for details. The positive direction along the line of intersection is from the lower to the higher numbered element edge that is intersected, and a positive angle indicates rebars oriented down into the plane of the model (where the plane is parallel to the z-axis in plane strain analysis or the θ-axis for axisymmetric analysis), as shown in *Figure 9*.

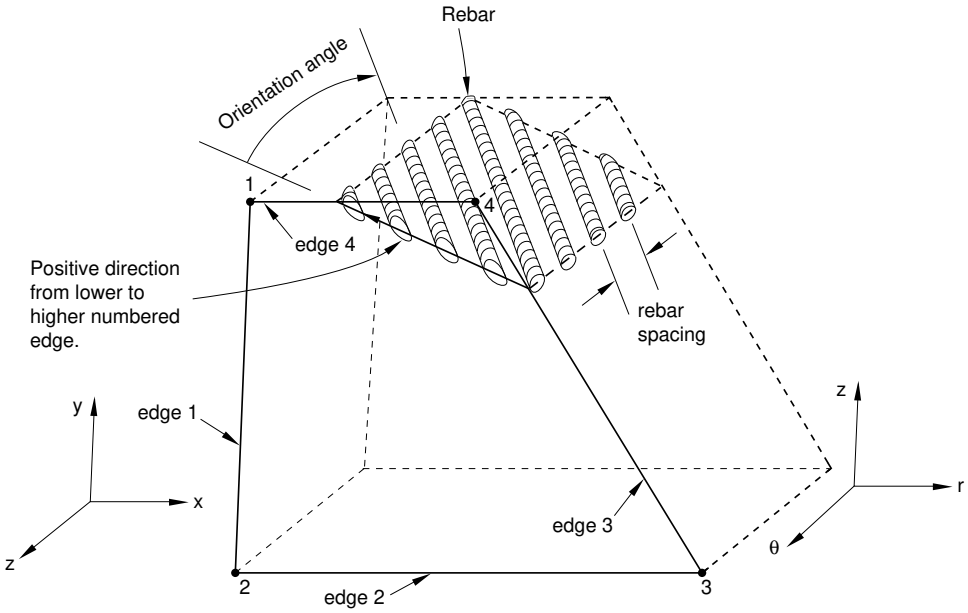


Figure 9: Orientation of rebars in plane and axisymmetric solid elements.

If an orientation angle other than 0 or 90° is specified for rebar in an axisymmetric element without twist, it is assumed that the rebar in the element are balanced (i.e., half the rebar lie at the specified angle α and the other half at the angle $-\alpha$).

Defining Isoparametric Rebars

For isoparametric rebars the intersection of the rebar layer with the plane of the model will lie along the mapping of a constant isoparametric line in the element. You specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the rebar spacing, s; the rebar orientation, α (as described above); the fractional distance *from* the edge, f (the ratio of the distance between the edge and the rebar to the distance across the element); and the edge number from which the rebars are defined. In addition, for axisymmetric elements you specify the radial position at which the rebar spacing is measured.

If the radial position for the rebar spacing is given for rebar in axisymmetric elements, the total cross-sectional area of rebar will remain constant as the radial position changes; this behavior corresponds to the number of rebar remaining constant as r increases; that is, the thickness of the smeared layer of rebar decreases as r increases. If the radial position for the rebar spacing is omitted (or is set to zero), Abaqus assumes that the spacing of the rebar remains constant; the thickness of the corresponding smeared layer is held fixed such that $t = A/s$.

Figure 10 shows an example of isoparametric rebar.

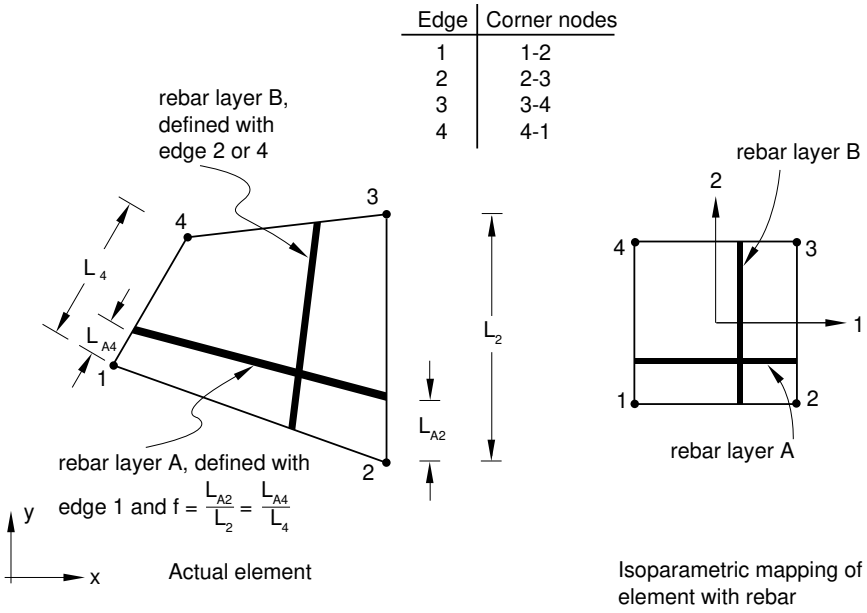


Figure 10: Isoparametric rebar layer definition in solid elements.

In the isoparametric mapping of the element, the line of rebars is parallel to one of the edges of the element. In this figure the line for rebar layer *A* can be defined using edges 1 or 3 and rebar layer *B* can be defined by edges 2 or 4. The fractional distance from edge 1 for rebar layer *A* is the ratio $f_1 = L_{A2}/L_2 = L_{A4}/L_4$; alternatively, layer *A* can be defined from edge 3, so that $f_3 = 1. - L_{A2}/L_2 = 1. - L_{A4}/L_4$.

Input File Usage: Use the following option to define layers of isoparametric rebars in planar and axisymmetric continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*, GEOMETRY=ISOPARAMETRIC

Defining Skew Rebars

For skew rebars the intersection of the rebar layer with the plane of the model can intersect any two edges of an element. You specify the elements that contain the rebars; the cross-sectional area, *A*, of each rebar; the rebar spacing, *s*; and the rebar orientation, α (as described above). In addition, for axisymmetric elements you specify the radial position at which the rebar spacing is measured. You also specify the fractional distance *along* the element edge, from the first node of the edge (as listed in *Figure 11*) to where the rebar layer intersects the edge, for all edges. Only the two values corresponding to the two edges that the rebar intersects can be nonzero.

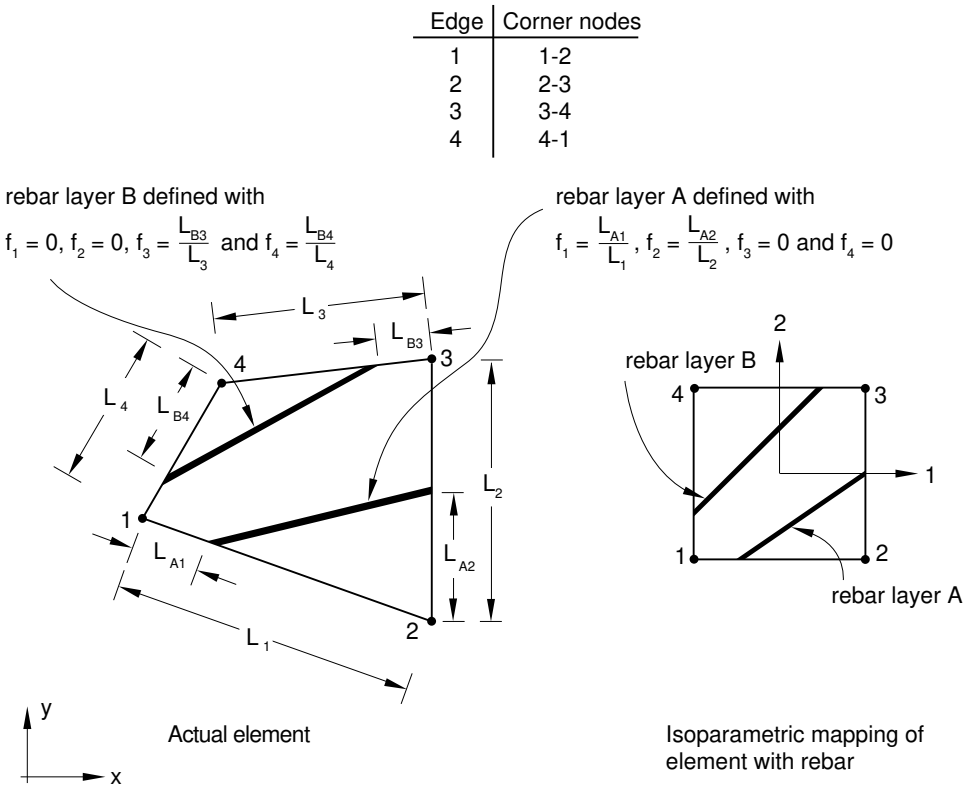


Figure 11: Skew rebar layer definition in solid elements.

Figure 11 shows an example of skew rebar. In the isoparametric mapping of the element, the line of rebars intersects two of the element edges. The intersection points are located by defining a fractional distance along each intersected edge. In this figure rebar layer *A* is defined by the ratio $f_1 = L_{A1}/L_1$ along edge 1 and the ratio $f_2 = L_{A2}/L_2$ along edge 2. Rebar layer *B* is defined by the ratio $f_3 = L_{B3}/L_3$ along edge 3 and the ratio $f_4 = L_{B4}/L_4$ along edge 4.

Defining skew rebars in continuum elements can increase the run time for an Abaqus/Explicit analysis significantly. The element's stable time increment will, in most cases, be determined by the stable time increment of the rebar, which is proportional to the rebar length. The rebar length is determined by factors including the rebar surface position in the element, the rebar spacing, the rebar area, and the rebar orientation within the rebar surface. If a skew rebar in a continuum element is defined such that it intersects two adjacent element edges, the resulting rebar length could be considerably less than the average element edge length, thus resulting in a very small element stable time increment.

Input File Usage: Use the following option to define layers of skew rebars in planar and axisymmetric continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*, GEOMETRY=SKEW

Defining Single Rebars in Two-Dimensional Axisymmetric and Generalized Plane Strain Continuum Elements

You can define single rebars in axisymmetric and generalized plane strain continuum elements. In this case the rebar is assumed to be at right angles with the plane of the model—in the thickness direction for generalized plane strain elements or the hoop direction for axisymmetric elements.

The intersection of the rebar with the plane of the model is defined by the fractional distances along edges 1 and 2 of the intersections of constant isoparametric lines that pass through the rebar location (see *Figure 12*). The fractional distances are measured from the first edge node listed in *Figure 12*.

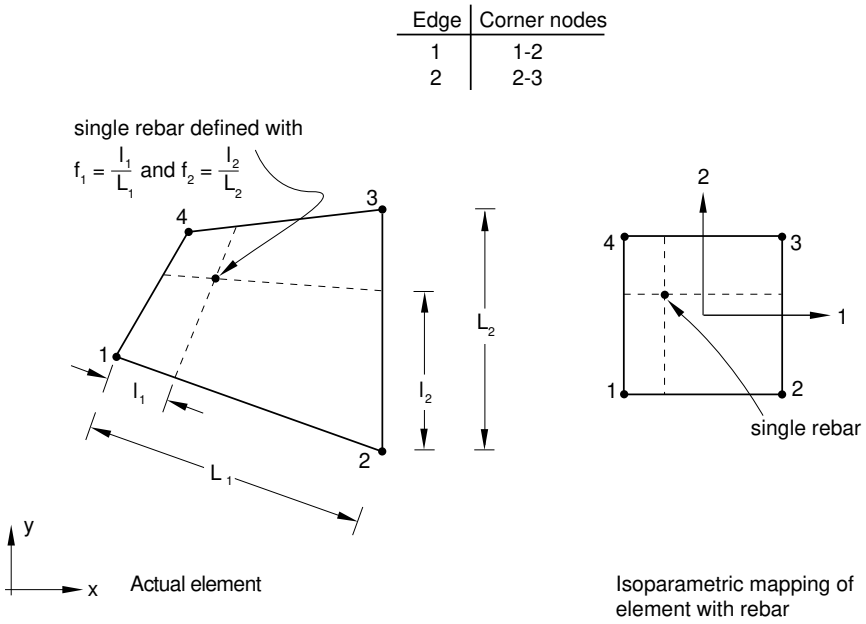


Figure 12: Single rebar in a solid element.

You specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; and the fractional distances locating the rebar's position in the element, f_1 and f_2.

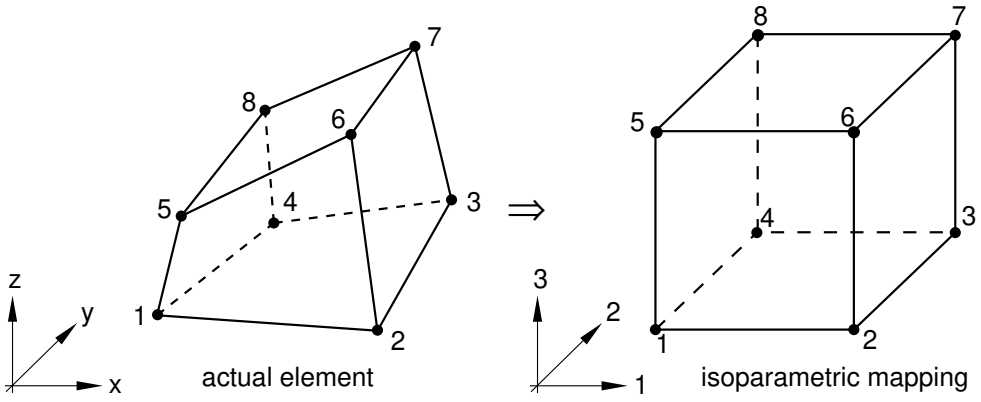
Input File Usage: Use the following option to define single rebars in axisymmetric and generalized plane strain continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*, SINGLE

Defining Layers of Rebars in Three-Dimensional Continuum Elements

By default, the rebars in three-dimensional continuum elements are defined as layers lying in surfaces. The surfaces are most easily defined with respect to the isoparametric mapped cube of the element. Therefore, you must consider how the rebar will be defined before generating the mesh; if the rebar surfaces are not taken into account in designing the mesh, the rebar definition can be very inefficient.

In the isoparametric mapped cube the rebar surface always has two edges (opposite to one another) that are parallel to an isoparametric direction. The isoparametric directions are defined in *Figure 13*. You specify this isoparametric direction (1, 2, or 3).



Isoparametric direction: 1 (parallel to the 1-2 edge of the element and intersecting face 1-4-8-5)

Edge	Corner nodes
1	1-4
2	4-8
3	8-5
4	5-1

Isoparametric direction: 2 (parallel to the 1-4 edge of the element and intersecting face 1-5-6-2)

Edge	Corner nodes
1	1-5
2	5-6
3	6-2
4	2-1

Isoparametric direction: 3 (parallel to the 1-5 edge of the element and intersecting face 1-2-3-4)

Edge	Corner nodes
1	1-2
2	2-3
3	3-4
4	4-1

Figure 13: Isoparametric direction and edge definitions for three-dimensional elements.

A particular face of the element, which is perpendicular to this isoparametric direction in the isoparametric mapped cube, is used to define the position of the other two edges of the surface; the faces are defined in *Figure 13*, where the edges of the faces are also defined.

If isoparametric rebars are defined, the two edges of the rebar surface that are not parallel to the user-specified isoparametric direction will be parallel to one of the other two isoparametric directions; in the isoparametric-mapped cube one isoparametric coordinate is constant on the rebar surface. *Figure 14* illustrates this concept with an element containing two layers of isoparametric rebars.

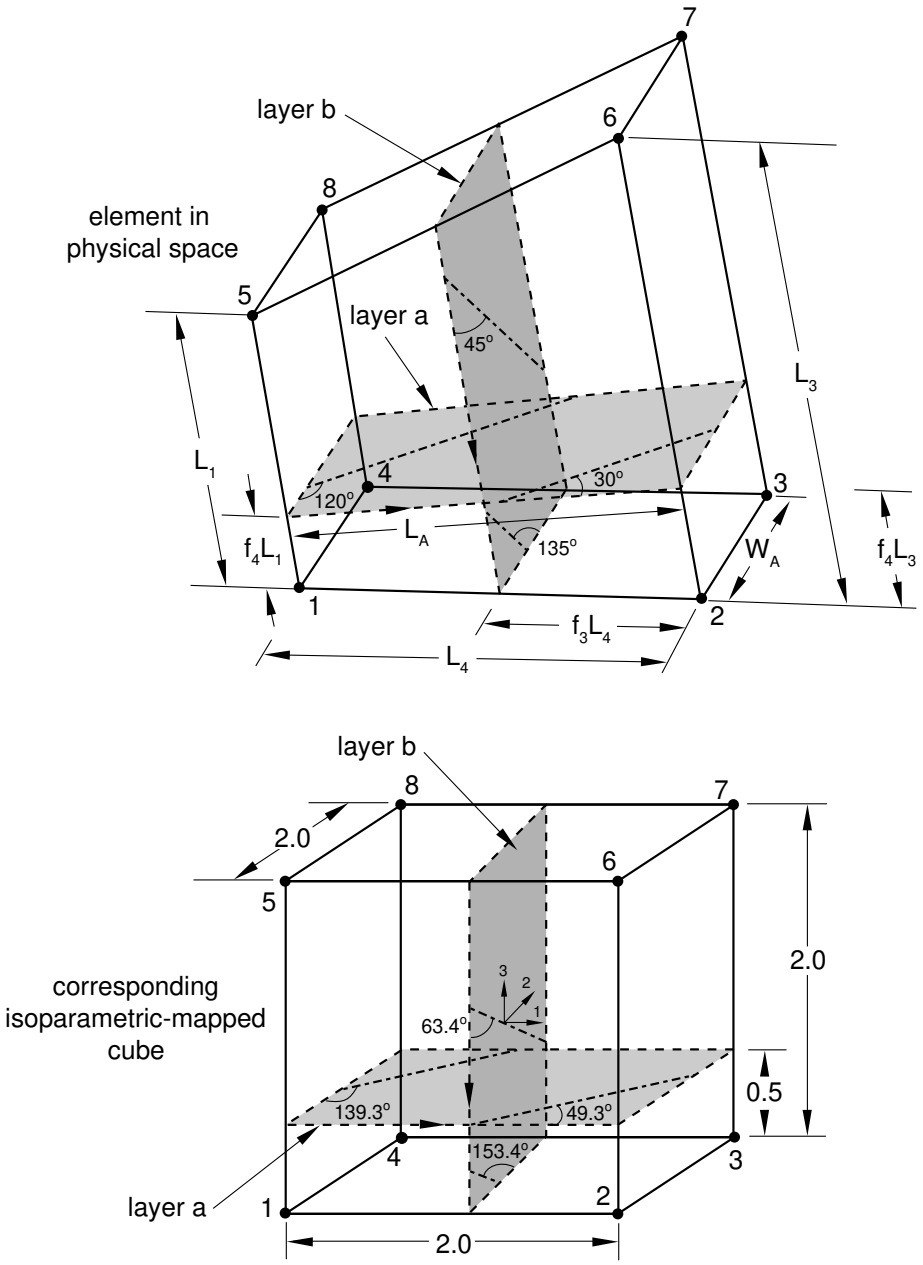


Figure 14: Element with two layers of isoparametric rebar.

The position of each surface is given by the fractional distance *f* from an edge of the face defined in *Figure 13* for the isoparametric direction chosen; you must specify the edge from which the fractional distance is measured.

If skew rebars are defined, the two edges of the rebar surface, which are not parallel to the user-specified isoparametric direction, are generally not parallel to one of the other isoparametric directions. The positions of these two edges of the rebar surface are specified by the intersection of the rebar surface with edges of the intersecting face, defined in *Figure 13*, for the isoparametric direction chosen; the intersections are given by the fractional distance *f* along each edge of the face. (Note that the fractional distance is *along* the edge for skew rebars; for isoparametric rebars the fractional distances are measured *from* an edge.) The fractional distance along an edge is measured from the first node of the edge. All four fractional distances must be given, but only two can be nonzero.

The orientation angle, α, of the rebars within the rebar layer is defined in the isoparametric-mapped cube; it is measured in degrees and is the angle between the line of intersection of the rebar surface with the face for the isoparametric direction chosen and the rebar. The positive direction of the line of intersection is from the lower numbered edge to the higher numbered edge; the positive direction for the rebars points into the elements. An example is shown in *Figure 15*. The orientation angle is defined in the rebar layer in the isoparametric-mapped cube; therefore, the definition is the same for isoparametric and skew rebar.

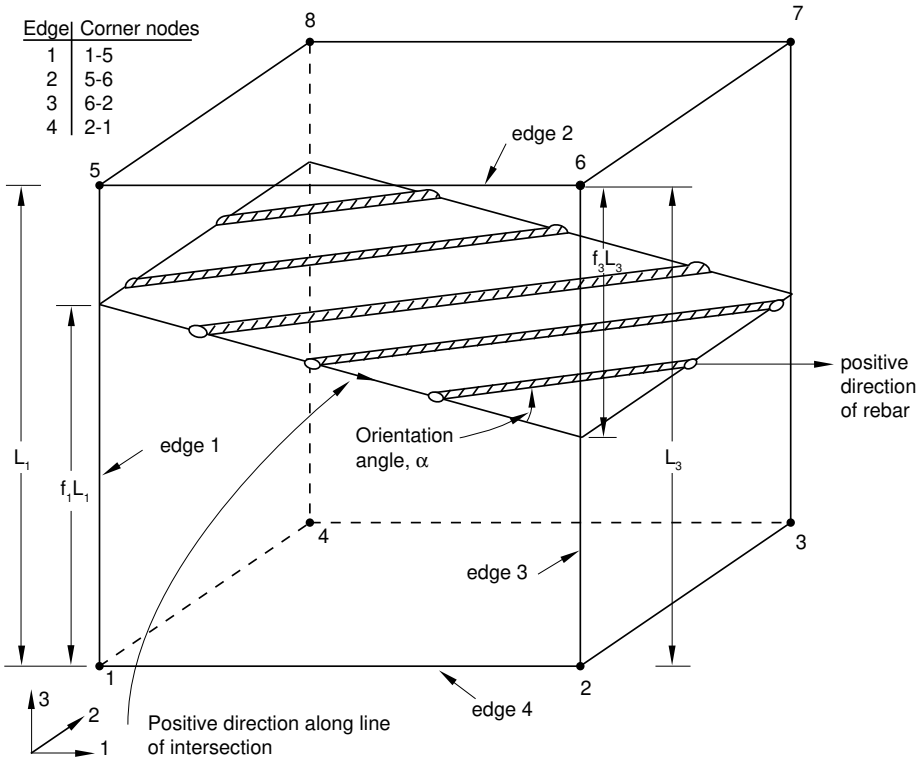


Figure 15: Orientation example for three-dimensional skew rebar modeling, isoparametric direction 2. Shown in the mapped isoparametric element.

If the rebar layer is not flat in physical space, the orientation angle at each integration point may be different. Since it is possible to define only one orientation angle per element, an average value orientation angle for the element must be used; for reasonable meshes this approximation should not affect the results significantly.

Defining Isoparametric Rebars

You specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the rebar spacing, s; the rebar orientation, α (as described above); the fractional distance, f, from the edge; the number of the edge from which the fractional distance is measured; and the isoparametric direction of the rebar surface.

Input File Usage: Use the following option to define layers of isoparametric rebars in three-dimensional continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*,
GEOMETRY=ISOPARAMETRIC

Example: Isoparametric Rebar

For example, the following input defines the isoparametric rebar shown in *Figure 14*:

```
*HEADING
ISOPARAMETRIC REBAR
*NODE
 1,   0.,   0.
 2,  10.,   0.
 3,  10.,   5.
 4,   0.,   5.
 5,   0.,   0.,   7.5
 6,  10.,   0.,  12.5
 7,  10.,   5.,  12.5
 8,   0.,   5.,  7.5
*ELEMENT, TYPE=C3D8R, ELSET=ONE
 1,1,2,3,4,5,6,7,8
*REBAR, ELEMENT=CONTINUUM, MATERIAL=STEEL,
 GEOMETRY=ISOPARAMETRIC, NAME=LAYER_A
 ONE,.04,2.5,49.32628,0.25,4,2
*REBAR, ELEMENT=CONTINUUM, MATERIAL=STEEL,
 GEOMETRY=ISOPARAMETRIC, NAME=LAYER_B
 ONE,.04,1.,63.43494,0.5,3,2
*MATERIAL, NAME=STEEL
*ELASTIC
 30.E6,
 …
```

Rebar layers *A* and *B* are defined using isoparametric direction 2. From *Figure 13* the position of the layers must be given with respect to the face with nodes 1-5-6-2.

The fractional distance defining the position of intersection of layer *A* with this face can be measured from edge 4 (edge with nodes 2–1) along edge 3 (edge with nodes 6–2), as shown in *Figure 14*. For layer *A*, $f_4 = .25$. It could also be given from edge 2 (edge with nodes 5–6), so that $f_2 = 1.0 - f_4 = .75$.

The orientation of rebar for layer *A* in physical space is defined by an angle, β, equal to 30° for layer *A*. This angle must be transformed into the corresponding angle in the isoparametric-mapped cube. This transformation can be done as follows: consider a single rebar that intersects the intersecting line (described above) and an adjacent edge (see *Figure 16*).

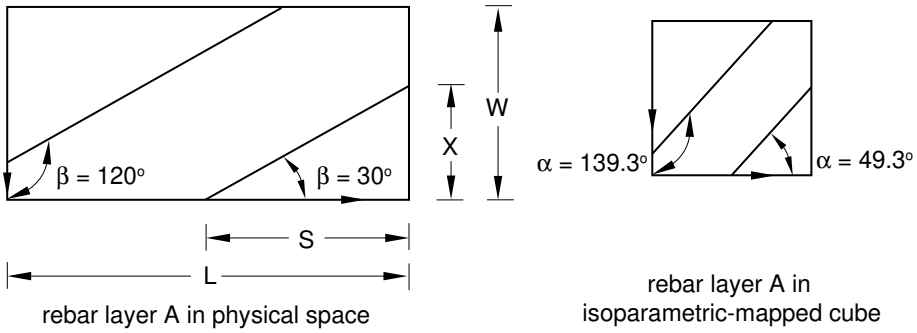


Figure 16: Example defining isoparametric rebar.

From the figure $\tan\beta = X/S$. The length of the rebar layer along the intersecting line is *L*, and the length of the opposite edge is *W*. Consider the same rebar in the rebar layer in the isoparametric-mapped cube. The orientation angle, α, is given by $\tan\alpha = x/s$, where $x = 2X/W$ and $s = 2S/L$. (The 2 is included because the isoparametric-mapped cube is a 2 × 2 × 2 cube.) This expression can be simplified to give

$$\tan\alpha = \frac{X}{W}\frac{L}{S} = \frac{L}{W}\tan\beta.$$

For layer *A*, $L = 10.0778$, $W = 5.$, $\beta = 30°$, and $\alpha = 49.33°$, where α is the orientation angle that must be specified.

The fractional distance defining the position of the intersection of layer *B* with this face can be measured from edge 3 (edge with nodes 6–2); $f_3 = .5$. It could also be measured from edge 1 (edge with nodes 1–5), such that $f_1 = 1.0 - f_3$. The orientation angle for layer *B* in the rebar layer is 45°. In the isoparametric-mapped cube $L = 10.$, $W = 5.$, $\beta = 45°$, and $\alpha = 63.43°$.

Since an isoparametric rebar layer always lies in two of the isoparametric directions, an alternative but equivalent definition can be given. For example, layer *A* also lies in isoparametric direction 1, with the intersecting face having nodes 1-4-8-5. The fractional distance for layer *A*, measured from edge 1 (edge with nodes 1–4), is $f_1 = .25$. The positive sense of the line of intersection is from edge 2 (edge with nodes 4–8) to edge 4 (edge with nodes 5–1); therefore, $\beta = 120°$, $L = 5.$, $W = 10.077$, and $\alpha = 139.32°$.

Layer *B* also lies in isoparametric direction 3, with the intersecting face having nodes 1-2-3-4. The fractional distance for layer *B*, measured from edge 2 (edge with nodes 2–3), is $f_2 = .5$. The positive sense of the intersecting line is from edge 1 (edge with nodes 1–2) to edge 3 (edge with nodes 3–4); therefore, the orientation angle of the rebar in physical space is $\beta = 135°$, $L = 5.$, $W = 10.$, and in the isoparametric-mapped cube $\alpha = 153.43°$.

Defining Skew Rebars

You specify the elements that contain the rebars; the cross-sectional area, *A*, of each rebar; the rebar spacing, *s*; the rebar orientation, α (as described above); and the isoparametric direction. In addition, you specify the fractional distance *falong* the element edge for each edge of the intersecting face defined in *Figure 13*. Only the values corresponding to the two edges that the rebar intersects can be nonzero.

Input File Usage: Use the following option to define layers of skew rebars in three-dimensional continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*, GEOMETRY=SKEW

Example: Skew Rebar

For example, the following input defines the skew rebar shown in *Figure 17*:

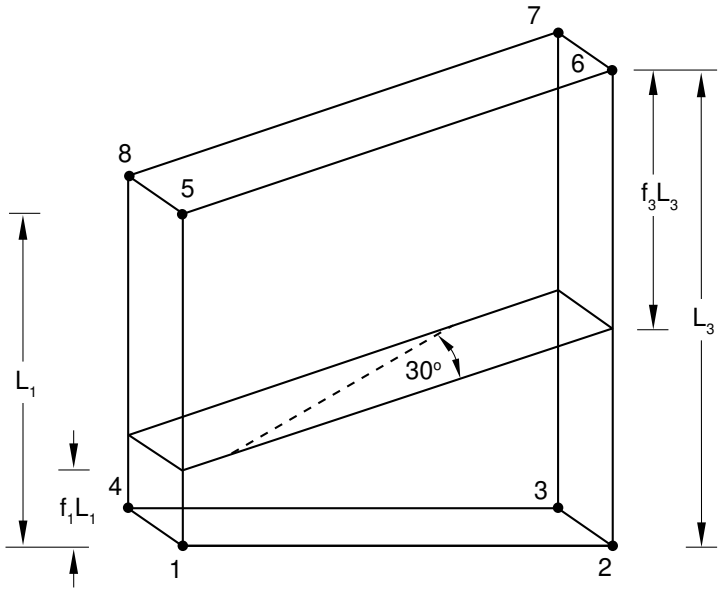


Figure 17: Example defining skew rebar.

```
*HEADING
*NODE
 1,   0.,   0.
 2,  10.,   0.
 3,  10.,   5.
 4,   0.,   5.
 5,   0.,   0.,   7.5
 6,  10.,   0.,  12.5
 7,  10.,   5.,  12.5
 8,   0.,   5.,  7.5
*ELEMENT, TYPE=C3D8R, ELSET=ONE
 1,1,2,3,4,5,6,7,8
*REBAR, ELEMENT=CONTINUUM, MATERIAL=STEEL, GEOMETRY=SKEW,
 NAME=LAYER_A
 ONE, .04, 2.5, 55.28, , 2
 .2, 0., .4, .0
*MATERIAL, NAME=STEEL
*ELASTIC
 30.E6,
 …
```

The rebar layer is defined using isoparametric direction 2. The intersecting face is defined in *Figure 13* and has nodes 1-5-6-2. The position of the rebar layer is given by its intersection with the edges of this face; the fractional distances, f_1 and f_3, are shown in *Figure 17*. The orientation angle β of the rebar in physical space is 30°. Following the same procedure for calculating α as was described for isoparametric rebar, $L = 12.5$, $W = 5.$, and the orientation angle in the isoparametric-mapped cube α is 55.28°.

Defining Single Rebars in Three-Dimensional Continuum Elements

You can define single rebars in three-dimensional continuum elements; in this case the rebar is assumed to be placed along one of the element's isoparametric directions. The rebar is then located by its intersection with the intersecting face (defined in *Figure 13*). The intersections of constant isoparametric lines with edges 1 and 2 of the intersecting face are given by fractional distances along edges 1 and 2, measured from the first node of each edge, as shown in *Figure 12*.

You specify the elements that contain the rebars; the cross-sectional area, A, of each rebar; the fractional distances locating the rebar's position in the element, f_1 and f_2; and the isoparametric direction. Give the fractional distances with respect to edge 1 and edge 2 for the isoparametric direction chosen, as defined in *Figure 13*.

Input File Usage: Use the following option to define single rebars in three-dimensional continuum elements:

**REBAR*, ELEMENT=CONTINUUM, MATERIAL=*mat*, SINGLE

Defining the Rebar Material

The material properties of the rebars are distinct from those of the underlying element and are defined by a separate material definition (*Material Data Definition*). You must associate each rebar definition with a set of material properties.

The following material behavior cannot be used in Abaqus/Standard to define rebar materials:

- *Porous Metal Plasticity*.

The following material behaviors cannot be used in Abaqus/Explicit to define rebar materials:

- *Defining Fully Anisotropic Elasticity*;
- *Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*;
- *Equation of State*;
- *Hill Anisotropic Yield/Creep*;
- *Porous Metal Plasticity*;
- *Extended Drucker-Prager Models*;
- *Modified Drucker-Prager/Cap Model*;
- *Crushable Foam Plasticity Models*; or
- *Cracking Model for Concrete*.

Although Abaqus/Standard will allow for a rebar material to be defined with orthotropic elasticity (*Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*) or anisotropic elasticity (*Defining Fully Anisotropic Elasticity*), D_{1111} is the only meaningful material constant in these definitions. D_{1111} is used to compute the strain in the rebar direction, ε_{11}, using the corresponding stress component, σ_{11}, as discussed in *Linear Elastic Behavior*; no other strain or stress components exist in rebars.

In Abaqus/Standard density is ignored for the rebar material properties. Hence, the mass of the rebar is neglected in eigenvalue extraction and implicit dynamic procedures and for gravity, centrifugal, and rotary acceleration distributed loads.

Input File Usage: Use the following option to associate a material definition with a rebar definition:

**REBAR*, ELEMENT=*elem*, MATERIAL=*mat*

Initial Conditions

Initial conditions (*Initial Conditions*) can be used to define rebar prestress or solution-dependent values for rebars.

Defining Prestress in Rebar

For structures in which reinforcing is defined (such as reinforced concrete structures), you can use initial conditions to define the prestress in the rebars.

In such cases in Abaqus/Standard the structure must be brought to a state of equilibrium before it is actively loaded by means of an initial static analysis step (*Static Stress Analysis*) with no external loads applied (or, perhaps, with the "dead" loads only)—see *Defining Initial Stresses*.

Input File Usage: **INITIAL CONDITIONS*, TYPE=STRESS, REBAR
element number or element set name, rebar name, prestress value

Holding Prestress in Rebar in Abaqus/Standard

If prestress is defined in the rebars and unless the prestress is held fixed, it will be allowed to change during an equilibrating static analysis step; this is a result of the straining of the structure as the self-equilibrating stress state establishes itself. An example is the pretension type of concrete prestressing in which reinforcing tendons are initially stretched to a desired tension before being covered by concrete. After the concrete cures and bonds to the rebar, release of the initial rebar tension transfers load to the concrete, introducing compressive stresses in the concrete. The resulting deformation in the concrete reduces the stress in the rebar.

Alternatively, you can keep the initial stress defined in some or all of the rebars constant during this initial equilibrium solution. An example is the post-tension type of concrete prestressing; the rebars are allowed to slide through the concrete (normally they are in conduits), and the prestress loading is maintained by some external source (prestressing jacks). The magnitude of the prestress in the rebar is normally part of the design requirements and must not be reduced as the concrete compresses under the loading of the prestressing. Normally, the prestress is held constant only in the first step of an analysis. This is generally the more common assumption for prestressing.

If the prestress is not held constant in analysis steps following the step in which it is held constant, the stress in the rebar will change due to additional deformation in the concrete. If there is no additional deformation, the stress in the rebar will remain at the level set by the initial conditions. If the loading history is such that no plastic deformation is induced in the concrete or rebar in steps subsequent to the steps in which the prestress is held constant, the stress in the rebar will return to the level set by the initial conditions upon removal of the loading applied in those steps.

Input File Usage: **PRESTRESS HOLD*

Defining the Initial Values of Solution-Dependent State Variables for Rebars

You can define the initial values of solution-dependent state variables for rebars within elements. See *Initial Conditions* for details.

Input File Usage: **INITIAL CONDITIONS*, TYPE=SOLUTION, REBAR

Output

Rebar force output is available at the rebar integration locations with output variable RBFOR. The rebar force is equal to the rebar stress times the current rebar cross-sectional area. The current cross-sectional area of the rebar is calculated by assuming the rebar is made of an incompressible material, regardless of the actual material definition. For rebars in membrane or shell elements output variables RBANG and RBROT identify the current orientation of isoparametric or skew rebar within the element and the relative rotation of the rebar as a result of finite deformation, respectively. These quantities are measured with respect to the user-specified isoparametric direction in the element, not the default local element system or the orientation-defined system. See *Rebar modeling in shell, membrane, and surface elements*.

See *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers* for information on additional output quantities such as stress and strain. For rebars in membrane or shell elements with multiple integration points, output quantities are available at the integration points and at the centroid of the element.

Specifying the Direction for Rebar Angle Output in Shell and Membrane Elements

The output quantities RBANG and RBROT can be measured from either of the isoparametric directions in the plane of the shell or the membrane. You can specify the desired isoparametric direction from which the rebar angle will be measured (1 or 2). In axisymmetric shells and membranes the first isoparametric direction coincides with the meridional direction, and the second isoparametric direction coincides with the hoop direction. The rebar angle is measured from the isoparametric direction to the rebar with a positive angle defined as a counterclockwise rotation around the element's normal direction. The default direction is the first isoparametric direction.

Input File Usage: Use any of the following options:

**REBAR*, ELEMENT=SHELL, MATERIAL=*mat*, ISODIRECTION=*n*
**REBAR*, ELEMENT=AXISHELL, MATERIAL=*mat*, ISODIRECTION=*n*
**REBAR*, ELEMENT=MEMBRANE, MATERIAL=*mat*, ISODIRECTION=*n*
**REBAR*, ELEMENT=AXIMEMBRANE, MATERIAL=*mat*, ISODIRECTION=*n*

Example

As an example, a user-defined local coordinate system is used to define skewed rebar in a shell element (skew angle $\alpha = 30°$), and the output value of RBANG is 75°, as illustrated in *Figure 18*:

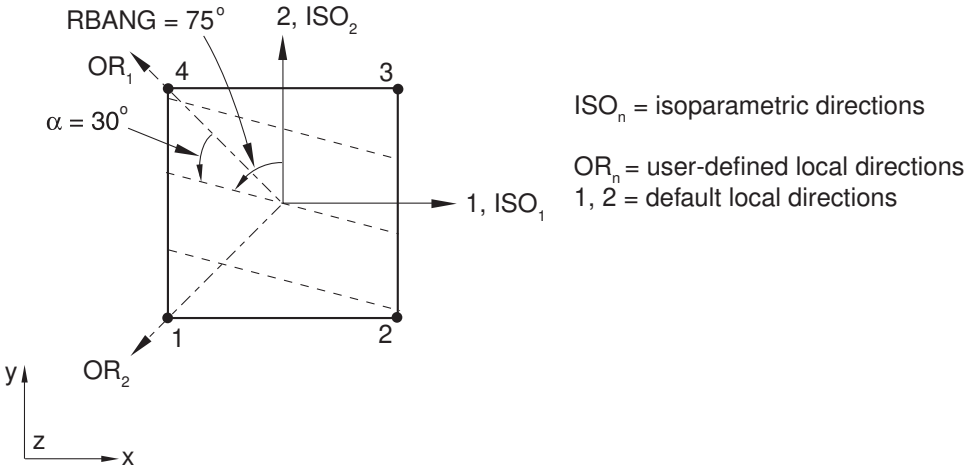


Figure 18: RBANG measurement for skew rebar defined relative to user-defined local coordinate directions.

```
*REBAR, ELEMENT=SHELL, MATERIAL=MAT1, NAME=REBARB,
 GEOMETRY=SKEW, ORIENTATION=ORIENT, ISODIRECTION=2
 ELSET1, 0.01, 0.1, 0.0, 30.
*ORIENTATION, SYSTEM=RECTANGULAR, NAME=ORIENT
 -0.7071, 0.7071, 0.0, -0.7071, -0.7071, 0.0
 3, 0.0
```

The rebars are located at the midsurface of the shell. Output variable RBANG is measured from the second isoparametric direction to the rebar. If the first isoparametric direction were chosen instead, output variable RBANG would report an angle of 165°.

Visualizing Rebar Orientation and Results in Rebar

Abaqus/CAE does not support visualization of element-based rebar or rebar results. Abaqus/CAE does support visualization of rebar defined as described in *Defining Reinforcement*.

Orientations

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Distribution Definition*
- *Connection Types*
- *About the Material Library*
- *Material Data Definition*
- *Fabric Material Behavior*
- *Distributed Loads*
- *Kinematic Coupling Constraints*
- *Coupling Constraints*
- *Inertia Relief*
- **ORIENTATION*
- *Creating datum coordinate systems*

Overview

A user-defined orientation is used to define a local coordinate system for:

- definition of material properties—for example, anisotropic materials or jointed materials (a local coordinate system must be defined if anisotropic material properties are defined for solid elements);
- definition of local material directions, such as the in-plane fill and warp yarn directions of a fabric material or the fiber directions of anisotropic hyperelastic materials;
- definition of fiber dispersion for multiscale materials or short-fiber reinforced linear composite materials in Abaqus/Explicit;
- definition of rebars in shell, membrane, and surface elements;
- definition of rotary inertia and connector elements;
- definition of coupling constraints;
- definition of loading directions for distributed general tractions, shear tractions, and general edge loads;
- definition of local tangent directions for contact in Abaqus/Standard;
- material calculations at integration points;
- output of components of stress, strain, and element section force; and
- definition of a local system of rigid body motion directions for inertia relief in Abaqus/Standard.

A user-defined orientation cannot be used:

- at points where the smeared crack concrete material behavior (see *Concrete Smeared Cracking*) is also used in Abaqus/Standard;
- to specify a local coordinate system for defining nodal coordinates—see *Specifying a Local Coordinate System in Which to Define Nodes* or *Specifying a Local Coordinate System for the Nodal Coordinates* instead; or
- to specify a local coordinate system for applying loads and boundary conditions—see *Transformed Coordinate Systems* instead.

Considerable generality is provided in the way the local system can be defined, since this system must often change from point to point because of the shape and construction of the structure being modeled. You can define the local orientation directly. The direct data methods provided in Abaqus are intended to give sufficient generality to model most cases easily: they are particularly useful for regular geometry. Distributions (see *Distribution Definition*) can be used to define spatially varying local coordinate systems for solid continuum, shell, and membrane (in Abaqus/Standard) elements directly for arbitrary geometries.

In Abaqus/Standard you can alternatively define the local orientation in user subroutine *ORIENT*.

Assigning a Name to an Orientation

You must assign a name to each orientation definition. This name is used by various features to refer to the orientation definition.

Input File Usage: **ORIENTATION*, NAME=*name*

Abaqus/CAE Usage: Any module: **Tools**->**Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Name:** *name*

Defining a Local Coordinate System in a Model That Contains an Assembly of Part Instances

In a model defined in terms of an assembly of part instances, you can define a local orientation at the part, part instance, or assembly level. An orientation defined at the part or part instance level is rotated according to the positioning data given for each instance of that part (or for the part instance). This includes the case when an orientation is defined using a distribution. See *Assembly Definition* and *Distribution Definition*.

Defining a Local Coordinate System Directly

A two-stage process is used to define the local system directly.

1. You define the local coordinate system at the particular location at which it is required. You can select a rectangular, cylindrical, or spherical coordinate system. The coordinate system is defined in terms of points *a*, *b*, and *c*, as shown in *Figure 1*. You can select the method for defining points *a*, *b*, and *c*, as described below.

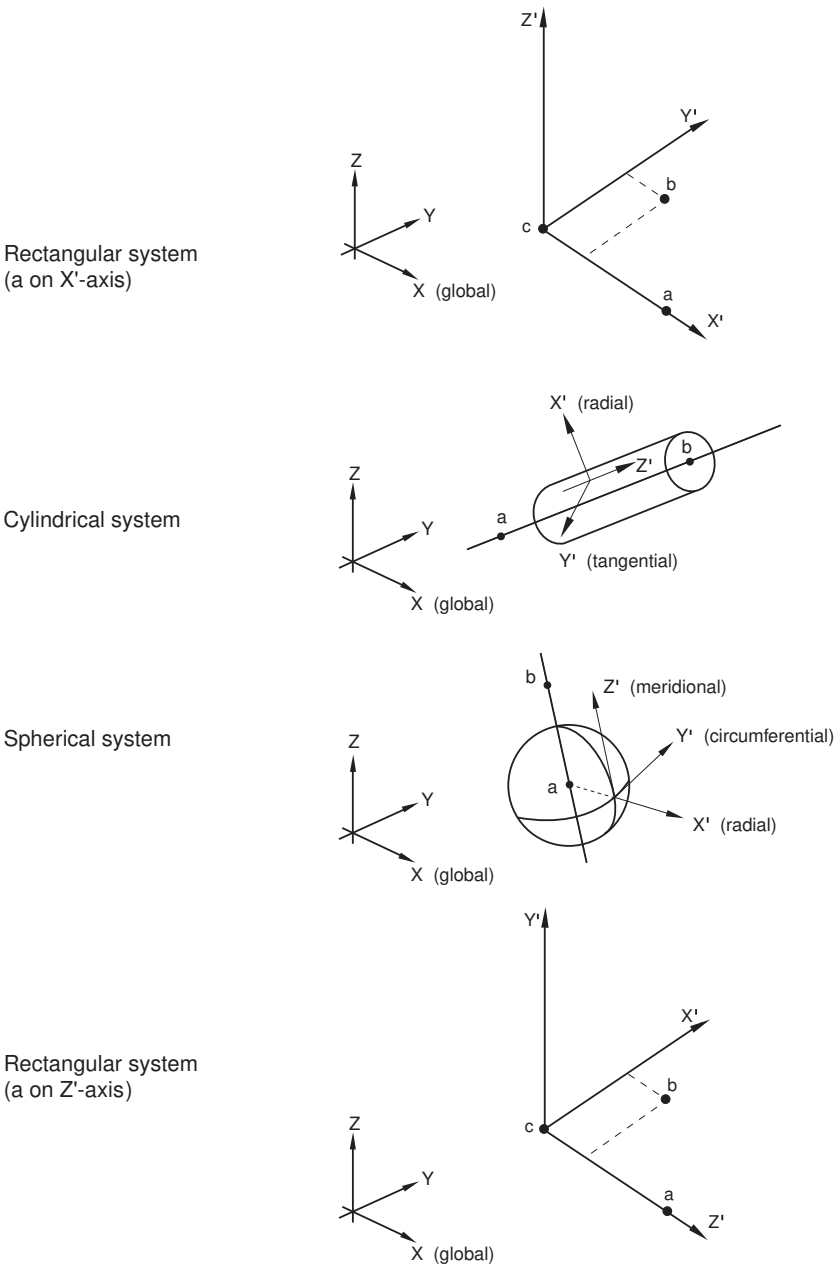


Figure 1: Orientation systems.

2. Optionally, you can specify an additional rotation by identifying one of these local directions ($\boldsymbol{X'}$, $\boldsymbol{Y'}$, or $\boldsymbol{Z'}$) as a rotation axis and giving a rotation, in degrees, about that axis. The local system is then rotated through this angle about the specified axis. This method of defining a local system is required for contact surfaces in Abaqus/Standard, shells, membranes, gasket elements, and when the orientation is associated with a composite solid section. The additional rotation is illustrated in *Figure 2*.

a. 1-direction specified.

b. 2-direction specified.

c. 3-direction specified.

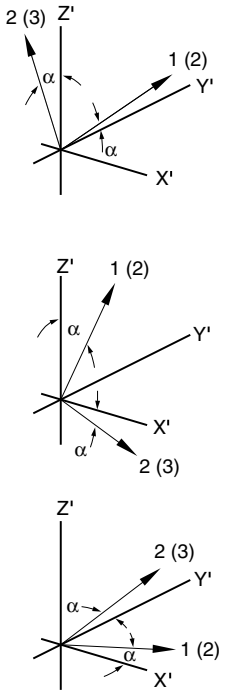


Figure 2: Specifying rotation about a local axis for shell elements, membrane elements, gasket elements (in parentheses), composite solids (in parentheses), and contact surfaces in Abaqus/Standard.

The local coordinate system for composite solids is indicated by X', Y', and Z'. The local coordinate system for other element types is indicated by 1, 2, and 3; the axis labels in parentheses are oriented for gasket elements.

Available Coordinate Systems

Rectangular, cylindrical, and spherical coordinate systems are available.

Defining a Rectangular Coordinate System

A rectangular Cartesian coordinate system is shown in *Figure 1*(a). The rectangular coordinate system is the default. Alternatively, you can define a rectangular Cartesian coordinate system as shown in *Figure 1*(d).

Input File Usage: **ORIENTATION*, NAME=*name*, SYSTEM=RECTANGULAR

**ORIENTATION*, NAME=*name*, SYSTEM=Z RECTANGULAR

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Rectangular**

Defining a Cylindrical Coordinate System

A cylindrical coordinate system is shown in *Figure 1*(b). The local axes are X'=radial, Y'=tangential, Z'=axial.

Input File Usage: **ORIENTATION*, NAME=*name*, SYSTEM=CYLINDRICAL

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Cylindrical**

Defining a Spherical Coordinate System

A spherical coordinate system is shown in *Figure 1*(c). The local axes are X'=radial, Y'=circumferential, Z'=meridional.

Input File Usage: **ORIENTATION*, NAME=*name*, SYSTEM=SPHERICAL

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: select any method, and click **OK**: **Spherical**

Methods for Defining a Coordinate System

You can define a coordinate system by specifying the locations of points *a*, *b*, and *c* directly; by specifying the locations of points *a*, *b*, and *c* relative to global node numbers; by specifying the locations of points *a*, *b*, and *c* relative to local node numbers; by specifying an offset from another coordinate system; or by specifying two lines in the coordinate system.

Defining a Coordinate System by Specifying the Locations of Points *a*, *b*, and *c* Directly

You can specify the coordinates of points *a*, *b*, and *c* directly. These coordinates should be appropriate to the system chosen. This method is the default.

You can define a rectangular Cartesian coordinate system (X', Y', Z') by specifying three points (*a*, *b*, and *c*) that lie on the X'-Y' plane, as shown in *Figure 1*(a). Point *c* is the origin of the system, point *a* must lie on the X'-axis, and point *b* must lie on the X'-Y' plane. Although not necessary, it is intuitive to select point *b* such that it is on or near the local Y'-axis.

Alternatively in Abaqus/Standard you can define a rectangular Cartesian coordinate system (X', Y', Z') by specifying three points (*a*, *b*, and *c*) that lie on the X'-Z' plane, as shown in *Figure 1*(d). Point *c* is the origin of the system, point *a* must lie on the Z'-axis, and point *b* must lie on the X'-Z' plane. Although not necessary, it is intuitive to select point *b* such that it is on or near the local X'-axis.

For rectangular coordinate systems the default location of the origin (point *c*) is the global origin.

You define a cylindrical coordinate system by giving the two points, *a* and *b*, on the polar axis of the cylindrical system, as shown in *Figure 1*(b).

You define a spherical coordinate system by giving the center of the sphere, *a*, and point *b* on the polar axis, as shown in *Figure 1*(c).

To define a spatially varying local coordinate system directly on solid continuum and shell elements, you can specify the coordinates of points *a* and *b* on an element-by-element basis using a distribution. Using a distribution to define the coordinates of the optional point *c* is not currently supported. See *Distribution Definition*.

Input File Usage: **ORIENTATION*, NAME=*name*, DEFINITION=COORDINATES

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**, **Method**: **3 points**

Defining a Coordinate System by Giving Global Node Numbers for Points *a*, *b*, and *c*

You can locate points *a*, *b*, and *c* at nodes by specifying three global node numbers. For a rectangular coordinate system the default location of the origin (point *c*) is the global origin.

Input File Usage: **ORIENTATION*, NAME=*name*, DEFINITION=NODES

Abaqus/CAE Usage: You cannot define a coordinate system by giving global node numbers in Abaqus/CAE.

Defining a Coordinate System by Giving Local Node Numbers for Points *a*, *b*, and *c*

You can locate points *a*, *b*, and *c* by specifying the local node numbers of an element. Local node numbers refer to the order in which nodes are specified in the element connectivity. For example, local node number 2 corresponds to the second node specified for the element definition. This definition method allows for variation of the local coordinate system on an element-by-element basis with a single orientation definition. For example, if local node number 2 is given as the location of point *c* and local node number 3 is given as the location of point *a*, the local $\boldsymbol{X'}$-direction is defined to be parallel to the (2, 3) side of the element. By default, the origin (point *c*) of the local coordinate system is the first node of the element (local node number 1).

Input File Usage: **ORIENTATION*, NAME=*name*, DEFINITION=OFFSET TO NODES

Abaqus/CAE Usage: You cannot define a coordinate system by giving local node numbers in Abaqus/CAE.

Defining a Coordinate System by Giving an Offset from Another Coordinate System

You can define a coordinate system by specifying an offset from an existing coordinate system.

Input File Usage: You cannot define a coordinate system by giving an offset from another coordinate system in the input file.

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: **Offset from CSYS**

Defining a Coordinate System by Giving Two Edges

You can define a coordinate system by specifying two edges. The first edge defines the *X*- or *R*-axis, and the *X*–*Y* or $\boldsymbol{R-\theta}$ plane passes through the second.

Input File Usage: You cannot define a coordinate system by giving two edges in the input file.

Abaqus/CAE Usage: Any module: **Tools->Datum**: **Type**: **CSYS**: **2 lines**

Defining Fiber Dispersion for Multiscale or Short-Fiber Reinforced Linear Composite Materials

You can model multiscale materials with fiber dispersion (see *Specifying a Second-Order Orientation Tensor in Abaqus/Explicit* and *Specifying a Second-Order Orientation Tensor in Abaqus/Standard*) or short-fiber reinforced linear composites (see *Defining the Elasticity of a Short-Fiber Reinforced Composite*).

In Abaqus/Explicit, you must specify the second-order tensor that characterizes the fiber dispersion as part of a local orientation system definition if the values vary spatially. In addition, when modeling short-fiber reinforced linear composites, you must specify a local orientation system that is aligned with the principal directions of the second-order tensor.

When specifying a spatially varying second-order tensor in Abaqus/Standard, you can either specify the distribution of the second-order tensor as part of a local orientation system definition or as part of a material definition (see *Specifying a Second-Order Orientation Tensor in Abaqus/Standard*).

Input File Usage: Use the following option to define an orthogonal system and components of the second-order tensor with respect to that system to specify the fiber dispersion of a multiscale material or a short-fiber reinforced linear composite material:

**ORIENTATION*, DISPERSION

Abaqus/CAE Usage: Specifying the fiber dispersion of a multiscale material or a short-fiber reinforced linear composite material is not supported in Abaqus/CAE.

Defining Local Material Directions for Anisotropic Hyperelastic Materials

When modeling anisotropic hyperelastic materials with an invariant-based formulation (see *Invariant-Based Formulation*) you must define the local directions that characterize each family of fibers. These directions need not be orthogonal in the initial configuration. You can specify these local directions with respect to an orthogonal orientation system at a material point. Up to three local directions can be specified as part of the definition of a local orientation system. The local directions can be output as field variables to the output database (see *Output*).

Input File Usage: Use the following option to define an orthogonal system and *N* local directions with respect to that system to identify the preferred directions of an anisotropic hyperelastic material:

**ORIENTATION*, LOCAL DIRECTIONS=*N*

Abaqus/CAE Usage: Local material directions cannot be defined in Abaqus/CAE.

Defining Yarn Directions in the Reference Configuration for a Fabric Material

In general, the yarn directions in a fabric material might not be orthogonal to each other in the reference configuration (see *Fabric Material Behavior*). You can specify these local directions with respect to the in-plane axes of an orthogonal orientation system at a material point. Both the local directions and the orthogonal system are defined together as a single orientation definition. If the local directions are not specified, these directions are assumed to match the in-plane axes of the orthogonal system defined. The local direction might not remain orthogonal with deformation. Abaqus updates the local directions with deformation and computes the nominal strains along these directions and the angle between them (the fabric shear strain). The constitutive behavior for the fabric defines the nominal stresses in the local system in terms of the fabric strain. The local directions can be output as field variables to the output database (see *Output*).

Input File Usage: Use the following option to define an orthogonal system and the local directions with respect to that system to identify the yarn directions in the reference configuration:

**ORIENTATION*, LOCAL DIRECTIONS=2

Abaqus/CAE Usage: Yarn directions for fabric materials cannot be defined in Abaqus/CAE.

Defining a Local Coordinate System in Abaqus/Standard Using a User Subroutine

In some cases the simplest way to specify a local system is by means of a user subroutine. User subroutine *ORIENT* is provided in Abaqus/Standard. In this case the user subroutine is called each time that an orientation definition is needed. In a model defined in terms of an assembly of part instances, the local directions defined by user subroutine *ORIENT* must be defined relative to the coordinate system of the assembly.

Input File Usage: **ORIENTATION*, NAME=*name*, SYSTEM=USER

Abaqus/CAE Usage: You can enter the name of an orientation defined in user subroutine *ORIENT* whenever a user-defined orientation is allowed.

Multiple References to an Orientation Definition

Because the orientation is independent of the material definition and they can both be referenced in any element property definition, the ability to describe complex structural components (such as laminated composite shells) is quite general and straightforward to use.

An orientation definition can be used as often as needed and with different material or element type definitions; for example, it can be used for different layers of a shell where the orientation is the same.

Large-Displacement Considerations

In large-displacement analysis a user-defined orientation rotates with the average rigid body motion of the material point, the rigid body when the orientation is used with ROTARYI elements, the first node of the joint in JOINTC elements, the pipeline edge for pipe-soil interaction elements, the appropriate surface for contact in Abaqus/Standard, or the reference node when the orientation is used with coupling constraints. However, when an orientation is defined for spring, dashpot, or gasket elements in Abaqus/Standard, the local directions always remain fixed in space.

Because the material directions rotate with the average rigid body motion at a material point, using anisotropic elasticity to model a material that is not truly a continuum can give significant errors if shear deformation is large. For example, an individual fiber in a reinforcing belt of a tire can shear relatively easily with respect to fibers in other directions. The fibers rotate with the actual deformation of the material point and not with the average rigid body motion. In this case the anisotropic behavior is better modeled with rebars or as a fabric material. The fabric material model in Abaqus/Explicit tracks the current yarn directions as local directions with respect to the orthogonal coordinate system.

Use with Two-Dimensional Solid Elements

When a user-defined orientation is used with two-dimensional solid elements such as plane stress, plane strain, or torsionless axisymmetric elements, the orientation must redefine only the *X*- and *Y*-directions: the third direction must remain unchanged (*Z*-direction for plane strain and plane stress elements, θ-direction for axisymmetric elements). When a user-defined orientation is used for material behavior with axisymmetric elements with twist, all three directions can be redefined. For axisymmetric elements, including the CGAX and CAXA families of elements, the global 1-, 2-, and 3-directions are the radial, axial, and hoop directions, respectively. Cylindrical or spherical orientations might be appropriate for axisymmetric elements only if the local Z'-direction is in the global 3-, or hoop, direction.

Use with Shell, Continuum Solid Shell, Membrane, or Gasket Elements or with Contact Surfaces

When a user-defined orientation is used with shell, continuum solid shell, membrane, or gasket elements or with contact surfaces, Abaqus first rotates and then projects the orientation system onto the element or contact surface using the algorithm described in this section.

Abaqus first rotates (through the additional rotation angle) the user-defined local coordinate system about the specified rotation axis. If you do not specify a rotation axis or an additional angle, Abaqus will by default use the local 1-axis and a rotation of 0°. After the rotation, Abaqus follows a cyclic permutation (1, 2, 3) of the axes and projects the axis following the axis for additional rotation onto the contact surface or onto the surface of the element to form the local material 1-direction (or the local material 2-direction for gaskets). The remaining material direction is then defined by the cross product of the element normal and the projected direction. Thus, for example:

1. If you choose the user-defined 1-axis as the axis for additional rotation, Abaqus projects the 2-axis onto the element or contact surface. This will be local direction 1 for contact surfaces, shells, and membranes and local direction 2 for gaskets.
2. Abaqus takes the positive element or contact surface normal as the local 3-direction for contact surfaces, shells, and membranes and the local 1-direction for gaskets.
3. Abaqus computes the local 2-direction (3-direction for gaskets) by taking the cross product of the element or contact surface normal and the local 1-direction (2-direction for gaskets), such that the three local axes form an orthonormal, right-handed local coordinate system.

When the axis for additional rotation points in a direction that is opposite to the element or contact surface normal, the local 2-direction (3-direction for gaskets) is reversed with respect to the corresponding user-defined axis; see *Figure 3*. This does not apply in the case of an orientation used to define rebars; see below.

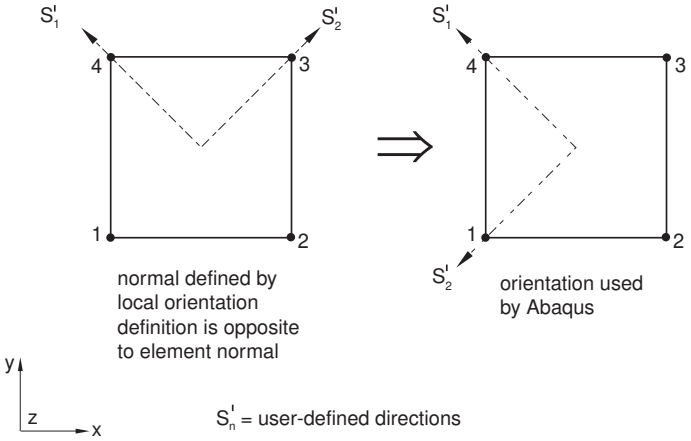


Figure 3: The local 3-direction (1-direction for gaskets) will be in the same direction as the element or contact surface normal.

As an example, the orientation of the spiral-wound layer of the cylindrical shell shown in *Figure 4* would be given by defining a cylindrical coordinate system and then specifying the rotation axis as the 1-axis and giving the rotation angle α (in degrees). The local 1- and 2-directions for material property specification and material calculations are then those indicated in the figure.

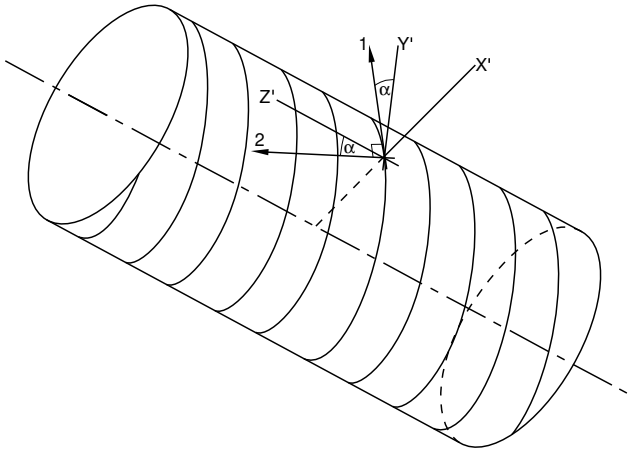


Figure 4: Spiral-wound cylindrical shell layer: material orientation example.

The projected directions are most easily understood when the axis for additional rotation is approximately perpendicular to the element or contact surface.

To define a spatially varying local coordinate system directly, you can specify the additional angle of rotation on an element-by-element basis using a distribution. See *Distribution Definition*.

Defining Rebars in Shell, Membrane, and Surface Elements

The orientation of skew rebars in shell, membrane, and surface elements can be defined relative to a user-defined orientation (see *Defining Reinforcement*). In this case the local coordinate system is calculated as follows:

1. The local 1-direction follows a cyclic permutation of the additional rotation direction; for example, if you choose the user-defined 1-axis as the axis for additional rotation, Abaqus projects the 2-axis onto the element. This will be the local 1-direction.
2. The axis for additional rotation is made orthogonal to the element to create the local 3-direction. This local 3-direction need not be in the same direction as the element normal; in fact it will be in the opposite direction when the dot product of the axis for additional rotation and the element normal is negative.
3. Abaqus computes the local 2-direction by taking the cross product of the local 3-direction and the local 1-direction, such that the three local axes form an orthonormal, right-handed local coordinate system.

Because the local 3-direction might be opposite to the element normal, the definition of rebars is independent of the element connectivity.

Special Considerations When Defining Orientations on Contact Surfaces in Abaqus/Standard

When a user-defined orientation is used to define the local tangent directions on a surface of a three-dimensional contact pair in Abaqus/Standard (see *Contact Formulations in Abaqus/Standard*), you cannot define points *a* and *b* by giving local node numbers (see *Figure 1*).

For geometrically nonlinear analysis the local tangent directions of a contact pair rotate with the surface on which the directions were defined initially. These rotated local tangent directions are further rotated to ensure that the normal vector, computed using the cross product of the rotated local tangent directions, corresponds to the normal vector on the main surface when the secondary node comes into contact.

Arbitrary local tangent directions can be defined for a “line”-type secondary surface defined on three-dimensional beam, truss, or pipe elements. When this surface comes into contact with the main surface during a large-displacement analysis, the local tangent directions are projected onto the main surface.

Use with Laminated Shells

There are two ways in which a user-defined orientation can be used in the section definition of a laminated shell. In each case the name referenced in the shell section definition is the name of the user-defined orientation.

The first is to associate the user-defined orientation with the entire composite shell section definition. Then each layer's orientation angle can be given relative to this section orientation (or the default shell coordinate directions if no section orientation is used). The angle is given as an additional rotation about the shell normal after the orientation directions are projected onto the shell surface. Section forces (available only from Abaqus/Standard) are given in the local system specified for the section.

The second is to specify the name of each layer's orientation separately; this method allows different orientation definitions to be referenced for the different layers. Section forces and strains are still reported in the local orientation defined for the entire section (or the default shell coordinate directions if no section orientation is used). The individual layer orientations are used for material calculations and for output of stress and strain.

See *Using a Shell Section Integrated during the Analysis to Define the Section Behavior* and *Using a General Shell Section to Define the Section Behavior* for more information.

Use with Laminated Three-Dimensional Solid and Continuum Solid Shell Elements

When a user-defined orientation is used with composite solid elements (available only in Abaqus/Standard), one of the local directions must be identified as the axis for additional rotation. There are two ways in which this orientation can be used with a composite solid section definition to specify the material orientation for individual layers. In each case the name referenced in the solid section definition is the name of the user-defined orientation.

The first is to associate the user-defined orientation with the entire composite solid section definition. Then each layer's orientation angle can be given relative to this section orientation. The angle is given as an additional rotation about the local direction defined as the axis for additional rotation. For continuum solid shell elements, the angle is given as an additional rotation about the element normal after the orientation directions are projected onto the element midsurface.

The second is to specify the name of each layer's orientation separately; this method allows different orientation definitions to be referenced for the different layers. (In this case any user-defined orientation associated with the entire solid section will be ignored.) For continuum solid shell elements, each layer's orientation is projected onto the element midsurface.

See *Defining the Elements Section Properties* for more information.

Use with Pipe-Soil Interaction Elements

An arbitrary user-defined orientation can be defined for pipe-soil interaction elements (available only in Abaqus/Standard). In a large-displacement analysis the local orientation system rotates with the rigid body motion of the underlying pipeline. In a small-displacement analysis the local system is defined by the initial geometry of the PSI element and remains fixed in space during the analysis.

Use with Beam, Frame, and Truss Elements

See *Beam Element Cross-Section Orientation* for information on defining local material directions for beams, frames, or trusses.

Use with the Fabric Material Model

The fill and the warp yarn directions in the fabric plane are allowed to rotate with respect to each other under shear deformations (see *Fabric Material Behavior*). The current yarn directions are tracked with respect to the orthogonal coordinate system that also rotates with the material.

Use with the Jointed Material Model

When a user-defined orientation is used to define a joint system orientation for the jointed material model available in Abaqus/Standard (see *Jointed Material Model*), only the local coordinate system need be defined. It is assumed that the first direction is the direction normal to the plane of the joint and the other directions are in the plane of the joint. An additional axis of rotation cannot be used.

Use with Rotary Inertia and Connector Elements

A user-defined orientation must be used to define the local directions for certain connection types used to define connector elements (see *Connection Types*).

A user-defined orientation can be used with SPRING1, SPRING2, DASHPOT1, DASHPOT2, JOINTC, JOINT2D, JOINT3D, and ROTARYI elements to provide a local system for defining the direction of action of such elements. Points *a*, *b*, and *c* (see *Figure 1*) cannot be defined by giving local node numbers when the orientation is used for these elements. If you do not specify an axis for additional rotation, the local 1-direction with no additional rotation will be chosen as the default.

Use with the Kinematic Coupling Constraint

User-defined orientations can be used in Abaqus/Standard to define the local coordinate systems in which constraint directions are specified for a kinematic coupling constraint (see *Kinematic Coupling Constraints*). In this case you cannot define points *a*, *b*, and *c* by giving local node numbers (see *Figure 1*).

Use with Surface-Based Coupling Constraints

User-defined orientations can be used to define the local coordinate systems in which surface-based coupling constraint directions are specified (see *Coupling Constraints*). In this case you cannot define points *a*, *b*, and *c* by giving local node numbers (see *Figure 1*).

Use with Inertia Relief

A user-defined orientation can be used in Abaqus/Standard to define a local system of directions along which the inertia relief loads are computed (see *Inertia Relief*). In this case you cannot define points *a*, *b*, and *c* by giving local node numbers (see *Figure 1*).

Use with Distributed General Traction, Shear Traction, and General Edge Loads

User-defined orientations can be used in Abaqus to define the local coordinate systems in which the loading directions for distributed general tractions, shear tractions, and general edge loads are specified. See *Distributed Loads*.

Orientations Defined with Distributions

Spatially varying local coordinate systems (for material definitions, material calculations, and output) defined with a distribution can be applied only to solid continuum, membrane (in Abaqus/Standard), and shell elements. See *Solid (Continuum) Elements*, *Membrane Elements*, *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*, and *Using a General Shell Section to Define the Section Behavior*.

Output

When a user-defined orientation is used in an element section definition, the stress, the strain, and the element section force components are output in the local system.

For a fabric material the output of the regular material point tensors such as stress and strain are given in an orthogonal coordinate system even when the local yarn directions are non-orthogonal. However, the nominal fabric stress SFABRIC and the nominal fabric strain EFABRIC are also available for output (see *Fabric Material Behavior*).

This use of a local system is indicated by a footnote in the printed output tables from Abaqus/Standard. An orientation used with the jointed material model does not affect the output.

When a user-defined orientation is used in Abaqus/Standard with kinematic or distributing coupling constraints, the local system is indicated in the analysis input file processor output tables.

Local coordinate systems are written automatically to the output database with the exception of systems defined by specifying points *a* and *b* relative to local or global node numbers or systems defined through a user subroutine. Any additional rotations specified are ignored.

Material directions are written automatically to the output database. They can also be written to the Abaqus/Standard results file (with at least one output variable specified; see *Output of Local Directions to the Results File*). The material directions can be visualized in Abaqus/CAE by selecting **Plot**->**Material Orientations** in the Visualization module.

Surface Definition

In this section:

- *About Surfaces*
- *Element-Based Surface Definition*
- *Node-Based Surface Definition*
- *Analytical Rigid Surface Definition*
- *Eulerian Surface Definition*
- *Operating on Surfaces*

About Surfaces

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Element-Based Surface Definition*
- *Node-Based Surface Definition*
- *Analytical Rigid Surface Definition*
- *Eulerian Surface Definition*
- *Operating on Surfaces*
- *Integrated Output Section Definition*
- *Coupled Acoustic-Structural Analysis*
- *Distributed Loads*
- *Prescribed Assembly Loads*
- *Mesh Tie Constraints*
- *Coupling Constraints*
- *Shell-to-Solid Coupling*
- *About Contact Interactions*
- *Defining Tied Contact in Abaqus/Standard*
- *Cavity Radiation in Abaqus/Standard*

Overview

In Abaqus surfaces:

- can be used to define contact and interactions, including acoustic-structural interactions;
- can define regions used to prescribe distributed surface loads;
- can be used to tie dissimilar meshes together;
- can define cavities used for a cavity radiation analysis in Abaqus/Standard;
- can define pre-tensioned sections used in prescribing assembly loads in Abaqus/Standard;
- can define sections used for tracking the average motion of a surface in Abaqus/Explicit;
- can define sections for output quantities such as the total force transmitted through a surface;
- are geometric entities that have an area associated with them but have zero volume;
- have an identifiable orientation defined by their normals;
- are defined by specifying nodes or node sets, an analytic curve or surface, an Eulerian material instance, or element faces, edges, or ends; and
- can be deformable, rigid, or partially deformable and partially rigid.

This section describes the general rules that apply when creating surfaces in Abaqus.

Why Use Surfaces?

Surfaces can be used to model the interaction of two or more distinct bodies in a mechanical, acoustic, coupled acoustic-structural, coupled thermomechanical, coupled thermal-electrical-structural, thermal, coupled

thermal-electrical, or cavity radiation analysis. A rigid surface can be used to represent a body that is much stiffer than the rest of the model in a mechanical or coupled thermomechanical analysis, with the limitation that no heat can be transferred to the rigid body. In acoustic-structural analysis, surfaces can be used to define impedance boundary conditions, including first-order conditions for modeling acoustic radiation.

Surfaces can be used to define a region on which a distributed surface load is prescribed; this can facilitate user input of distributed surface loads for complex models. In addition, surfaces can be used to define multi-point or coupling constraints. Surfaces can also define pre-tension sections used in prescribing assembly loads in Abaqus/Standard.

Finally, surfaces can be used to define sections to obtain output of accumulated quantities; this provides a "free body diagram" output, allowing analyses of "force-flow" through a statically indeterminate structure.

The following types of surfaces can be defined in Abaqus:

- Element-based surfaces are defined on the faces, edges, or ends of elements. The elements can be deformable or rigid, leading to a surface that is deformable or rigid. When some of the deformable elements underlying a surface are part of a rigid body, the surface will become partially deformable and partially rigid.

  In Abaqus/Explicit a default element-based surface that includes all bodies in the model is provided for use with the general contact algorithm.
- Node-based surfaces are defined on nodes and, hence, are by definition discontinuous. A user-defined area can be associated with each node on the surface.
- Analytical surfaces are defined directly in geometric terms and are always rigid.
- Eulerian material surfaces are defined on material instances in an Eulerian section. These surfaces are available in Abaqus/Explicit for use with the general contact algorithm.

Element-based surfaces contain more intrinsic information than either node-based surfaces or analytical rigid surfaces. When an element-based surface is used in a mechanical contact analysis, Abaqus can associate a surface area with each node and can calculate the contact stress acting on the surface. In contrast, Abaqus may not be able to calculate accurate contact stresses when a node-based surface (*Node-Based Surface Definition*) is used because the actual area associated with each node may not be correct. In addition, when a surface formed by shell, membrane, or rigid elements is used, Abaqus can consider the thickness and possibly the offset of the reference surface of these elements in some applications that refer to surfaces. For example, these thicknesses are accounted for by all contact algorithms available in Abaqus/Explicit and by the surface-to-surface, small-sliding contact formulation in Abaqus/Standard.

Contact between two node-based surfaces or a node-based surface with itself is not allowed; contact between two analytical rigid surfaces is not allowed. Contact between two rigid surfaces defined using rigid elements is not allowed in Abaqus/Standard and is allowed only with penalty contact in Abaqus/Explicit.

Surface definitions cannot change from step to step; however, new surfaces can be defined upon restart.

Internal Surfaces Created by Abaqus/CAE

In Abaqus/CAE many modeling operations are performed by picking geometry with the mouse. For example, a contact pair can be defined by picking faces on geometric part instances. Each such face must be translated into a surface in the input file. Such a surface is assigned a name by Abaqus/CAE and is marked as internal. These internal surfaces can be viewed using display groups in the Visualization module of Abaqus/CAE (see *Using display groups to display subsets of your model*).

Input File Usage: **SURFACE*, NAME=*surface_name*, INTERNAL

Restrictions on Surfaces

Refer to the subsequent sections on the different surface types available in Abaqus for details on the general restrictions that apply to all surface definitions of a given type. In addition, some features in Abaqus that use surfaces impose other restrictions on surface characteristics. These limitations are discussed in the following sections:

- *Integrated Output Section Definition*
- *Distributed Loads*
- *Mesh Tie Constraints*
- *Coupling Constraints*
- *Shell-to-Solid Coupling*
- *About Contact Interactions*
- *About Contact Pairs in Abaqus/Standard*
- *About General Contact in Abaqus/Explicit*
- *About Contact Pairs in Abaqus/Explicit*

In models that are defined in terms of an assembly of part instances, all surfaces must belong to a part, part instance, or the assembly. All of the general restrictions on surfaces still apply in such models. Additional rules are given in *Assembly Definition*.

Element-Based Surface Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- *Integrated Output Section Definition*
- *Distributed Loads*
- *Prescribed Assembly Loads*
- *Mesh Tie Constraints*
- *Coupling Constraints*
- *Shell-to-Solid Coupling*
- *About Contact Interactions*
- *Cavity Radiation in Abaqus/Standard*
- **SURFACE*
- *What is a surface?*

Overview

An element-based surface:

- can be defined on solid, structural, rigid, surface, gasket, or acoustic elements;
- can be deformable or rigid;
- can be defined on any combination of elements in many cases;
- can be defined on the exterior of any body; and
- can be defined on the interior of any body that is modeled with continuum, shell, membrane, surface, beam, pipe, truss, or rigid elements (for example, to define a cross-section through a body) either by simply cutting the body with a plane or by identifying the elements and the corresponding interior facets.

For details about defining node-based surfaces, see *Node-Based Surface Definition*. For details about defining analytical rigid surfaces, see *Analytical Rigid Surface Definition*. For details about defining surfaces using Boolean combinations of existing surfaces, see *Operating on Surfaces*.

Defining Element-Based Surfaces

You must assign a name to all element-based surfaces; this name can be used with various features to define a contact model, a surface-based load, or a surface-based constraint. In addition, you must specify the region of your model on which the surface is defined. In an input file you can define element-based surfaces on element faces, edges, or ends. In Abaqus/CAE you can define element-based surfaces on geometric or element faces, edges, or ends. The methods for defining surfaces depend on the underlying element type and are discussed later in this section.

In an input file you need only specify an element number or element set name and all exposed element faces of these elements (or “contact edges” of beam, pipe, and truss elements) will be included in the surface. Optionally(and the only available method in Abaqus/CAE), you can specify individual faces, edges, or ends, which allows you direct control over which faces, edges, or ends are to be included in the surface.

For general contact in Abaqus/Explicit the surface perimeter edges are generated automatically from the surface facets for use in edge-to-edge contact constraints; you can specify that geometric feature edges should be included as well (see *About General Contact in Abaqus/Explicit* and *Assigning Surface Properties for General Contact in Abaqus/Explicit* for more information).

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT (default)

An element number or element set name is specified as the first entry of each data line. Optionally, an element face, edge, or end identifier can be specified as the second entry on a data line. The face and edge identifiers used in Abaqus are discussed later in this section.

Multiple data lines can be used to define a surface. For example, `SURF_1` can be specified by the following input:

```
*SURFACE, NAME=SURF_1, TYPE=ELEMENT
ELSET_1,
ELSET_2, S2
```

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools**->**Surface**->**Create**: **Name:** *surface_name*

General Restrictions on Element-Based Surfaces

Elements defining a single surface must satisfy the following rules, regardless of how the surface is used in Abaqus:

- Two-dimensional, axisymmetric, and three-dimensional elements cannot be mixed in the same surface definition.
- In Abaqus/Standard deformable elements cannot be combined with rigid elements to define a single surface, but can be combined with other deformable elements that are part of a rigid body (see *Rigid Body Definition*).
- The following element types cannot be mixed with other element types in the same surface definition:
  - Coupled thermal-electrical-structural elements
  - Coupled temperature-displacement elements
  - Heat transfer elements
  - Pore pressure elements
  - Coupled thermal-electrical elements
  - Acoustic finite or infinite elements
- The axisymmetric solid Fourier elements with nonlinear, asymmetric deformation (CAXA elements) cannot form element-based surfaces.
- The face identifier label is required to import an element-based surface from an input file.

Surface Discretization

For element-based surfaces Abaqus uses a faceted geometry defined by the finite element mesh as the surface definition. The surface in a coarse finite element model might not be a very good approximation for contact modeling if the physical surface is curved. Therefore, sufficient mesh refinement must be used to ensure that the faceted surface is a reasonable approximation of the curved physical surface. Alternatively, some curved

surface geometries might be more effectively modeled with analytical rigid surfaces (see *Analytical Rigid Surface Definition*).

Creating Surfaces on Solid, Continuum Shell, and Cohesive Elements

There are three ways to define the facets of an element-based surface on solid, continuum shell, and cohesive elements:

1. by instructing Abaqus to generate the "free surface" from the exposed faces of elements,
2. by specifying the particular faces for each element, and
3. in Abaqus/Explicit by instructing Abaqus to generate an interior surface from element faces that are not exposed (that is, not part of the "free surface" of the model).

The automatic free surface generation approach is the simplest method of defining exterior surfaces on solid elements. Specifying the element faces gives you exact control over which element faces (any combination of exterior and interior faces) form the surface. Automatic generation of an interior surface is the simplest method of defining interior surfaces on solid elements (interior surfaces can be useful for modeling surface erosion due to element failure).

It is possible to use all three approaches in the same surface definition when creating a single surface.

Generating the Free Surface Automatically

You can define the facets of a surface by specifying a series of elements. The faces of these elements that are on the exterior (free) surface of the model are included in the surface definition.

When the free surface generation method is used to define surfaces, the specified elements can be a mixture of continuum and structural elements.

Multi-point constraints (*General Multi-Point Constraints*) involving nodes on exposed surfaces are not taken into account during free surface generation, which can result in faces that are not on the exterior of a body being included in surface definitions. For example, the nodes of the elements in element set `REFINED` shown in *Figure 1* are used in linear, mesh-refinement constraints. The surfaces generated with and without multi-point constraints are shown in *Figure 1*.

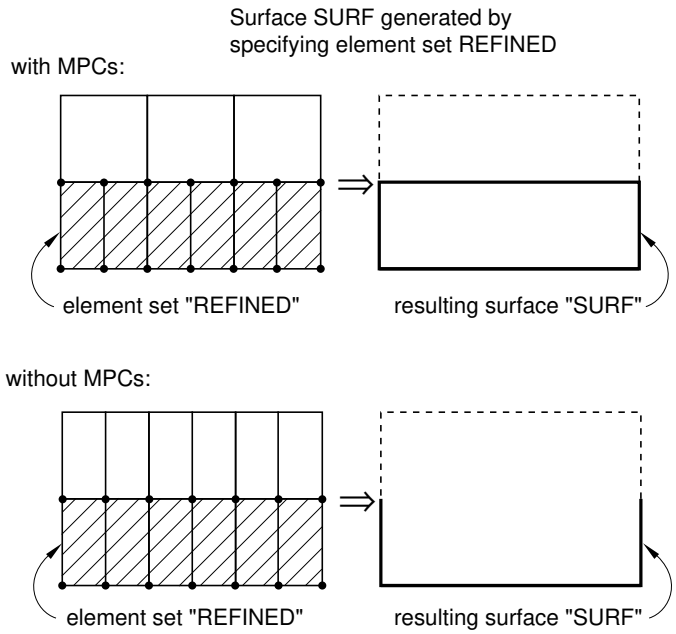
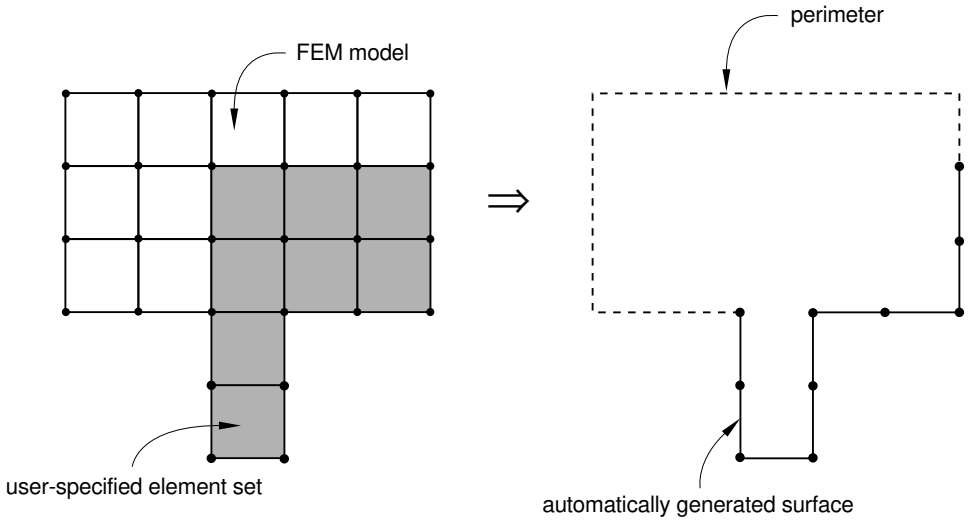


Figure 1: Effect of multi-point constraints on automatic surface generation.

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set,

For example, if the name of the shaded element set in the figure below is `ESETA`, the surface named `ASURF` is specified by

**SURFACE*, NAME=ASURF, TYPE=ELEMENT
ESETA,



Abaqus/CAE Usage: The automatic free surface generation method is not supported in Abaqus/CAE.

Special Treatment of Cohesive Elements for Automatic Free Surface Generation

The definition of exposed faces of elements for the purpose of automatic free surface generation has the following unique aspects regarding cohesive elements:

- Faces of noncohesive elements along an interface of shared nodes with cohesive elements are considered exposed.
- The top and bottom faces of all cohesive elements are considered exposed; side faces of cohesive elements are never considered exposed.

See *Modeling with Cohesive Elements* for examples of surfaces on or near cohesive elements.

Creating Surface Facets by Specifying Solid, Continuum Shell, and Cohesive Element Faces

You can define the facets of a surface by identifying the element faces that should be included in the surface definition.

Element face numbers are defined in *About the Element Library*. *Table 1* contains a list of valid face identifiers for all solid, continuum shell, and cohesive elements. The face identifier can refer to individual elements or to entire element sets.

Table 1: Surface definition face identifier labels for solid, continuum shell, and cohesive elements.

Elements		Face Labels
DCCAX2(D)		SPOS, SNEG
CPEG3(H)(T) CPS3(T) CPE3(H)(T) CAX3(H)(T) CGAX3(H) AC2D3 ACAX3 DC2D3(E) DCAX3(E)	CPEG6(M)(H)(T) CPS6M(T) CPE6(M)(H)(T) CAX6(M)(H)(T) CGAX6(M)(H)(T) AC2D6 ACAX6 DC2D6(E) DCAX6(E)	S1, S2, S3
CGAX4(R)(H)(T) CPEG4(H)(I)(R)(T) CPS4(I)(R)(T) CPE4(H)(I)(R)(T)(P) CAX4(H)(I)(R)(T)(P) C3D4(H)(T) AC2D4(R) ACAX4(R) AC3D4 DC2D4(E) DCAX4(E) DC3D4(E) DCC2D4(D) COH2D4	CGAX8(R)(H) CPEG8(R)(H)(T) CPS8(R)(T) CPE8(H)(R)(T)(P) CAX8(R)(H)(T)(P) C3D10(M)(H)(I)(T) AC2D8 ACAX8 AC3D10 DC2D8(E) DCAX8(E) DC3D10(E) DCCAX4(D) COHAX4	S1, S2, S3, S4
C3D6(H)(T) AC3D5 AC3D6 C3D5(H) CCL9(H) DC3D5 DC3D6(E)	SC6R C3D15(H)(V) AC3D15 CCL18(H) DC3D15(E) COH3D6	S1, S2, S3, S4, S5

Elements		Face Labels
C3D8(H)(I)(R)(T)(P) C3D27(R)(H) AC3D8(R) CCL12(H) DC3D8(R)(E) DCC3D8(D) SC8R	C3D20(H)(R)(T)(P) AC3D20 CCL24(R)(H) DC3D20(E) COH3D8	S1, S2, S3, S4, S5, S6

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or set, face identifier

When you specify the element faces to define surfaces, the specified elements cannot be a mixture of continuum and structural elements; however, each data line of the surface definition can refer to different element types.

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools**->**Surface**->**Create**: **Name:** *surface_name*, pick faces in viewport

Generating an Interior Surface Automatically

Abaqus/Explicit provides two approaches to define eroding surfaces for a solid element mesh for use in general contact (see *Modeling Surface Erosion*). The recommended approach dynamically evolves the list of surface faces to correspond to currently exposed faces of elements that have not failed. The other approach statically creates all the possible interior faces and tracks which of these faces are active. These methods give approximately the same results, but the dynamically evolving approach often uses much less memory and tends to be faster.

Elements that do not have any interior faces by definition (such as shell elements, beam elements, pipe elements, and membrane elements) are ignored.

Multi-point constraints are not taken into account when generating interior surfaces. This can result in faces that are on the interior of a body being excluded from the surface definition.

Generating a Dynamically Evolving Eroding Surface

In this recommended approach the surface evolves to correspond to the currently exposed faces of the specified element set. At a given point in the simulation, this surface might be a combination of originally exposed faces and faces that were originally in the interior.

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, ERODING

Abaqus/CAE Usage: Generating a dynamically evolving eroding surface is not supported in Abaqus/CAE.

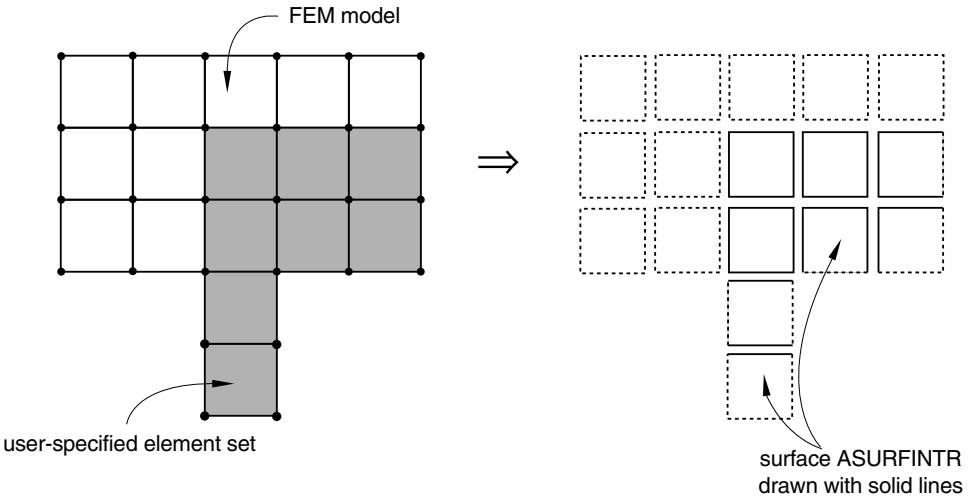
Generating a Static Interior Surface

In this approach all faces of the specified elements that are not on the exterior (free) surface of the model are included in the surface definition. Abaqus tracks which of these faces are currently exposed. The automatic generation of an interior surface is equivalent to constructing a surface consisting of all faces of the elements and then subtracting the free surfaces of those elements. A static interior surface is less convenient (because faces on the original exterior must be included separately) and less efficient (due to memory allocation for all faces of all elements rather than just currently active faces) to use than a dynamically evolving eroding surface.

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, INTERIOR

For example, if the name of the shaded element set in the figure below is `ESETA`, the surface named `ASURFINTR` (the elements in the figure have been reduced in size to differentiate faces that share the same nodes) is specified by

**SURFACE*, NAME=ASURFINTR, TYPE=ELEMENT
ESETA, INTERIOR



Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools**->**Surface**->**Create**: **Name:** *surface_name*, **Type**: **Mesh**; pick element faces or edges from an interior surface

You can use the selection tools to select from an interior entity of a model; see *Selecting interior surfaces*.

Creating Surfaces on Structural, Surface, and Rigid Elements

There are five ways to define surfaces on structural, surface, and rigid elements:

1. You can create a single-sided surface with a well-defined orientation by indicating either the top or bottom surface of each specified element.
2. You can create a double-sided surface by specifying only the elements and letting Abaqus generate the “free surface” from the exposed faces.
3. You can create an edge-based surface.
4. You can create a cross-section surface on the ends of beam, pipe, and truss elements.
5. You can create a three-dimensional curve-type surface along the length of beam, pipe, and truss elements by specifying only the elements and letting Abaqus generate the “free surface.”

It is possible to use any or all the above approaches in the same surface definition as long as it makes sense in the use of that surface with other features in Abaqus. *Table 2* contains a list of valid face and edge identifiers for structural, surface, and rigid elements.

Table 2: Surface definition face and edge identifier labels for structural, surface, and rigid elements.

Elements		Face and Edge Labels
SAX1 MAX1 MGAX1 M3D6 M3D9(R) MCL9 DS8 DSAX2 SFMAX2 SFMGAX2 SFM3D4(R) SFM3D8(R) SFMCL6	SAX2(T) MAX2 MGAX2 M3D8(R) MCL6 DS4 DSAX1 SFMAX1 SFMGAX1 SFM3D3 SFM3D6 SFMCL9 RAX2	SPOS, SNEG
B21(H) B23(H) PIPE21(H) T2D2(H)(T)	B22(H) (Abaqus/Standard) PIPE22(H) T2D3(H)(T)	SPOS, SNEG
B22 (Abaqus/Explicit) B32(H)(OS) ELBOW31(B)(C) PIPE31(H) T3D2(H)(T)	B31(H)(OS) B33(H) ELBOW32 PIPE32(H) T3D3(H)(T)	END1, END2; must use node-based surfaces with the contact pair algorithm in Abaqus/Explicit.
STRI3 S3(R)(S) M3D3	STRI65 R3D3	SPOS, SNEG, E1, E2, E3
ACIN2D2 ACINAX2	ACIN2D3 ACINAX3	SPOS E1, E2
S4(R)(S)(W)(5) S9R5 M3D4(R)	S8R5(T) R3D4	SPOS, SNEG, E1, E2, E3, E4
ACIN3D3	ACIN3D6	SPOS E1, E2, E3
ACIN3D4	ACIN3D8	SPOS E1, E2, E3, E4

Defining Single-Sided Surfaces

You can define a single-sided surface on the positive or negative face of structural, surface, or rigid elements. The positive face is defined as the one in the direction of the positive element normal, and the negative face is defined as the one in the direction opposite to the element normal. The definition of the element normal for all elements is given in *About the Element Library*.

You must ensure that all the specified elements have their normals oriented consistently. If they are oriented as shown in *Figure 2*, the surface normals will reverse direction as the surface is traversed and improper results might occur when the surface is used with features requiring an orientation such as distributed surface loads.

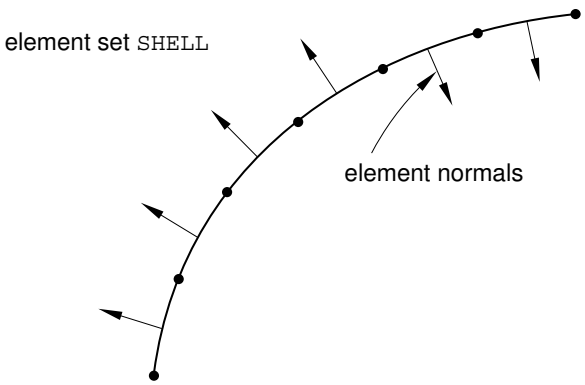


Figure 2: Inconsistent orientation of structural element normals can result in an invalid surface.

Further, an error message will be issued and the analysis will terminate if this condition is detected for surfaces used with mesh tie constraints in Abaqus/Standard or with contact pairs. To correct the surface orientations in this figure, two separate element sets with different face identifiers should be used.

Input File Usage: Use the following option to define a surface on the positive face of a structural, surface, or rigid element:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, SPOS

Use the following option to define a surface on the negative face of a structural, surface, or rigid element:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, SNEG

For example, single-sided surfaces on the positive faces of the elements in element set SHELL can be defined using input similar to

**SURFACE*, NAME=BSURF, TYPE=ELEMENT
SHELL, SPOS

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools**->**Surface**->**Create**: **Name:** *surface_name*, pick face in viewport, click mouse button 2, and specify the side of the selected face

Defining Double-Sided Surfaces

You can create double-sided surface facets on three-dimensional shell, membrane, surface, and rigid elements using the automatic surface facet generation approach (that is, specifying only the element numbers or sets). Some applications that refer to surfaces do not allow the use of double-sided surfaces: examples include contact pairs in Abaqus/Standard and features requiring an oriented surface such as distributed surface loads. When double-sided surfaces can be used, they are often preferred to single-sided surfaces. In some applications, such as when defining the contact domain for general contact, it does not matter whether single- or double-sided surfaces are used.

When double-sided surfaces are used with contact pairs in Abaqus/Explicit, the normals of all the underlying elements do not need to have a consistent positive orientation: Abaqus/Explicit will define the contact surface such that its facets have consistent normals, even if the underlying elements do not have consistent normals. The facet normals will be the same as the element normals if the element normals are all consistent; otherwise, an

arbitrary positive orientation is chosen for the surface. The positive orientation is significant only with respect to the sign of the contact pressure output variable for the contact pair algorithm, CPRESS (see *Output*).

Although contact is enforced unconditionally on both sides of a surface when self-contact is used with contact pairs, contact is enforced on both sides of a surface used in two-body contact only when that surface is double-sided (if allowed). The use of single-sided surfaces with contact pairs is sometimes desirable: the resolution of large initial overclosures in contact pairs is more robust with single-sided surfaces than with double-sided surfaces (see *Contact Initialization for Contact Pairs in Abaqus/Explicit*). However, single-sided contact is generally more limiting than double-sided contact; it might cause an analysis to fail due to excessive element distortion or not enforce the contact conditions realistically if a secondary node unexpectedly moves behind a main surface. This condition can occur, for example, when large deformations or rigid-body motions are present or due to complex tool shapes in a forming analysis.

Input File Usage: Use the following option to define a double-sided surface on three-dimensional shell, membrane, surface, or rigid elements in Abaqus/Explicit:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set,

For example, double-sided surfaces on the elements in element set `SHELL` can be defined using input similar to

**SURFACE*, NAME=BSURF, TYPE=ELEMENT
SHELL,

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: **Name:** *surface_name*, pick face in viewport, click mouse button 2, and choose **Both sides**

Defining Edge-Based Surfaces

You can define an edge-based surface on three-dimensional shell, membrane, surface, or rigid elements by specifying the individual edges. Alternatively, you can specify that perimeter edges are used to form the surface. It is possible to use both methods in the same surface definition when creating a single surface.

Input File Usage: Use the following option to specify the individual edges that form the surface:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, edge identifier

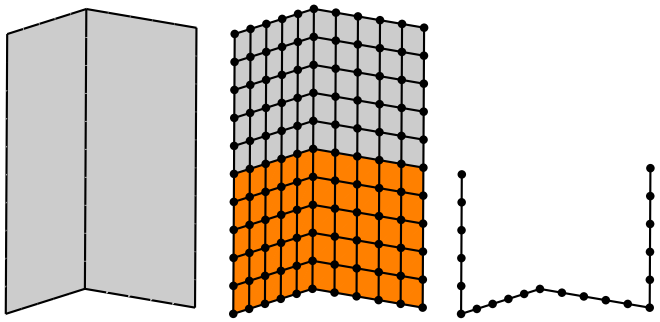
The individual edge identifiers used in Abaqus are listed in *Table 2*.

Use the following option to specify that all the edges of the elements that are on the exterior (free) surface of the model are used to form the surface:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, EDGE

For example, consider a shell with a fold on the left side of the figure below. If the shell mesh is as shown in the middle of this figure and element set shaded orange is named `ESETA`, the edge-based surface named `ESURF` comprised of perimeter edges shown on the right could be specified by the following input:

**SURFACE*, NAME=ESURF, TYPE=ELEMENT
ESETA, EDGE



Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: **Name:** *surface_name*, pick edges in viewport

In Abaqus/CAE you can specify that all the edges of the elements that are on the exterior (free) surface of the model are used to form the surface by directly picking all the free edges in the viewport.

Defining a Surface over the Cross-Section at the Ends of Beam, Pipe, and Truss Elements

To define a surface over the cross-section of beam, pipe, or truss elements, you must specify the end on which the surface is defined. Surfaces created on the ends of these elements can be used only for integrated output request (see *Integrated Output*) and integrated output section (see *Integrated Output Section Definition*) definitions.

Input File Usage: Use the following option to define a surface over the cross-section of a beam, pipe, or truss element:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, END1 or END2

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: **Name:** *surface_name*, pick three-dimensional wire region in viewport, click mouse button 2, and choose **End (Magenta)** or **End (Yellow)**

Defining a Surface along the Length of Three-Dimensional Beam, Pipe, and Truss Elements

You cannot specify the faces to define a surface along the length of three-dimensional beams, pipes, or trusses because their element connectivity cannot define a unique element or surface normal. Instead, you must specify that Abaqus should generate a surface for these elements. Therefore, the use of surfaces along the length of these elements is restricted.

In Abaqus/Standard element-based surfaces created along the length of three-dimensional beam, pipe, or truss elements can be used with the general contact algorithm or tie constraints. In a contact pair simulation, they can be used only as secondary surfaces. There are several advantages to using an element-based surface rather than a node-based surface when modeling contact in Abaqus/Standard with three-dimensional beams, pipes, or trusses:

1. The default local tangent directions are parallel and orthogonal to the element axis.
2. Abaqus/Standard calculates the contact results as contact forces per unit length rather than just contact forces.
3. It can be easier to define an element-based surface than a node-based surface.

In Abaqus/Standard a surface definition is not allowed for cases where three or more three-dimensional beams, pipes, or trusses are joined at a common node because of the lack of uniquely defined element tangents.

In Abaqus/Explicit element-based surfaces created along the length of three-dimensional beam, pipe, or truss elements can be used only with the general contact algorithm or tie constraints. To define contact for these elements using the contact pair algorithm, the nodes forming the beam, pipe, or truss elements can be included in a node-based surface definition (*Node-Based Surface Definition*) and a contact pair can be defined for this node-based surface and a non-node-based surface.

Surfaces along the length of three-dimensional beam, pipe, or truss elements cannot be used to prescribe a distributed surface load since the loading direction is not unique.

Input File Usage: Use the following option to define a surface along the length of a three-dimensional beam, pipe, or truss element:

**SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
element number or element set,

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: **Name:** *surface_name*, pick three-dimensional wire region in viewport, click mouse button 2, and choose **Circumferential**

Surfaces along the Length of Two-Dimensional Beam, Pipe, and Truss Elements

Surfaces created along the length of two-dimensional beam, pipe, and truss elements can be used as main surfaces in a contact pair simulation because the underlying elements have unique element normals that lie in the plane of the model. These surfaces can also be used to prescribe distributed surface loads.

Shell, Membrane, or Rigid Element Thickness and Shell Offset

Some applications that refer to surfaces will account for underlying element thicknesses and any offset of the midsurface relative to the reference surface for surfaces based on shell, membrane, or rigid elements. For example, all the contact algorithms available in Abaqus/Explicit can account for these effects. Of the contact algorithms available in Abaqus/Standard, only the surface-to-surface small-sliding contact formulation can account for these effects. See the following sections for additional details on applications that can account for surface thickness and offset:

- *Mesh Tie Constraints*
- *Contact Formulations in Abaqus/Standard*
- *Assigning Surface Properties for General Contact in Abaqus/Explicit*
- *Assigning Surface Properties for Contact Pairs in Abaqus/Explicit*

Creating Surfaces on Gasket Elements

When surfaces are defined on gasket elements, automatic surface facet generation cannot be used because only the top and bottom element faces can be used to create surfaces (see *About Gasket Elements*). Abaqus/Standard cannot create surfaces on gasket link elements since the top and bottom surfaces are each reduced to a single node. For other gasket elements you must specify the top and bottom surfaces directly. The positive face of the element is in the thickness direction of the element. The definition of the thickness direction of all gasket elements is given in *Defining the Gasket Element's Initial Geometry*. The negative face is defined as the face in the direction opposite to the thickness direction of the element.

Input File Usage: Use the following option to define a surface on the positive face of a gasket element:

*SURFACE, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, SPOS

Use the following option to define a surface on the negative face of a gasket element:

*SURFACE, NAME=*surface_name*, TYPE=ELEMENT
element number or element set, SNEG

For example, single-sided surfaces on the positive faces of the elements in element set `GASKET` can be defined using input similar to

*SURFACE, NAME=BSURF, TYPE=ELEMENT
GASKET, SPOS

Abaqus/CAE Usage: Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: **Name:** *surface_name*, pick top or bottom faces in viewport

Surfaces on Three-Dimensional Gasket Line Elements

There are several advantages to using an element-based surface rather than a node-based surface when modeling contact in Abaqus/Standard with three-dimensional gasket line elements:

1. The local tangent directions are parallel and orthogonal to the gasket line element, which is useful for output purposes and for anisotropic friction definition.
2. Abaqus/Standard calculates the contact results as contact forces per unit length rather than just contact forces.

Surfaces created on three-dimensional gasket line elements can be used only as secondary surfaces because Abaqus/Standard cannot form unique normals for these surfaces.

Creating Interior Cross-Section Surfaces

To study the "force-flow" through various paths in a model, you must create interior surfaces that cut through one or more components (similar to a cross-section) so that you can request integrated output of the total force transmitted across these surfaces (see *Requesting Integrated Output for "Force-Flow" Studies*). Abaqus provides a simple method to create such an interior surface over the element facets, edges, or ends by cutting through a region of the model with a plane. The region can be identified using one or more element sets. If no element sets are specified, the region consists of the whole model. The cutting plane is defined by specifying the coordinates of a point on the plane and a vector normal to the plane. Alternatively, the cutting plane can be defined by specifying the global node numbers of point *a* on the plane and point *b* that lies off the cutting plane with the normal determined as the vector from point *a* to point *b*. Abaqus then automatically forms a surface close to the specified cutting plane by selecting the element facets, edges, or ends of the continuum solid, shell, membrane, surface, beam, pipe, truss, or rigid elements in the selected region. The surface generated in this manner is an approximation for the cutting plane.

Multi-point mesh constraints are ignored while generating the interior surface based on the cutting plane; therefore, the result might be a surface that is not continuous if these constraints stitch disjointed meshes together in a region that is cut by the cutting plane. When the cutting plane intersects a beam, pipe, or truss element, the entire element is shown in the Visualization module of Abaqus/CAE as being part of the surface. However, if this surface is used for integrated output, only the element nodal forces from the element end that lies on the positive side as defined by the normal to the cutting plane are included in the integrated output. Point mass and rotary elements, connector elements, spot welds, and spring elements will not be part of the generated surface even if they are cut by the cutting plane.

Input File Usage: Use the following option to define the cutting surface by specifying coordinates of a point on the plane and a vector normal to the plane:

**SURFACE*, NAME=*surface_name*, TYPE=CUTTING SURFACE, DEFINITION=COORDINATES

Use the following option to define the cutting surface by specifying global node numbers of points *a* and *b*:

**SURFACE*, NAME=*surface_name*, TYPE=CUTTING SURFACE, DEFINITION=NODES

Abaqus/CAE Usage: Interior cross-section surfaces are not supported in Abaqus/CAE.

Whole-Model Free Surface in an Abaqus/Explicit Input File

In an Abaqus/Explicit input file you can create a surface containing the exposed faces of all elements (and "contact edges" of beam, pipe, and truss elements) in the model except cohesive elements by specifying a blank element set name and a blank face identifier. This "free" surface of the model can be used as the base surface for the cropping and combining operations; without modifications this surface is similar to the default all-inclusive surface commonly used in general contact (see *About General Contact in Abaqus/Explicit*).

Input File Usage: **SURFACE*, NAME=*surface_name*, TYPE=ELEMENT
,

Abaqus/CAE Usage: The whole-model automatic free surface generation method is not supported in Abaqus/CAE.

Trimming the Perimeter of an Open Surface

An "open" surface is one that has ends in two dimensions or an outside edge in three dimensions. The ends of a two-dimensional surface and the edge of a three-dimensional surface are called the surface's "perimeter." Since Abaqus allows a surface to be defined as only a part of the surface of a body, it might have a perimeter even though it is defined on a closed body. Abaqus automatically performs surface "trimming" on solid element meshes. You can change the default setting when a surface is created, providing some basic control over the extent of surfaces.

Surface trimming:

- is a recursive procedure that removes undesirable convex corners near the perimeter of an open surface (see the example below for details);
- has no effect on closed surfaces (ones with no ends or edges);
- is performed automatically, unless the surface is used as a main surface in a finite-sliding simulation in Abaqus/Standard or the surface is used with the contact pair algorithm in Abaqus/Explicit;
- can be used only for external surfaces on solid element meshes (either specified surfaces or automatically generated free surfaces); and
- has no effect on surfaces used with the contact pair algorithm in Abaqus/Explicit.

Input File Usage: Use the following option to suppress automatic surface trimming:

**SURFACE*, TYPE=ELEMENT, NAME=*surface_name*, TRIM=NO

Abaqus/CAE Usage: Automatic surface trimming cannot be suppressed in Abaqus/CAE.

The Effect of Surface Trimming

The effect of surface trimming is best explained by means of an example. *Figure 3* illustrates the effect of trimming for two different surfaces defined on the same simple two-dimensional mesh.

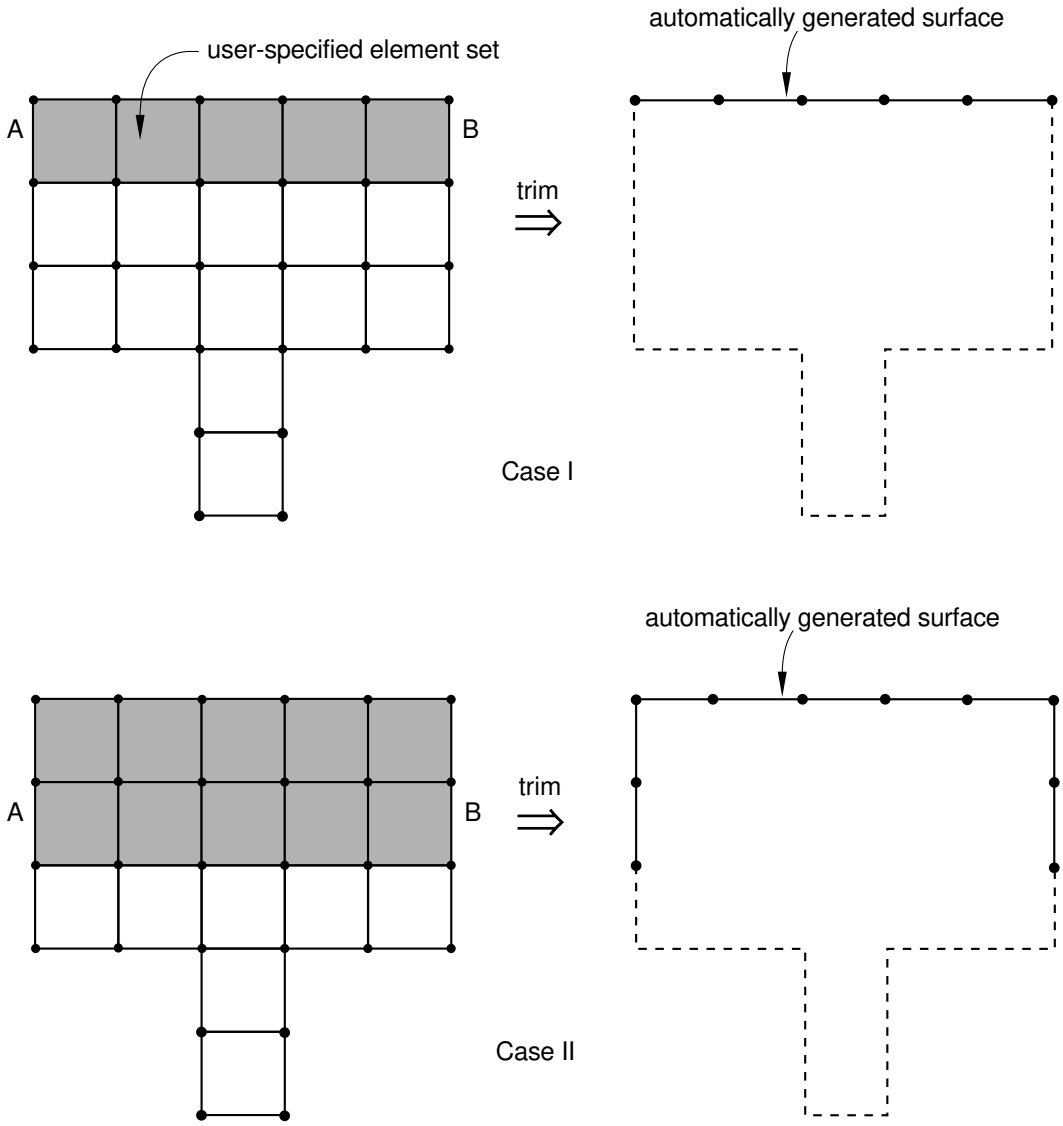


Figure 3: Case I: Faces *A* and *B* are removed when trimming is done since one node of each of the faces is an end node and the other is a corner node. Case II: Faces *A* and *B* are not removed when trimming is done since one node of each of the faces is an end node but the other is not a corner node.

In Case I the surface definition consists of a single layer of elements on the perimeter of the model. Using automatic surface facet generation, the resulting default surface (curve) includes the vertical element faces *A* and *B* since these faces lie on the perimeter of the model. Trimming the default surface created in Case I eliminates faces *A* and *B* since their presence results in the two spurious corners near the perimeter of the curve.

Abaqus uses a special criterion in deciding to remove faces *A* and *B* from the original open curve. A face is removed if one of its end nodes is an endpoint and either of the following is true: another face node is a node on

an element corner belonging to the curve or the face normal differs by more than 30° from the normal of an adjacent face also belonging to the curve. To be a node on an element corner belonging to the curve means to be a node on two different faces of the same element, both of which are part of the curve. The face removal criterion is applied recursively to the curve definition until all corners on or near the perimeter of the curve have been removed. This procedure is generalized for three-dimensional surface definitions.

In Case II in *Figure 3* trimming would not result in the elimination of faces *A* and *B* because neither of the endpoints of these two faces meets the criterion described above.

Why Abaqus Will, by Default, Trim Most Surfaces

Trimming of surfaces used for application of distributed loads is usually desired since loads are normally applied to specific sides of a body. Any surface that is used for application of a distributed load will, by default, be trimmed.

In Abaqus/Standard trimming the secondary surface in contact or interaction simulations results in more accurate estimates of the contact pressures, heat fluxes, and electrical current densities along the perimeter of the surface. Any surface that is used as a secondary surface in a contact or interaction simulation will, by default, be trimmed. If the secondary surface is left untrimmed, the nodes at the corners of the surface will be assigned additional contact area from the element faces around the corners that might never be involved in the interaction between the surfaces. This additional contact area introduces errors into the estimates of the contact output variables at those nodes. Main surfaces in small-sliding simulations will, by default, be trimmed; Abaqus/Standard will normally form a better approximate surface. However, main surfaces in finite-sliding contact simulations will, by default, be left untrimmed, and they should extend far enough away from all expected regions of contact. This practice protects against the possibility of the secondary surface nodes sliding off the main surface (see *Common Difficulties Associated with Contact Modeling in Abaqus/Standard*). By default, element-based surfaces defined with element face identifiers are also left untrimmed in Abaqus/Standard.

Node-Based Surface Definition

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Surfaces*
- *Mesh Tie Constraints*
- *About Contact Interactions*
- **SURFACE*

Overview

A node-based “surface”:

- can be used only as a “secondary surface” in contact calculations;
- can be used as a “secondary” or “main surface” in a surface-based tie constraint;
- is convenient in three-dimensional cases where Abaqus cannot construct a unique physical surface on the elements, such as a pipe modeled with pipe elements contacting the ocean floor or cables modeled with trusses contacting the ground after they break;
- should be used with caution or not at all if accurate contact stresses are needed or if heat will be exchanged between the two surfaces;
- can be assigned a nonzero thickness for use with the general contact algorithm in Abaqus/Explicit;
- should not be used to model a shell or membrane surface if the thickness and midsurface offset need to be considered in the problem;
- must either contain nodes that are all part of the same rigid body or not contain any nodes that are part of a rigid body if the node-based surface is to be used in a penalty contact pair in Abaqus/Explicit;
- in Abaqus/Standard does not provide heat conduction between surfaces in fully coupled temperature-displacement analysis or pore fluid flow between surfaces in coupled pore pressure–displacement analysis;
- in Abaqus/Standard does not provide heat conduction and electrical conduction between surfaces in a fully coupled thermal-electrical-structural analysis; and
- does not include circumferential friction when used with axisymmetric elements with twist (CGAX, MGAX elements).

Alternatives to node-based surfaces are element-based surfaces (see *Element-Based Surface Definition*) and, in the case of rigid surfaces, analytical rigid surfaces (see *Analytical Rigid Surface Definition*). See *Operating on Surfaces* for information on defining surfaces using Boolean combinations of existing surfaces.

Creating a Node-Based Surface

You create a node-based surface by specifying the nodes or node sets that form the surface. You must assign a name to the node-based surface; this name will be used when defining contact interactions that involve the surface.

An optional associated area can be defined for each node. If no area is defined for a node and the surface is defined in a contact pair, the area specified as part of the contact property definition is used. If no area is specified as part of the contact property definition, a unit area is used.

In Abaqus/Explicit the area used in contact pair calculations for a node in a node-based surface is always 1.0, regardless of the user-specified value. Therefore, the contact pressure output variable in Abaqus/CAE should be interpreted as the contact force when a node-based surface is used for contact pairs in Abaqus/Explicit.

In models that are defined in terms of an assembly of part instances, all surfaces must belong to a part, part instance, or the assembly. Additional rules are given in *Assembly Definition*.

When the nodes of shell and membrane elements are used in a node-based surface, the thickness and midsurface offset of the shell or membrane at each node are not considered. However, a nonzero thickness can be assigned to node-based surfaces when used with the general contact algorithm in Abaqus/Explicit (see *Assigning Surface Properties for General Contact in Abaqus/Explicit* for more information).

Input File Usage: **SURFACE*, NAME=*name*, TYPE=NODE
node number or node set, area

Analytical Rigid Surface Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- *About Contact Interactions*
- *RSURFU*
- **RIGID BODY*
- **SURFACE*

Overview

An analytical rigid surface:

- can be two-dimensional or three-dimensional;
- must be defined as model data;
- can be used with the infinitesimal-sliding, small-sliding, or finite-sliding mechanical contact formulations;
- should be oriented such that the analytical rigid surface's outward normal points toward any body it may contact; and
- is associated with a node, known as the rigid body reference node, whose motion governs the motion of the surface.

What Are Analytical Rigid Surfaces and Why Use Them?

Analytical rigid surfaces are geometric surfaces with profiles that can be described with straight and curved line segments. These profiles can be swept along a generator vector or rotated about an axis to form a three-dimensional surface. An analytical rigid surface is associated with a rigid body reference node, whose motion governs the motion of the surface. An analytical rigid surface does not contribute to the rigid body's mass or inertia properties (see *Rigid Body Definition*). The degrees of freedom of the rigid body reference node become active only when the analytical surface is used in a contact interaction or when an element (such as a spring element or a mass element) is connected to the rigid body reference node.

Analytical rigid surfaces are always single-sided with their orientation specified through their definition. Therefore, contact interaction is recognized only on the outer boundary of an analytical rigid surface. To model contact on both sides of a thin structure, use an analytical rigid surface that wraps around the boundary of the thin structure.

Advantages

Using analytical rigid surfaces instead of defining element-based rigid surfaces provides two important advantages in contact modeling.

- Many curved geometries can be modeled exactly with analytical rigid surfaces because of the ability to parameterize the surface with curved line segments. The result is a smoother surface description, which can reduce contact noise and provide a better approximation to the physical contact constraint.
- Using analytical rigid surfaces instead of rigid surfaces formed by element faces may result in decreased computational cost incurred by the contact algorithm.

The use of curved line segments instead of many linear facets will decrease the time spent in contact tracking operations. Additional computational savings may be realized in three dimensions because of the intrinsic two-dimensional descriptions of the analytical surfaces.

Disadvantages

There are also some disadvantages to using analytical rigid surfaces for contact modeling.

- An analytical rigid surface must always act as a main surface in a contact interaction. Therefore, contact cannot be modeled between two analytical rigid surfaces.
- Contact forces and pressures cannot be contoured on an analytical rigid surface. However, contact forces and pressures can be plotted on the secondary surface.
- The use of a very large number (thousands) of segments to define an analytical rigid surface can degrade performance. In most cases it is not necessary to use a large number of segments to define an analytical rigid surface, because curved segment types are allowed. In rare cases in which a very large number of segments would be necessary, the analysis may be more efficient if an element-based rigid surface is used instead (see *Element-Based Surface Definition*).
- An analytical rigid surface does not contribute to the mass and rotary inertia properties of the rigid body with which it is associated. Therefore, if the mass distribution on an analytical rigid surface needs to be accounted for, equivalent mass and rotary inertia properties must be defined for the rigid body by using MASS and ROTARYI elements, or a finite element discretization of the surface should be used instead of an analytical rigid surface (see *Rigid Body Definition*).
- In Abaqus/Explicit reaction force output for a rigid body containing an analytical rigid surface is calculated only for constraints that are active at the reference node (e.g., constraints specified as boundary conditions). If the net contact force on the rigid body corresponding to an unconstrained degree of freedom is desired, it must be calculated from the rigid body's acceleration and mass.

Creating an Analytical Rigid Surface

You can define the following types of simple, two- or three-dimensional, geometric analytical surfaces:

- planar (two-dimensional) surfaces,
- three-dimensional cylindrical (swept) surfaces, and
- three-dimensional surfaces of revolution.

In Abaqus/Standard if none of these surfaces is adequate, you can define a more general analytical surface with user subroutine *RSURFU*.

Analytical rigid surfaces are useful when the cross-sections of the surfaces can be represented by straight and curved line segments. The curved segments can be either circular or parabolic arcs. In two-dimensional simulations the line segments are defined in the global coordinate system of the deformable model. In three-dimensional simulations a local, two-dimensional coordinate system must be created, and the line segments are then defined in that system. The two standard types of three-dimensional analytical rigid surfaces available are shown in *Figure 1*.

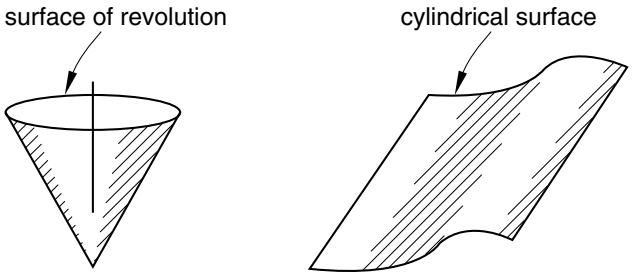


Figure 1: Examples of three-dimensional rigid surfaces.

You must indicate which type of analytical surface (planar, cylindrical, or revolution) is being created and assign a name to the surface. In addition, you must define the analytical surface as part of a rigid body by specifying the name of the analytical surface and the rigid body reference node that will control the motion of the surface in a rigid body definition.

An Abaqus model can be defined in terms of an assembly of part instances (see *Assembly Definition*). A part can contain only one analytical surface. A part containing an analytical surface definition cannot also contain elements.

Input File Usage: Use both of the following options to create an analytical rigid surface:

**SURFACE*, TYPE=*analytical_surface_type*, NAME=*name*
**RIGID BODY*, ANALYTICAL SURFACE=*name*, REF NODE=*n*

Abaqus/CAE Usage: Part module: **Create Part**: **Name:** *analytical_rigid_part*: select **Analytical rigid** as the **Type**

Then do one of the following:

Any module except Sketch, Job, and Visualization: **Tools->Surface->Create**: select *analytical_rigid_part*

Interaction module: **Create Constraint**: **Rigid body**: **Analytical Surface**: **Edit**: select *analytical_rigid_part*

Interaction module: **Create Interaction**: *any valid type*: select *analytical_rigid_part* as one of the regions involved in contact

Defining a Surface Profile

The surface profile is the collection of line segments defining the cross-section of the surface. The surface type determines whether the profile is swept (cylindrical surfaces), revolved (surfaces of revolution), or, in the two-dimensional case, used as is (planar surfaces).

You construct a profile by providing the endpoint of each line segment in the profile; the starting point is always the endpoint of the previous segment, or, in the case of the first segment, the point specified as the starting point. The center points of circular arcs must be given. Abaqus can define only arcs that are less than 179.74°; thus, it will use the shorter arc defined by the data provided (use two adjacent arcs to define a longer arc). For parabolic arcs you must give a third point that lies on the parabola and within the arc.

Two-Dimensional Rigid Surfaces

To define a planar rigid surface, specify the line segments forming the rigid surface's profile in the global coordinate system. If the analytical surface is being defined inside a part, specify the line segments in the local part coordinate system.

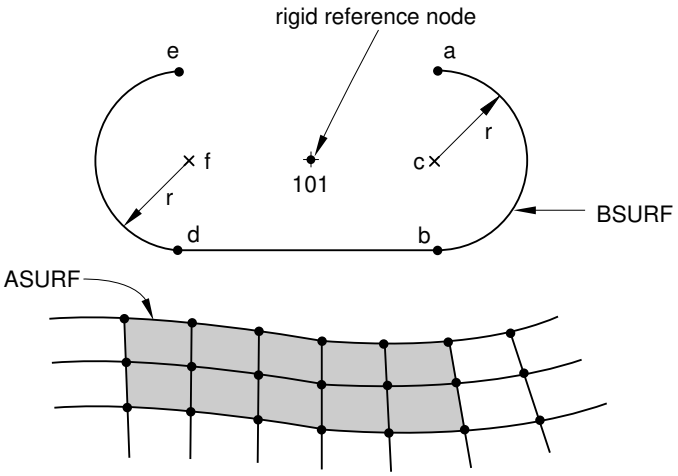
Input File Usage:

**SURFACE*, TYPE=SEGMENTS, NAME=*name*
data lines to define the line segments forming the surface

For example, the definition of the two-dimensional rigid surface depicted in the figure below is

**SURFACE*, TYPE=SEGMENTS, NAME=BSURF
START, x_a, y_a
CIRCL, x_b, y_b, x_c, y_c
LINE, x_d, y_d
CIRCL, x_e, y_e, x_f, y_f
**RIGID BODY*, ANALYTICAL SURFACE=BSURF, REF NODE=101

where x_i and y_i are the global coordinates of the points shown in the figure.



Abaqus/CAE Usage:

Part module: **Create Part**: **Name:** *analytical_rigid_part*: select **2D Planar** or **Axisymmetric** as the **Modeling Space** and **Analytical rigid** as the **Type**

Three-Dimensional Cylindrical Rigid Surfaces

To define a cylindrical rigid surface in a model that is not defined in terms of an assembly of part instances, specify the points *a*, *b*, and *c* shown in *Figure 2* that define the local coordinate system.

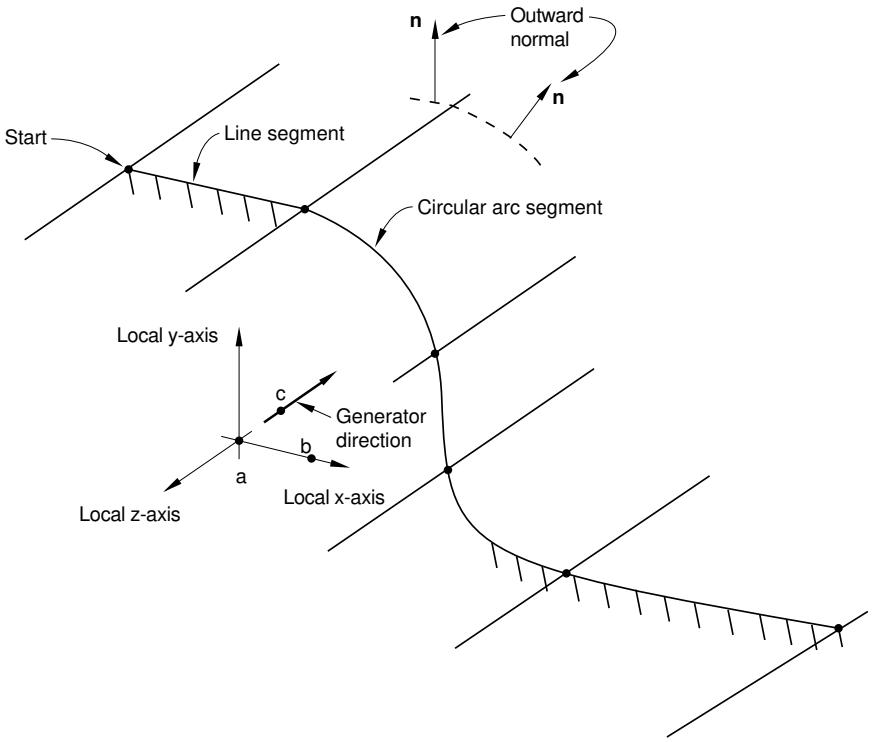


Figure 2: Cylindrical rigid surface.

Give the coordinates of these points—(X_a, Y_a, Z_a), (X_b, Y_b, Z_b), and (X_c, Y_c, Z_c)—in the default global coordinate system. As shown in *Figure 2*, point *a* defines the origin of the local system; point *b* defines the local *x*-axis; and point *c* defines the generator vector, which is the *negative* local *z*-axis. If the segment ac is not perpendicular to ab, Abaqus will automatically adjust point *c* within the plane defined by points *a*, *b*, and *c*, such that they become perpendicular. The line segments forming the profile of the rigid surface are defined in the local *x*–*y* plane. The three-dimensional surface is formed by sweeping this profile along the generator vector. The resulting surface extends to infinity in both the positive and negative directions of the generator vector.

To define a cylindrical rigid surface within a part, specify the line segments forming the profile of the rigid surface in the part coordinate system. For an analytical surface defined within a part (or part instance), point *a* is located at the origin of the part coordinate system, point *b* is located on the part *x*-axis, and point *c* is located on the negative part *z*-axis. If the segment ac is not perpendicular to ab, Abaqus will automatically adjust point *c* within the plane defined by points *a*, *b*, and *c*, such that they become perpendicular. You cannot redefine this analytical surface coordinate system; instead, you can position the surface in the model by giving positioning data when you instance the part (see *Assembly Definition*).

Input File Usage: **SURFACE*, TYPE=CYLINDER, NAME=*name*
$X_a, Y_a, Z_a, X_b, Y_b, Z_b$
X_c, Y_c, Z_c
data lines to define the line segments forming the surface

For example, the following input, where x_i and y_i are points in the local coordinate system, would define the rigid surface shown in *Figure 2* in a model that is not defined in terms of an assembly of part instances (the reference node is not shown in the figure):

*SURFACE, TYPE=CYLINDER, NAME=CSURF
X_a, Y_a, Z_a, X_b, Y_b, Z_b
X_c, Y_c, Z_c
START, x_s, y_s
LINE, x_1, y_1
CIRCL, …
…
*RIGID BODY, ANALYTICAL SURFACE=CSURF, REF NODE=*n*

Leave the first two data lines blank to define a cylindrical rigid surface within a part.

Abaqus/CAE Usage: Part module: **Create Part**: **Name:** *analytical_rigid_part*: select **3D** as the **Modeling Space**, **Analytical rigid** as the **Type**, and **Extruded shell** as the **Base Feature**

Three-Dimensional Surfaces of Revolution

To define a rigid surface of revolution in a model that is not defined in terms of an assembly of part instances, specify the two points *a* and *b* shown in *Figure 3* that define the local coordinate system.

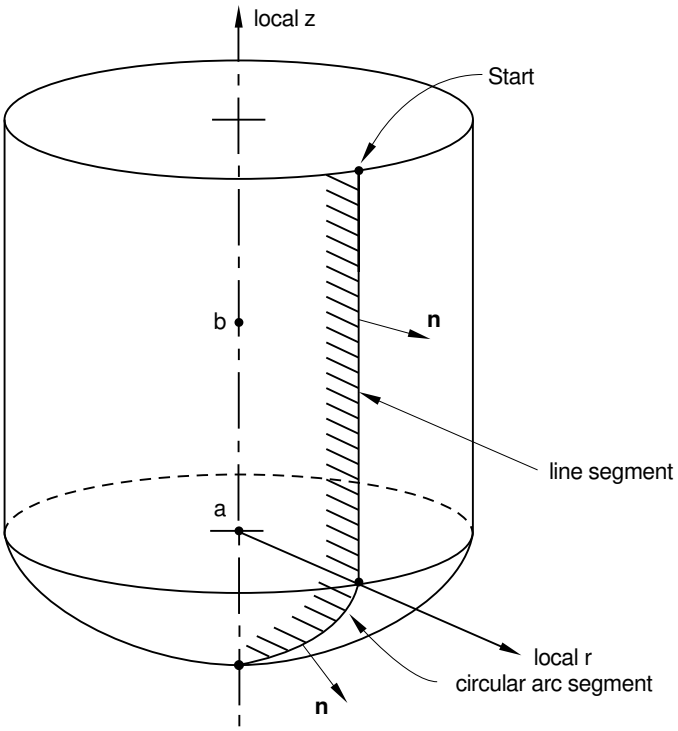


Figure 3: Rigid surface of revolution.

Give the coordinates of these points—(X_a, Y_a, Z_a) and (X_b, Y_b, Z_b)—in the default global coordinate system. As shown in *Figure 3*, point *a* defines the origin of the local system, and the vector from *a* to *b* defines the local *z*-axis, which is the axis of a cylindrical coordinate system. The line segments forming the profile of the surface of revolution are defined in the local *r*–*z* plane, where the local *r*-axis aligns with the radial axis of the cylindrical coordinate system. The three-dimensional surface is formed by revolving this profile about the axis of the cylindrical system, the local *z*-axis.

To define a rigid surface of revolution within a part, specify the line segments forming the cross-section of the rigid surface in the local part coordinate system. For an analytical surface defined within a part (or part instance), point *a* is located at the origin of the part coordinate system, the part *x*-axis aligns with the radial axis of the cylindrical coordinate system, and point *b* is located on the part *y*-axis. You cannot redefine this local axis; instead, you can position the surface in the model by giving positioning data when you instance the part (see *Assembly Definition*).

Input File Usage:

**SURFACE*, TYPE=REVOLUTION, NAME=*name*
$X_a, Y_a, Z_a, X_b, Y_b, Z_b$
data lines to define the line segments forming the surface

For example, the following input would define the rigid surface shown in *Figure 3* (the reference node is not shown in the figure):

**SURFACE*, TYPE=REVOLUTION, NAME=REVSURF
X_a, Y_a, Z_a, X_b, Y_b, Z_b
START, x_s, y_s
LINE, …
CIRCL, …
…
**RIGID BODY*, ANALYTICAL SURFACE=REVSURF, REF NODE=999

Leave the first data line blank to define a rigid surface of revolution within a part.

Abaqus/CAE Usage:

Part module: **Create Part**: **Name:** *analytical_rigid_part*: select **3D** as the **Modeling Space**, **Analytical rigid** as the **Type**, and **Revolved shell** as the **Base Feature**

Defining the Surface Normals

The outward surface normal for analytical rigid surfaces is determined by the direction of the line segments forming the profile of the surface. The sequence of line segments defines a vector $\mathbf{s}$ along the rigid surface from the starting point of the first segment to the ending point of the last segment. The outward surface normal is created by taking the cross product of the vector $\mathbf{e_3}$, the unit normal to the plane in which the surface is defined, and the vector $\mathbf{s}$, the tangent to the surface: $\mathbf{n} = \mathbf{e_3} \times \mathbf{s}$. *Figure 4* shows the vector $\mathbf{s}$ in the definition plane of an analytical rigid surface.

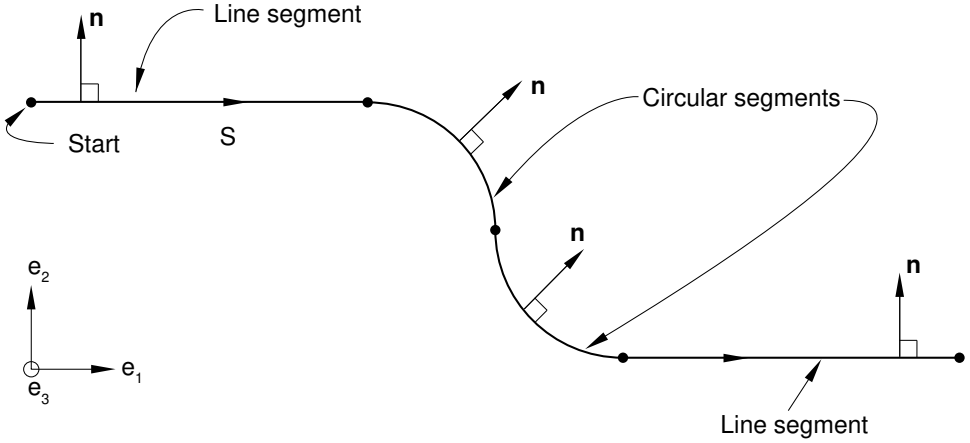


Figure 4: Orientation of surface normals for a rigid surface.

The unit vector $\mathbf{e_3}$ is defined such that $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$ form a right-handed orthonormal coordinate system. In-plane coordinate directions $\mathbf{e_1}$ and $\mathbf{e_2}$ depend on the type of analytical rigid surface being defined. For two-dimensional analytical rigid surfaces they correspond to the global *X*- and *Y*-axes in planar models and the *r*- and *z*-axes in axisymmetric models. For cylindrical rigid surfaces they correspond to the local *x*- and *y*-axes, and for rigid surfaces of revolution they correspond to the local *r*- and *z*-axes. The outward normals for a cylindrical rigid surface and rigid surface of revolution are shown in *Figure 2* and *Figure 3*, respectively.

If the line segments are specified in the wrong order, the surface normals of a rigid surface will appear in exactly the opposite direction to what was intended. Such a mistake can be corrected only by specifying the line segments in the opposite sequence.

Smoothing Analytical Rigid Surfaces

In many cases it can be beneficial to smooth surfaces to more accurately represent the surface geometry. In particular, it can be very difficult to obtain a converged solution in a finite-sliding Abaqus/Standard simulation if the main surface does not have continuous normal and surface tangent vectors (see *Contact Formulations in Abaqus/Standard*); therefore, it is important to smooth any sharp corners on the main surface so that discontinuities in these vectors are eliminated.

By default, Abaqus does not smooth main surfaces that are analytical rigid surfaces. Smooth transitions between adjacent line segments can always be created by manually inserting additional curved line segments. Alternatively, smooth surfaces can be generated automatically by Abaqus. You specify the radius of curvature, *r*, in the units of length used in the model, that Abaqus will use to construct a smooth transition between any discontinuous line segments forming the rigid surface. The default value of zero provides no smoothing of the surface.

The effect of a fillet radius on adjoining line segments and on adjoining line and circular arc segments is illustrated in *Figure 5*.

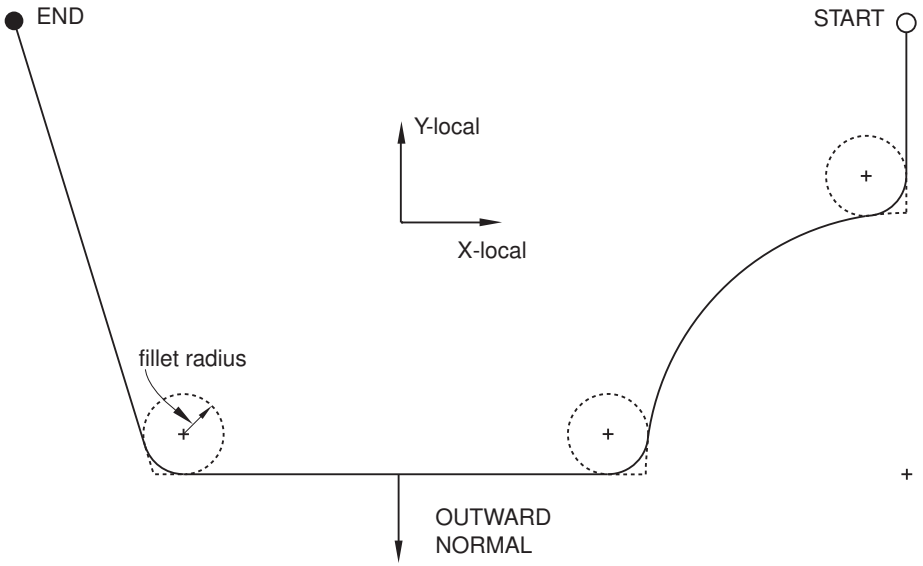


Figure 5: Effect of fillet radius on an analytical rigid surface.

The sharp corners have been smoothed using the fillet radius so that the normal and tangent surface vectors are continuous along the entire main surface. Any value *r* can be used in a model. However, if *r* is greater than the length of either of the two adjacent segments, no smoothing will occur. Therefore, a practical limit on the size of *r* is the length of the smallest line segment forming the surface.

Input File Usage: **SURFACE*, TYPE=*analytical_surface_type*, NAME=*name*, FILLET RADIUS=*r*

Abaqus/CAE Usage: When you create an analytical rigid part in Abaqus/CAE, you can create a fillet radius between segments or join the segments using arcs. See *Sketching simple objects*.

Surface Tangent Conventions

Abaqus forms analytical rigid surfaces such that the first surface tangent, $\mathbf{t_1}$, is always along the direction of the line segments forming the surface $\mathbf{s}$. The second surface tangent, $\mathbf{t_2}$, is defined such that the outward surface normal and the two surface tangents form a right-handed orthonormal system, as shown in *Figure 6*.

a. Two-dimensional cases b. Three-dimensional cases

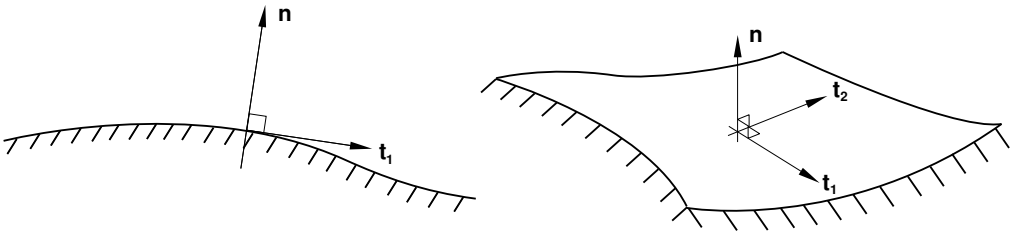


Figure 6: Surface tangent and outward normal definitions for analytical rigid surfaces.

Creating an Analytical Rigid Surface in a User Subroutine

More complicated analytical rigid surfaces can be defined in Abaqus/Standard by user subroutine *RSURFU*. Writing subroutine *RSURFU* to create a smooth surface is usually difficult, and convergence problems are often caused by inadequate surface definition in this subroutine. When using *RSURFU*, ensure that the outward surface normal and the two surface tangents form a right-handed orthonormal system. In two-dimensional cases the second surface tangent is always (0, 0, −1). You must also ensure that the surface is smooth in finite-sliding simulations and that the orientation of the rigid surface relative to the deformable surface is reasonable (i.e., the rigid surface cannot be inside the deformable surface).

Input File Usage: **SURFACE*, TYPE=USER, NAME=*name*

Abaqus/CAE Usage: User subroutine *RSURFU* is not supported in Abaqus/CAE.

Defining Analytical Rigid Surfaces When Drag Chain or Rigid Surface Elements Are Used

An alternative method of defining analytical rigid surfaces must be used to define the surface of the seabed when three-dimensional drag chain elements (available only in Abaqus/Standard) are used. This alternative method must also be used when rigid surface elements are used; these elements are required only when CAXA or SAXA elements contact a rigid surface. For this method the rigid surface must be flat and parallel to the *x*–*y* plane.

In a model defined in terms of an assembly of part instances, the rigid surface definition must appear inside the same part definition as the drag chain or rigid surface elements.

You must indicate which type of analytical surface (planar, cylindrical, or user-defined) is being created. Cylindrical rigid surfaces are not valid for use with CAXA or SAXA elements. In addition, you must assign a name to the surface and identify the rigid body reference node that will control the motion of the surface.

Input File Usage: **RIGID SURFACE*, TYPE=*surface_type*, NAME=*name*, REF NODE=*n*

Abaqus/CAE Usage: Drag chain and rigid surface elements are not supported in Abaqus/CAE.

Two-Dimensional Rigid Surfaces

To define a planar rigid surface, define the line segments forming the rigid surface's cross-section in the global coordinate system. You must provide the endpoint of each line segment; the starting point is always the endpoint of the previous segment, or, in the case of the first segment, the point specified as the starting point. The centers of the circular arcs, points *c* and *f* in *Figure 7*, must be given. Abaqus can define only arcs that are less than, but not equal to, 179.74°; thus, it will use the shorter arc defined by the data provided (use two adjacent arcs to define a longer arc). For parabolic arcs you must give a third point that lies on the parabola and within the arc.

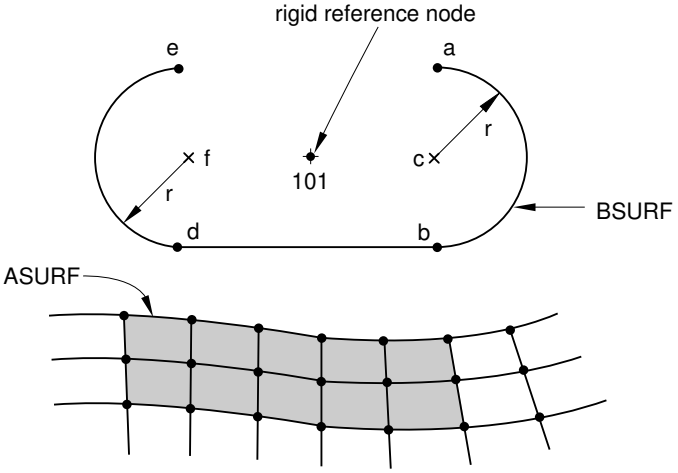


Figure 7: Two-dimensional analytical rigid surface contacting a deformable body.

Input File Usage: **RIGID SURFACE*, TYPE=SEGMENTS, NAME=*name*, REF NODE=*n*
START, *starting point X- or r-coordinate, starting point Y- or z-coordinate*
data lines to define the endpoints of the line segments forming the surface, beginning with the word LINE (for straight line segments), CIRCL (for circular arc segments), or PARAB (for parabolic arc segments)

Abaqus/CAE Usage: Drag chain and rigid surface elements are not supported in Abaqus/CAE.

Three-Dimensional Cylindrical Rigid Surfaces

To define a cylindrical rigid surface, specify the points *a*, *b*, and *c* shown in *Figure 2* that define the local coordinate system. Give the coordinates of these points—(X_a, Y_a, Z_a), (X_b, Y_b, Z_b), and (X_c, Y_c, Z_c)—in the default global coordinate system. As shown in *Figure 2*, point *a* defines the origin of the local system; point *b* defines the local *x*-axis; and point *c* defines the generator vector, which is the *negative* local *z*-axis. The line segments forming the cross-section of the rigid surface are defined in the local *x*–*y* plane. The three-dimensional surface is formed by sweeping this cross-section along the generator vector. The resulting surface extends to infinity in both the positive and negative directions of the generator vector.

Input File Usage: **RIGID SURFACE*, TYPE=CYLINDER, NAME=*name*, REF NODE=*n*
$X_a, Y_a, Z_a, X_b, Y_b, Z_b$
X_c, Y_c, Z_c
START, *starting point x-coordinate, starting point y-coordinate*
data lines to define the endpoints of the line segments forming the surface, beginning

with the word LINE (for straight line segments), CIRCL (for circular arc segments), or PARAB (for parabolic arc segments)

Abaqus/CAE Usage: Drag chain and rigid surface elements are not supported in Abaqus/CAE.

Eulerian Surface Definition

Products: Abaqus/Explicit

References:

- *About Surfaces*
- *Eulerian Analysis*
- *About Contact Interactions*
- **EULERIAN SECTION*
- **SURFACE*

Overview

An Eulerian surface:

- must be three-dimensional;
- must be defined as model data;
- can be used with the general contact algorithm in Abaqus/Explicit; and
- is created by specifying the name of an Eulerian material instance.

What Are Eulerian Surfaces and Why Use Them?

An Eulerian surface represents the exterior surface of a particular Eulerian material instance in an Abaqus/Explicit analysis. Since Eulerian materials flow through the Eulerian mesh, their surfaces cannot be defined by a simple list of element faces. Instead, these surfaces often lie within Eulerian elements and must be computed in each time increment using element volume fraction data.

You can use Eulerian surfaces to define specific interactions with Lagrangian surfaces in Abaqus/Explicit's general contact algorithm. Once defined, you can reference Eulerian surfaces in inclusions, exclusions, and interaction definitions. You cannot combine or crop Eulerian surfaces.

Eulerian surface definitions are not required for the use of Eulerian-Lagrangian contact. If you specify “automatic” contact for the entire model, the exterior surface of all Eulerian materials will automatically be considered for contact.

Advantages of Creating Eulerian Surfaces

You can use Eulerian surfaces to:

- Assign contact properties for contact interactions involving a particular Eulerian material instance.
- Apply surface tension to the material surface involving a particular Eulerian material instance.
- Exclude interactions between Eulerian materials and Lagrangian bodies that are unlikely to make contact, simplifying the contact problem and reducing computational cost.

Creating an Eulerian Surface

To create an Eulerian surface, you must specify the name of a material instance that is present in the model. The material instance names are defined as part of the Eulerian section (see *Eulerian Elements*). Abaqus/Explicit calculates the exterior boundary of the specified material instance and defines a surface corresponding to that boundary. The surface is recalculated in each time increment as the material deforms.

Input File Usage: **SURFACE*, TYPE=EULERIAN MATERIAL, NAME=*name*
material instance name,

Operating on Surfaces

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Surfaces*
- *Coupling Constraints*
- *Mesh-Independent Fasteners*
- *About General Contact in Abaqus/Explicit*

Overview

Combined surfaces:

- are created by performing a Boolean operation (union, intersection, or difference) on existing surfaces;
- can be formed from element-based or node-based surfaces;
- cannot be formed from Eulerian surfaces;
- can be used in the same way as other element-based or mode-based surfaces in Abaqus/Standard; and
- cannot be used with contact pairs in Abaqus/Explicit (but can be used with general contact in Abaqus/Explicit).

Cropped surfaces:

- are created by cropping an existing surface and keeping only that part of the surface that is enclosed in a specified rectangular box;
- can be formed from element-based or node-based surfaces;
- cannot be formed from Eulerian surfaces;
- can be used in the same way as other element-based or mode-based surfaces in Abaqus/Standard; and
- cannot be used with contact pairs in Abaqus/Explicit (but can be used with general contact in Abaqus/Explicit).

Creating a Combined Surface

You must assign a name to the combined surface; this name can be used with other features that refer to surfaces.

In models that are defined in terms of an assembly of part instances, all surfaces must belong to a part, part instance, or the assembly. Surfaces can be created at the part level and combined at the assembly level. Additional rules are given in *Assembly Definition*.

The surfaces being combined must be the same type; i.e., an element-based surface can be combined with another element-based surface but not with a node-based surface. Combined surfaces can be used to create another combined surface.

Union of Existing Surfaces

Any number of existing surfaces can be combined to create a new surface. If the surfaces being combined are element-based surfaces, the new surface will also be an element-based surface and any overlap among the surfaces will be merged. Similarly, if the surfaces being combined are node-based surfaces, the new surface will be a node-based surface and any overlap among the surfaces will be merged.

Input File Usage: **SURFACE*, NAME=*name*, COMBINE=UNION
list of surface names

Intersection or Difference of Existing Surfaces

The intersection or difference of two existing surfaces can be used to create a new surface. The difference operation subtracts the second surface from the first surface. When the intersection or difference operations are performed on element-based surfaces, they act only on the facets. A warning message is issued if the intersection operation results in an empty surface.

Input File Usage: Use the following option to create a new surface based on the intersection of two existing surfaces:

**SURFACE*, NAME=*name*, COMBINE=INTERSECTION
first surface name, second surface name

Use the following option to create a new surface based on the difference of two existing surfaces:

**SURFACE*, NAME=*name*, COMBINE=DIFFERENCE
first surface name, second surface name

Creating a Cropped Surface

You can create a new surface that will contain only those faces of an existing surface that have nodes inside a specified cropping box. For a node-based surface the new surface will contain only those nodes that are enclosed inside the cropping box. If the face has at least one node inside the box, the entire face is accepted as valid. You must assign a name to the new surface and specify the name of the existing surface from which the new surface is to be generated. Only one surface can be specified.

To define the location of the box, specify the coordinates of the lower corner of the box (X_{min}, Y_{min}, Z_{min}) and the coordinates of the opposite (upper) corner of the box (X_{max}, Y_{max}, Z_{max}). The cutting box can be rotated about the lower corner (X_{min}, Y_{min}, Z_{min}) if an optional rotation is defined. The coordinates of the two points, *a* and *b*, that define the rotation are given in the unrotated system. These points should be defined such that point *a* lies on the rotated *X*-axis and point *b* lies on the *X*–*Y* plane and close to the *Y*-axis.

Input File Usage: **SURFACE*, NAME=*name*, CROP
old_surface_name
X_{min}, Y_{min}, Z_{min}, X_{max}, Y_{max}, Z_{max}
X_a, Y_a, Z_a, X_b, Y_b, Z_b

For example, to crop the surface that contains all exposed faces in the model, use the following input:

```
*SURFACE, TYPE=ELEMENT, NAME=entire_surface
,
*SURFACE, NAME=name, CROP
entire_surface
Xmin, Ymin, Zmin, Xmax, Ymax, Zmax
Xa, Ya, Za, Xb, Yb, Zb
```

Rigid Body Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- *Element-Based Surface Definition*
- *Analytical Rigid Surface Definition*
- *Rigid Elements*
- **RIGID BODY*
- *Defining rigid body constraints*

Overview

A rigid body:

- can be two-dimensional planar, axisymmetric, or three-dimensional;
- is associated with a node, called the rigid body reference node, whose motion governs the motion of the entire rigid body;
- can consist of nodes, elements, and surfaces;
- can act as a method of constraint;
- can be used with connector elements in multibody dynamic simulations;
- can be used to prescribe the motion of a rigid surface for contact modeling;
- can be computationally efficient and, in Abaqus/Explicit, does not affect the global time increment; and
- can have temperature gradients or be isothermal in a fully coupled temperature-displacement analysis where thermal interactions are considered.

What Is a Rigid Body?

A rigid body is a collection of nodes, elements, or surfaces whose motion is governed by the motion of a single node, called the rigid body reference node. The relative positions of the nodes and elements that are part of the rigid body remain constant throughout a simulation. Therefore, the constituent elements do not deform but can undergo large rigid body motions. The mass and inertia of a rigid body can be calculated based on contributions from its elements or can be assigned specifically. Analytical surfaces can also be made part of the rigid body, whereas any surfaces based on the nodes or elements of a rigid body are associated automatically with the 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 from distributed loads applied to elements that are part of the rigid body. Rigid bodies interact with the remainder of the model in several ways. Rigid bodies can connect at the nodes to deformable elements, and surfaces defined on rigid bodies can continue on these deformable elements, provided that compatible element types are used. Rigid bodies can also be connected to other rigid bodies by connector elements (see *About Connectors*). Surfaces defined on rigid bodies can contact surfaces defined on other bodies in the model.

Determining When to Use a Rigid Body

Rigid bodies can be used to model very stiff components, either fixed or undergoing large motions. For example, rigid bodies are ideally suited for modeling tooling (that is, punch, die, drawbead, blank holder, roller, etc.). They can also be used to model constraints between deformable components, and they provide a convenient method of specifying certain contact interactions. Rigid bodies can be used with connector elements to model a wide variety of multibody dynamic problems.

It might be useful to make parts of a model rigid for model 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.

In multibody dynamic simulations rigid bodies are useful for many reasons. Although the motion of the rigid body is governed by the six degrees of freedom at the reference node, rigid bodies allow accurate representation of the geometry, mass, and rotary inertia of the rigid body. Furthermore, rigid bodies provide accurate visualization and postprocessing of the model.

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 reference node.

Rigid bodies are particularly effective for modeling relatively stiff parts of a model in Abaqus/Explicit for which tracking waves and stress distributions are 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.

Creating a Rigid Body

You must assign a rigid body reference node to the rigid body.

Input File Usage: **RIGID BODY*, REF NODE=*n*

Abaqus/CAE Usage: Interaction module:
Tools->**Reference Point**: select a point to act as a reference point
Create Constraint: **Rigid body**: **Point: Edit**: select reference point region

The Rigid Body Reference Node

A rigid body reference node has both translational and rotational degrees of freedom and must be defined for every rigid body. If the reference node has not been assigned coordinates, Abaqus will assign it the coordinates of the global origin by default. Alternatively, you can specify that the reference node should be placed at the center of mass of the rigid body. In fully coupled temperature-displacement analysis where a rigid body is considered as isothermal, a single temperature degree of freedom describing the temperature of the rigid body exists at the rigid body reference node. The rigid body reference node:

- can be connected to mass, rotary inertia, capacitance, or deformable elements;
- cannot be a rigid body reference node for another rigid body; and
- can have a temperature degree of freedom if the body is an isothermal rigid body.

Positioning the Reference Node at the Center of Mass

The specific location of the rigid body reference node relative to the rest of the rigid body or to its center of mass is important if nonzero boundary conditions are to be applied to the rigid body or concentrated loads are to be applied at the reference node. In many problems of rigid body dynamics, it might be desirable to apply loads and boundary conditions to the rigid body at its center of mass. In addition, it might be useful to monitor the configuration of the rigid body at its center of mass for output purposes. However, it might be difficult to locate the center of mass a priori when the rigid body mass and inertia properties (discussed below) contain contributions from a finite element discretization or a complex arrangement of MASS and ROTARYI elements.

By default, the rigid body reference node will not be repositioned. You can specify that it should be repositioned at the calculated center of mass. In this case if a MASS element is defined at the rigid body reference node, the calculated center of mass used for repositioning includes all mass contributions except the mass at the reference node. The MASS element is then repositioned at the center of mass and included in the mass properties of the rigid body. If the only mass contribution to the rigid body is from a MASS element defined at the rigid body reference node, the reference node will not be repositioned.

Input File Usage: Use the following option to indicate that the reference node should not be repositioned (the default):

**RIGID BODY*, REF NODE=*n*, POSITION=INPUT

Use the following option to specify that the rigid body reference node should be repositioned at the calculated center of mass:

**RIGID BODY*, REF NODE=*n*, POSITION=CENTER OF MASS

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Rigid body**: toggle **Adjust point to center of mass at start of analysis**

The Collection of Nodes that Constitute the Rigid Body

In addition to the rigid body reference node, rigid bodies consist of a collection of nodes that is either generated by assigning elements or associated directly with the rigid body. While both the translational and rotational degrees of freedom are active at the reference node, the active degrees of freedom at the remaining nodes of the rigid body depend on the element that they are part of.

The directly assigned nodes are typically part of deformable elements in the proximity, and their inclusion in the rigid body connects the deformable region to the rigid body. These nodes are one of two types:

- pin nodes, which have only translational degrees of freedom associated with the rigid body, or
- tie nodes, which have both translational and rotational degrees of freedom associated with the rigid body.

The above node types have important implications when the node is connected to rotary inertia elements, deformable structural elements, or connector elements or when the node has concentrated moments or follower loads applied to it. Rotary inertia elements and applied concentrated moments affect the rigid body only when associated with a tie node. Rigid body connections to deformable elements always involve the translational degrees of freedom; rigid body connections to deformable shell, beam, pipe, and connector elements also involve the rotational degrees of freedom if the connection is at a tie node. The behavior of the two types of connections is illustrated in *Figure 1*, which shows an octagonal rigid body connected to two deformable shell elements through nodes at opposite ends subjected to an applied rotational velocity.

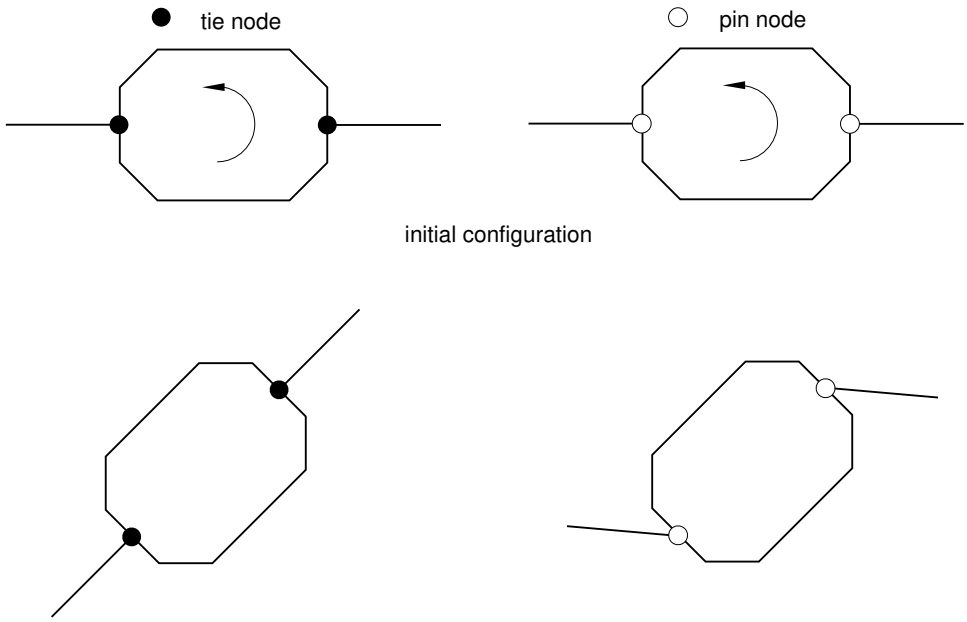


Figure 1: Rigid body with tie node and pin node connections.

The shell elements are shown with negligible bending in the figure above. When the nodes common to the rigid body and the shell elements are tie nodes, the rotation applied to the rigid body is transmitted directly to the shell elements. When the common nodes are pin nodes, the rigid body rotation is not transmitted directly to the shell elements, which can result in large relative motions between the rigid body and the adjacent shell structure.

Assigning Elements to a Rigid Body

To include elements in the rigid body definition, you specify the region of your model containing all the elements that are part of the rigid body. Elements in this region or nodes connected to the elements in this region cannot be part of any other rigid body. *Table 1* lists the continuum, structural, and rigid element types that can be included in a rigid body.

Table 1: List of valid elements that can be included in a rigid body (* indicates all elements beginning with the preceding label).

Rigid Body Geometry	Elements	
	Generate Pin Nodes	Generate Tie Nodes
Planar	CPE3*, CPE4*, CPE6*, CPE8*, CPS3, CPS4*, CPS6*, CPS8*, GK2D2, GKPS*, GKPE*, R2D2, T2D2*	B21*, B22*, B23*, FRAME2D, PIPE2*, RB2D2
Axisymmetric	CAX3, CAX4*, CAX6*, CAX8*, GKAX*, MAX*, RAX2	CGAX*, MGAX*, SAX1, SAX2*
Three-dimensional	C3D4*, C3D6*, C3D8*, C3D10*, C3D15*, C3D20*, C3D27*, GK3D*, M3D3, M3D4*, M3D6, M3D8*, M3D9*, SFM3D*, SFMAX*, SFMGAX*, R3D3, R3D4, T3D2*, CCL*, MCL*, SFMCL*	B31*, B32*, B33*, FRAME3D, PIPE*, RB3D2, S3*, S4*, S8*, S9*

The degrees of freedom active at the nodes of these elements depend on the element type (the same way they would if the elements were not included in the rigid body). If a connector element is part of a rigid body, the

active degrees of freedom at the corresponding nodes further depend on the connection type. In addition, if connector elements that activate material flow degree of freedom at nodes are included in the rigid body, the material and flow through the rigid body as that degree of freedom is constrained to the motion of the rigid body.

The following elements cannot be declared as rigid:

- Acoustic elements
- Axisymmetric-asymmetric continuum and shell elements
- Coupled thermal-electrical elements
- Diffusive heat transfer/mass diffusion elements and forced convection/diffusion elements
- Eulerian elements
- Discrete elements
- Generalized plane strain elements
- Gasket elements with thickness-direction behavior
- Heat capacitance elements
- Inertial elements (mass and rotary inertia)
- Infinite elements
- Piezoelectric elements
- Special-purpose elements
- Substructures
- Thermal-electrical-structural elements
- User-defined elements

If elements of more than one type or section definition are part of a rigid body, the specified region will contain elements with different section definitions. When continuum or structural elements are assigned to a rigid body, they are no longer deformable and their motion is governed by the motion of the rigid body reference node. Element stiffness calculations are not performed for these elements, and they do not affect the global time increment in Abaqus/Explicit. However, the mass and inertia of the rigid body includes contributions from these elements as calculated from their section and material density definitions (see *About the Element Library*). Mass and rotary inertia elements, as well as point heat capacitance elements, should not be included in the specified region. Contributions to a rigid body from mass, rotary inertia, and heat capacitance elements are accounted for automatically when these elements are connected to nodes that are part of the rigid body.

A list of nodes that are part of a rigid body is generated automatically when you assign elements to a rigid body. The node list is constructed as a unique list including all the nodes that are connected to elements in the specified region. Nodes in this list cannot be part of any other rigid body. The type of each node, pin, or tie is determined by the type of elements on the rigid body to which it is connected. Shell, beam, pipe, and rigid beam elements generate tie nodes; solid, membrane, truss, and rigid (other than beam) elements generate pin nodes (see *Table 1*). For nodes that are connected to both elements that generate pin nodes and elements that generate tie nodes, the common node is defined as the tie type.

All elements that are part of a rigid body must be of like geometry. Therefore, elements contained in the specified region must be either planar, axisymmetric, or three-dimensional. The geometry of the elements determines the geometry of the rigid body as shown in *Table 1*.

Input File Usage: Use the following option to assign elements to a rigid body:

**RIGID BODY*, REF NODE=*n*, ELSET=*name*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Rigid body**: **Body (elements)**: **Edit**: select body regions

Assigning Nodes to a Rigid Body

To assign nodes directly to a rigid body, you specify all the desired pin nodes and all the tie nodes separately. These nodes become part of the rigid body in addition to any nodes that have been generated from elements assigned to the rigid body. The following rules apply when assigning nodes directly to a rigid body:

- The rigid body reference node cannot be contained in either the set of pin nodes or the set of tie nodes.
- Nodes that are part of the set of pin nodes cannot also be contained in the set of tie nodes.
- Nodes that are contained in the set of pin nodes or the set of tie nodes cannot be part of any other rigid body definition.
- Nodes that are generated automatically from elements assigned to the rigid body that are also contained in the set of pin nodes are classified as pin nodes, regardless of their element connections.
- Nodes that are generated automatically from elements assigned to the rigid body that are also contained in the set of tie nodes are classified as tie nodes, regardless of their element connections.

The types of nodes generated by elements included in a rigid body can, therefore, be overridden by assigning the nodes directly to the rigid body, thereby allowing you greater flexibility to define a constraint with a rigid body by easily specifying the type of connection the rigid body makes with its attached deformable finite elements.

Input File Usage: Use the following option to assign nodes to a rigid body:

**RIGID BODY*, REF NODE=*n*, PIN NSET=*name*, TIE NSET=*name*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Rigid body**: **Pin (nodes)**: **Edit**: select pin regions, and **Tie (nodes)**: **Edit**: select tie regions

Assigning Analytical Surfaces to a Rigid Body

You can assign an analytical surface to a rigid body. The procedure for creating and naming an analytical rigid surface is described in *Analytical Rigid Surface Definition*. Only one analytical surface can be defined as part of the rigid body definition.

Input File Usage: Use the following option to assign an analytical rigid surface to a rigid body:

**RIGID BODY*, REF NODE=*n* or *name*, ANALYTICAL SURFACE=*name*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Rigid body**: **Analytical Surface**: **Edit**: select analytical surface regions

Defining a Rigid Body in a Model That Is Defined in Terms of an Assembly of Part Instances

An Abaqus model can be defined in terms of an assembly of part instances (see *Assembly Definition*). A rigid body in such a model can be created from deformable elements at either the part level or the assembly level. In either case all node and element definitions must belong to one or more parts. If all nodes making up the rigid body belong to the same part, create a rigid part by defining the rigid body at the part level.

Multiple deformable part instances can be combined into a single rigid body by creating an assembly-level node or element set that spans the part instances, then defining the rigid body at the assembly level to refer to that set. The rigid body reference node can also be defined at the assembly level, if necessary.

Rigid Body Mass and Inertial Properties

When a rigid body is not constrained fully, the mass and inertia properties of the rigid body are important to its dynamic response. In Abaqus/Explicit an error message will be issued if there is no mass (or rotary inertia) corresponding to an unconstrained degree of freedom. Abaqus automatically calculates the mass, center of mass, and rotary inertia of each rigid body and prints the results to the data (`.dat`) file if model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). The following rules are used to determine the mass and inertia of a rigid body:

- The mass of each continuum, structural, and rigid element that is part of the rigid body contributes to the rigid body's mass, center of mass, and rotary inertia properties.
- Point mass elements that are connected to any node that is part of a rigid body or to the rigid body reference node contribute to the rigid body's mass, center of mass, and rotary inertia properties.
- Rotary inertia elements that are connected to any tie node or to the rigid body reference node contribute to the rigid body's rotary inertia properties.

Since the rotational degrees of freedom at a pin node are not part of a rigid body, rotary inertia elements connected to a pin node do not contribute to the rigid body inertia but are rather associated with the independent rotation of the node.

Defining Mass and Inertia Properties by Discretization

In many cases it is desirable to model rigid components for which the mass, center of mass, and rotary inertia are not readily available. In Abaqus it is not necessary to define the mass and inertia properties of the rigid body directly. Instead, a finite element discretization can be used to model the rigid components, and Abaqus will automatically calculate the properties from the discretization. Rigid structures with one-dimensional rod or beam geometries can be modeled with beam or truss elements, structures containing two-dimensional surface geometries can be modeled with shell or membrane elements, and solid geometries can be modeled with solid elements. The mass contributions to the rigid body for each of these elements are based on that element's section properties (see *About the Element Library*) and the material density (see *Density*). Although both shell and membrane elements in a rigid body can yield similar mass contributions given similar section and density definitions, they will generate different node types (tie nodes for shells and pin nodes for membranes), which might affect the overall results. The same holds true for beam and truss elements.

In situations where one portion of a rigid component can be modeled with a finite element discretization but it is not convenient to do so for other portions, point mass and rotary inertia elements can be used to represent the mass distribution of these other portions. The mass, center of mass, and rotary inertia for the rigid body will then include the contributions from both the finite elements and the point mass and rotary inertia elements.

Abaqus uses the lumped mass formulation for low-order elements. As a consequence, the second mass moments of inertia can deviate from the theoretical values, especially for coarse meshes. This inaccuracy can be circumvented by adding point mass and rotary inertia elements with the correct inertia properties and eliminating the mass contribution from the solid elements. Alternatively, second-order elements could be used in Abaqus/Standard.

Defining Mass and Inertia Properties Directly

When the mass, center of mass, and rotary inertia properties of the actual rigid component are known or can be approximated, it is not necessary to use a finite element discretization or to use an array of point masses to generate the rigid body properties. You can assign these properties directly by locating the rigid body reference node at the center of mass (see *Positioning the Reference Node at the Center of Mass*) and by specifying the rigid body mass and rotary inertia at the reference node (see *Point Masses* and *Rotary Inertia*).

It might also be desirable to input mass properties directly at the center of mass but to specify boundary conditions at a location other than the center of mass. In this case you should place the rigid body reference node at the desired boundary condition location. In addition, you must define a tie node at the center of mass of the rigid body by correctly specifying its coordinates to coincide with the coordinates of the center of mass of the rigid body and then assigning it to a tie node set in the rigid body definition. You can then define the rigid body mass and rotary inertia at the tie node.

For most applications where mass properties are input directly, it might be necessary to assign additional elements or nodes to a rigid body so that the rigid body can interact with the rest of the model. For example, contact pair definitions could require rigid surfaces formed with element faces on the rigid body and additional pin or tie nodes might be necessary to provide the desired constraints with deformable elements attached to the rigid body. Abaqus will account for the mass and rotary inertia contributions from all elements on a rigid body; therefore, if you want to assign the rigid body mass properties directly, you should take care to ensure that contributions from other element types that are part of the rigid body do not affect the desired input mass properties. If rigid elements are part of the rigid body definition, you can set their mass contribution to zero by not specifying a density for these elements in the rigid body definition. If other element types are used to define the rigid body, you should set their density to zero.

Kinematics of a Rigid Body

The motion of a rigid body is defined entirely by the motion of its reference node. The active degrees of freedom at the reference node depend on the geometry of the rigid body (see *Conventions*). The geometry of a rigid body is planar, axisymmetric, or three-dimensional and is determined by the type of elements that are assigned to the rigid body. In the case where no elements are assigned to a rigid body, the geometry of the rigid body is assumed to be three-dimensional.

The calculated mass and rotary inertia properties for each of the active degrees of freedom for all rigid bodies are printed to the data (`.dat`) file if model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). These properties include the contributions from elements that are part of the rigid body, as well as point mass and rotary inertia elements at the nodes of the rigid body.

Although this calculated mass represents the true mass of the rigid body, Abaqus/Explicit actually uses an augmented mass in the integration of the equation of motion, which is conceptually similar to an added mass formulation. Essentially, the calculated mass and rotary inertia of the rigid body is augmented with the mass contributions of all of its attached deformable elements to create a larger, augmented mass and rotary inertia. Rotary inertia contributions from adjacent deformable elements are also included in the augmented rotary inertia if the nodal connection is at a tie node.

Rigid Body Motions

A rigid body can undergo free rigid body motion in each of its active translational degrees of freedom, as well as each of its active rotational degrees of freedom.

Boundary Conditions

Boundary conditions for rigid bodies should be defined as described in *Boundary Conditions* at the rigid body reference node. Reaction forces and moments can be recovered for all degrees of freedom that are constrained at the reference node. If a nodal transformation is defined at the rigid body reference node, boundary conditions are applied in the local system (see *Transformed Coordinate Systems*).

In Abaqus/Standard, if boundary conditions are applied to any nodes on a rigid body other than the rigid body reference node, Abaqus will attempt to transfer these boundary conditions to the reference node. If successful, you are warned that this transfer has taken place. Otherwise, an error message is produced (see *Overconstraint Checks* for more details).

In Abaqus/Explicit, if boundary conditions are applied to any nodes on a rigid body other than the rigid body reference node, these boundary conditions are ignored with the exception of the symmetry-type boundary conditions that can affect the contact logic at the perimeter of a surface in the Abaqus/Explicit contact pair algorithm (see *Contact Formulations for Contact Pairs in Abaqus/Explicit* and *Common Difficulties Associated with Contact Modeling Using Contact Pairs in Abaqus/Explicit*).

Constraints

In Abaqus/Standard nodes on a rigid body, excluding the rigid body reference node, cannot be used in a multi-point constraint or linear constraint equation definition.

In Abaqus/Explicit a multi-point constraint or linear constraint equation can be defined for any node on a rigid body, including the reference node.

Connector Elements

Connector elements can be used at any node of a rigid body, including the reference node, to define a connection between rigid bodies, between a rigid body and a deformable body, or from a rigid body to ground. Connector elements are convenient for providing multiple points of attachment on rigid bodies; modeling complex nonlinear kinematic constraints; specifying zero or nonzero boundary conditions at a point on a rigid body that is not the rigid body reference node; applying force actuation; and modeling discrete interactions, such as spring, dashpot, node-to-node contact, friction, locking mechanisms, and failure joints. Unlike multi-point constraints or linear constraint equations, connector elements retain degrees of freedom in the connection, thereby allowing output of information related to the connection (such as constraint forces and moments, relative displacements, velocities, accelerations, etc.). See *Connector Elements* for a detailed description of connector elements.

Planar Rigid Body

A rigid body with a planar geometry has three active degrees of freedom: 1, 2, and 6 (u_x, u_y, and ϕ_z). Here, the *x*- and *y*-directions coincide with the global *X*- and *Y*-directions, respectively. If a nodal transformation is defined at the rigid body reference node, the *x*- and *y*-directions coincide with the user-defined local directions. The coordinate transformation defined at the reference node must be consistent with the geometry; the local directions must remain in the global *X*–*Y* plane. All nodes and elements that are part of a planar rigid body should lie in the global *X*–*Y* plane.

Planar rigid bodies should be connected only to planar deformable elements. To model the connection of a rigid component with a planar geometry to three-dimensional deformable elements, model the planar rigid component as a three-dimensional rigid body consisting of the appropriate three-dimensional elements.

Axisymmetric Rigid Body

A rigid body with an axisymmetric geometry has three active degrees of freedom in Abaqus: 1, 2, and 6 (u_r, u_z, ϕ). Classical axisymmetric theory admits only one rigid body mode, which is displacement in the *z*-direction. To maximize the flexibility of using rigid bodies for axisymmetric analysis, Abaqus allows for three active degrees of freedom, although only the axial displacement is a rigid body mode.

The *r*- and *z*-directions coincide with the global *X*- and *Y*-directions, respectively. If a nodal transformation is defined at the rigid body reference node, the *r*- and *z*-directions coincide with the user-defined local directions. The coordinate transformation defined at the reference node must be consistent with the geometry; the local directions must remain in the global *X*–*Y* plane. All nodes and elements that are part of an axisymmetric rigid body should lie in the global *X*–*Y* plane.

Translation in the *r*-direction is associated with a radial mode, and rotation in the *r*–*z* plane is associated with a rotary mode (see *Figure 2*). For an axisymmetric rigid body in Abaqus each of these modes develop no hoop

stress, but mass and inertia computed for these degrees of freedom represent the modal mass associated with their modal motion. The mass properties for an axisymmetric rigid body are, therefore, calculated based on the initial configuration assuming the following:

- Point masses defined on nodes of the rigid body (see *Point Masses*) are assumed to account for the entire mass around the circumference of the body.
- Mass contributions from axisymmetric elements assigned to the rigid body include the integrated value around the circumference.
- The center of mass of the rigid body is located at the center of mass of the circumferential slice, as shown in *Figure 2*.

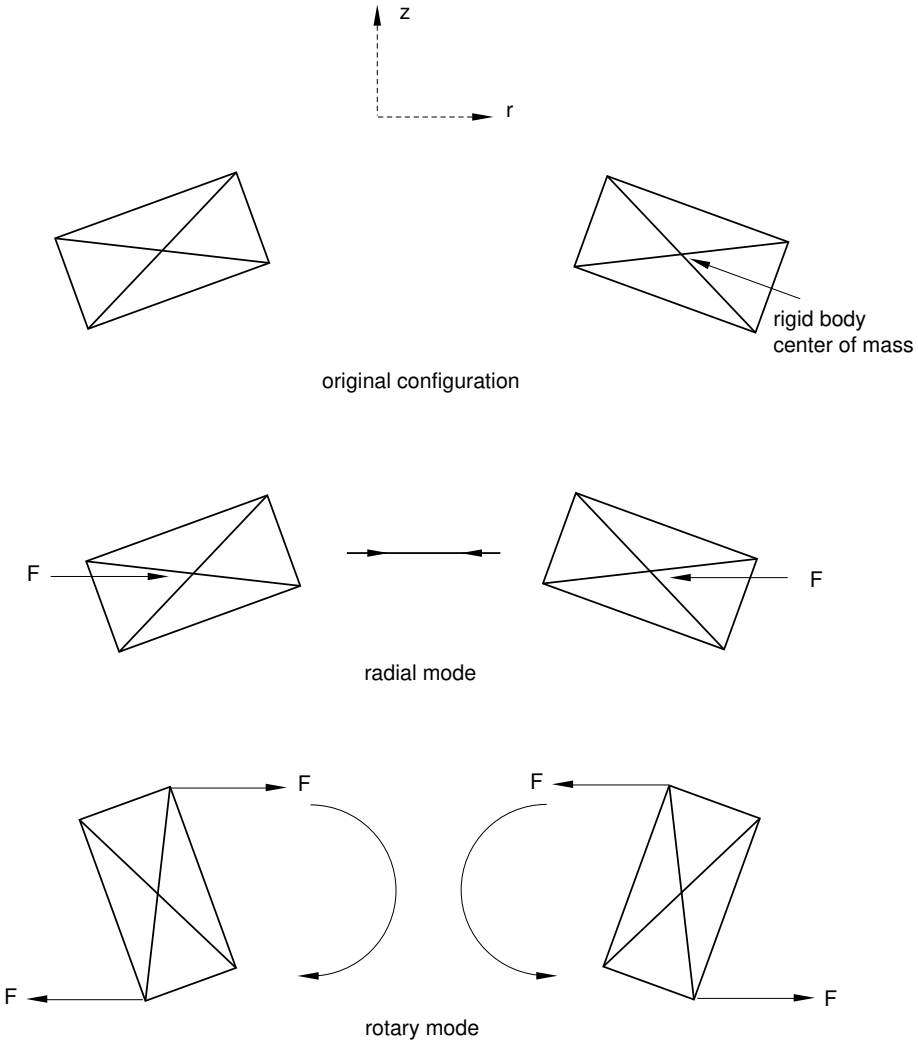


Figure 2: Axisymmetric rigid body modes.

If the rigid body reference node is positioned at the center of mass, the reference node for an axisymmetric rigid body will, thus, be repositioned at the center of mass of the circumferential slice.

These assumptions are consistent with the manner in which Abaqus handles other axisymmetric features but are noted here because of the deviation from classical rigid body theory.

Axisymmetric rigid bodies should be connected only to axisymmetric deformable elements. To model the connection of a rigid component with an axisymmetric geometry to three-dimensional deformable elements,

model the axisymmetric rigid component as a three-dimensional rigid body consisting of the appropriate three-dimensional elements.

Three-Dimensional Rigid Body

A rigid body with a three-dimensional geometry has six active degrees of freedom: 1, 2, 3, 4, 5, and 6 (u_x, u_y, u_z, ϕ_x, ϕ_y, ϕ_z). Here, the *x*-, *y*-, and *z*-directions coincide with the global *X*-, *Y*- and *Z*-directions, respectively. If a nodal transformation is defined at the rigid body reference node, the *x*-, *y*-, and *z*-directions coincide with the user-defined local directions.

In general, three-dimensional rigid bodies will possess a full, nonisotropic inertia tensor and can behave in a nonintuitive manner when they are spun about an axis that is not one of the principal inertia axes. Classical phenomena of rigid body dynamics (for example, precession, gyroscopic moments, etc.) can be simulated using three-dimensional rigid bodies in Abaqus.

In most cases three-dimensional rigid bodies should be connected only to three-dimensional deformable elements. If it is physically relevant, a three-dimensional rigid body can be connected to two-dimensional plane stress, plane strain, or axisymmetric elements; however, you should always constrain the *z*-displacement, *x*-axis rotation, and *y*-axis rotation of the rigid body. The above procedure is useful when incorporating a two-dimensional plane strain approximation in one region of a model and a three-dimensional discretization in another. Rigid bodies can be used to constrain the two finite element geometries at their interface as shown in *Figure 3*. A unique rigid body should be used at each node in the plane along the interface to handle the constraint properly.

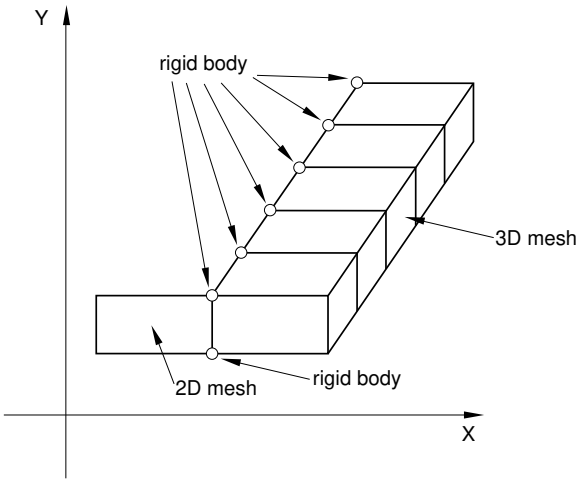


Figure 3: Rigid body nodes used to connect a two-dimensional and three-dimensional mesh.

Defining Loads on Rigid Bodies

Loads on a rigid body are assembled from contributions of all the loads on nodes and elements that are part of the rigid body. Loads are defined on nodes and elements that are part of a rigid body in the same manner that they are specified if the nodes and elements are not part of a rigid body. Contributions include:

- applied concentrated forces on pin nodes, tie nodes, and the rigid body reference node;
- applied concentrated moments on tie nodes and the rigid body reference node; and
- applied distributed loads on all elements and surfaces that are part of the rigid body.

Unless the point of action is through the rigid body center of mass, each of these loads will create both a force at and a torque about the center of mass, which will tend to rotate an unconstrained rigid body. If a nodal

transformation is defined at any rigid body nodes, concentrated loads defined at these nodes are interpreted in the local system. The local system defined by the nodal transformation does not rotate with the rigid body.

Concentrated moments defined on rigid body pin nodes do not contribute load to the rigid body but are rather associated with the independent rotation of that node. Independent rotation of a pin node exists only if it is connected to a deformable element with rotational degrees of freedom or a rotary inertia element. Follower forces (see *Specifying Concentrated Follower Forces*) can be defined at pin nodes if the independent rotation exists. However, the results might be nonintuitive since the direction of the force is determined by the independent rotation even though the follower force acts on the rigid body.

Rigid Bodies with Temperature Degrees of Freedom

Only rigid bodies that contain coupled temperature-displacement elements have temperature degrees of freedom. If it is reasonable to assume that a rigid body used in a fully coupled temperature-displacement analysis has a uniform temperature, you can define the rigid body as isothermal. A transient heat transfer process involving an isothermal rigid body assumes that the internal resistance of the body to heat is negligible in comparison with the external resistance. Thus, the body temperature can be a function of time but cannot be a function of position. The temperature degree of freedom that is created at the rigid body reference node describes the temperature of the body.

Thermal interactions for rigid bodies with analytical rigid surfaces are available only in Abaqus/Explicit and are activated by specifying that the rigid body is isothermal.

By default, rigid bodies are not considered isothermal and all nodes on a rigid body connected to coupled temperature-displacement elements will have independent temperature degrees of freedom. The fact that the nodes are part of a rigid body does not affect the ability of the coupled elements to conduct heat within the rigid body. However, the mechanical response will be rigid.

The lumped heat capacitance associated with the rigid body reference node of an isothermal body is calculated automatically if the rigid body is composed of coupled temperature-displacement elements for which a specific heat and a density property are defined. Otherwise, you should specify a point heat capacitance for the rigid body (see *Point Capacitance*). An error message will be issued in Abaqus/Explicit if no heat capacitance is associated with an isothermal rigid body for which temperature is not prescribed at the reference node.

- The capacitance of each coupled temperature-displacement element that is part of the rigid body contributes to the isothermal rigid body's capacitance. For an axisymmetric isothermal rigid body, capacitance contributions from axisymmetric elements assigned to the rigid body include the integrated value around the circumference.
- HEATCAP elements that are connected to any node that is part of a rigid body or the rigid body reference node contribute to the isothermal rigid body's capacitance. For an axisymmetric isothermal rigid body the point capacitances defined on nodes of the rigid body are assumed to account for the capacitance integrated around the circumference of the body.

Thermal loads acting on the reference node of an isothermal body are assembled from contributions of all the thermal loads on nodes and elements that are part of the rigid body. Heat transfer between a deformable body and an isothermal rigid body can occur during contact, as well as when the bodies are not in contact if gap conductance and gap radiation are defined (see *Thermal Contact Properties*). Heat transfer between two isothermal rigid bodies can occur only via gap conduction and gap radiation.

Input File Usage: **RIGID BODY*, ISOTHERMAL=YES

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Rigid body**: toggle on **Constrain selected regions to be isothermal**

Thermal Expansion of a Rigid Body

In Abaqus/Standard a rigid body can experience expansion due to a temperature increase. Each rigid body node expands along the line that joins the node to the rigid body reference node. The magnitude of the expansion depends on the distance of the node from the reference node. The temperature change for computing the expansion is the average of the temperature change at the node and the temperature change at the reference node. The temperature change at any node is the difference between the initial temperature of the node and the current temperature of the node. You must provide the value of the thermal expansion coefficient so that Abaqus/Standard can compute the expansion. The expansion coefficient of any material that is part of the rigid body is neglected. Thermal expansion can be used only when temperature is a field variable.

Input File Usage: **RIGID BODY*, ALPHA=*thermal_expansion_coeff*

Abaqus/CAE Usage: Thermal expansion of a rigid body is not supported in Abaqus/CAE.

Modeling Contact with a Rigid Body

Contact with a rigid body is modeled by specifying a contact interaction formed with a rigid surface and with a surface defined on another body (see *About Contact Pairs in Abaqus/Standard*, *About General Contact in Abaqus/Explicit*, or *About Contact Pairs in Abaqus/Explicit*). A rigid surface can be formed by nodes, element faces, or an analytical surface (see *Node-Based Surface Definition*, *Element-Based Surface Definition*, and *Analytical Rigid Surface Definition*).

Contact modeling can be a primary factor when choosing the appropriate rigid body geometry. Contact interactions should be formed with surfaces of like geometry. For example, a planar rigid body should be used to model contact either with deformable surfaces formed by two-dimensional plane stress or plane strain elements or via node-based surfaces with two-dimensional beam, pipe, or truss elements. Similarly, an axisymmetric rigid body should be used to model contact with surfaces formed by axisymmetric elements, and a three-dimensional rigid body should be used to model contact either with surfaces formed by three-dimensional element faces or via node-based surfaces with three-dimensional beam, pipe, or truss elements.

A rigid body must contain only two-dimensional or only three-dimensional elements. Nodes cannot be shared between two rigid bodies. Contact between two analytical rigid surfaces or between an analytical rigid surface and itself cannot be modeled.

Limitations in Abaqus/Standard

Contact between rigid bodies is allowed if the secondary surface belongs to an elastic body that has been declared as rigid. In this case softened contact should be prescribed to avoid possible overconstraints.

Contact between two rigid surfaces defined using rigid elements is not allowed.

Rigid beams and trusses cannot be included in a contact pair definition because surfaces from beams and trusses can be node-based surfaces only. A node-based surface must be a secondary surface, and elements that are part of a rigid body should be part of the main surface in a contact pair.

Limitations in Abaqus/Explicit

Contact between two rigid surfaces can be modeled in Abaqus/Explicit only if the penalty contact pair algorithm or the general contact algorithm is used; kinematic contact pairs cannot be used for rigid-to-rigid contact. Therefore, when converting two deformable regions of a model to two distinct rigid bodies for the purpose of model development, any contact interaction definitions between these rigid bodies must use penalty contact pairs or general contact.

For rigid-to-rigid contact involving analytical rigid surfaces, at least one of the rigid surfaces must be formed by element faces since contact between two analytical rigid surfaces cannot be modeled in Abaqus.

The penalty contact pair algorithm, which introduces numerical softening to the contact enforcement through the use of penalty springs, or the general contact algorithm must be used for all contact interactions involving a rigid body if an equation constraint, a multi-point constraint, a tie constraint, or a connector element is defined for a node on the rigid body.

Rigid beams and trusses cannot be included in a kinematic contact pair definition because surfaces from beams and trusses can be node-based surfaces only. A node-based surface must be a secondary surface, and elements that are part of a rigid body must be part of the main surface in a kinematic contact pair.

When a rigid surface acts as a secondary surface in a penalty contact pair or in general contact, initial penetrations of the rigid secondary nodes into the main surface will not be corrected with strain-free corrections (see *Contact Initialization for Contact Pairs in Abaqus/Explicit* and *Contact Initialization for General Contact in Abaqus/Explicit*). For contact pairs any initial penetrations of this type might cause artificially large contact forces in the initial increments. For general contact these initial penetrations are stored as contact offsets.

The motion of the rigid body is entirely defined by the motion of its reference node; therefore, each tie node of the rigid body undergoes the same rotation as the reference node. The output of the rotation at the rigid body nodes is not available except at the reference node.

Using Rigid Bodies in Geometrically Linear Abaqus/Standard Analysis

If rigid bodies are used in a geometrically linear Abaqus/Standard analysis (see *General and Perturbation Procedures*), the rigid body constraints are linearized. Consequently, except for analytical rigid surfaces, the distance between any two nodes belonging to the rigid body might not remain constant during the analysis if the magnitudes of the rotations are not small.

Integrated Output Section Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Output to the Output Database*
- **INTEGRATED OUTPUT SECTION*
- **INTEGRATED OUTPUT*
- **SURFACE*
- *Defining integrated output sections*

Overview

An integrated output section:

- can be two-dimensional or three-dimensional;
- can be used to track the average motion of a surface;
- can be used in association with integrated output requests to study the “force-flow” in the model; and
- does not impose any constraint on the motion of the surface.

Introduction

An integrated output section is a way to associate a surface with a coordinate system and/or a reference node for one or both of the following purposes:

- tracking the average motion of the surface; and/or
- expressing the force and the moment transmitted through the surface in a local coordinate system, with the moment taken about a point that moves with the surface.

The average motion of a surface can be obtained as the displacement and/or rotation history at the reference node on an integrated output section definition. You must define a reference node that is not connected to any other part of the finite element model and select whether the reference node follows only the average translation of the surface or both the translation and the rotation. Since the reference node is not connected to the rest of the model, an integrated output section definition used to track the average surface motion does not form a constraint on the motion of any nodes in the model.

The “force-flow” in a complicated model can be studied using integrated output sections defined over a number of interior cross-section-like surfaces cutting through various parts of the model. It can be equally useful to sum forces over an exterior surface in contact or to sum forces transmitted through a tie constraint between surfaces, which is done by associating an integrated output section definition with an integrated output request. The vector output quantities can be expressed in a coordinate system of choice by specifying an orientation on an integrated output section definition. This coordinate system can rotate by an amount given by the rotational degrees of freedom at the reference node. In addition, the output of the integrated moment across the surface can be taken about a location that can translate by an amount given by the translational degrees of freedom at the reference node. Integrated output over a given surface can be requested with different coordinate systems and reference nodes by employing multiple integrated output section definitions over the same surface.

Creating an Integrated Output Section

You must assign a name to each integrated output section. This name is used to associate the section with an integrated output request. In addition, you must identify the surface over which the section is being defined (see *Element-Based Surface Definition*).

Input File Usage: **INTEGRATED OUTPUT SECTION*, NAME=*section_name*, SURFACE=*surface_name*

Abaqus/CAE Usage: Step module: **Output->Integrated Output Sections->Create**: **Name:** *section_name*: select surface region

Creating Interior Cross-Section Surfaces

To study the "force-flow" through various paths in a model, you must create interior single-sided surfaces that cut through one or more regions (similar to a cross-section) so that you can request integrated output of the total force and moment transmitted across these surfaces. You can create such interior surfaces over the element facets, edges, or ends by simply cutting through one or more regions of the model with a plane; see *Creating Interior Cross-Section Surfaces* for more information.

The Integrated Output Section Reference Node

A reference node can be associated with an integrated output section for one or both of the following purposes:

- tracking the average motion of the surface; and/or
- computing the variables from an integrated output request in a coordinate system that moves with the motion of the reference node.

If the average surface motion must be tracked, you must define a reference node that is not connected to any other part of the finite element model and select whether the reference node follows only the average translation of the surface or both the translation and the rotation. The rotational degrees of freedom will be activated in addition to the translational degrees of freedom at the reference node if it is selected to follow the average rotation of the surface. Further, the initial position of the reference node might be adjusted to lie at the center of the surface automatically.

When an integrated output section with a reference node is associated with an integrated output request, the total moment transmitted through the section is computed with respect to the current location of the reference node. If the reference node has active rotational degrees of freedom, the coordinate system used to express the integrated output variables rotates with the rotation of the reference node.

Positioning the Reference Node at the Center of the Surface

The reference node can be repositioned automatically at the center of the surface in the initial configuration when the reference node is not connected to the rest of the model.

The default is to leave the reference node in its specified position.

Input File Usage: Use the following option to position the reference node at the center of the surface:

**INTEGRATED OUTPUT SECTION*, REF NODE=*n*, POSITION=CENTER

Abaqus/CAE Usage: Step module: integrated output section editor: **Anchor at reference point: Edit**: select reference point: **Move point to center of surface**

Setting the Reference Node to Track the Average Motion of the Surface

It is often meaningful to obtain integrated output over a surface using a coordinate system and a point that moves with the average surface motion. When the reference node is not connected to the rest of the model, it can be specified to translate with the average translation of the surface without any rotation or to both translate and rotate with the average motion of the surface. The average motion is based on the mass weighted motion of the individual nodes that are on the surface and are not part of any rigid body.

By default, the reference node does not track the average motion of the surface.

Input File Usage: Use the following option if the reference node must translate with the average translation of the surface:

**INTEGRATED OUTPUT SECTION*, REF NODE=*n*,
REF NODE MOTION=AVERAGE TRANSLATION

Use the following option if the reference node must both translate and rotate with the average translation of the surface:

**INTEGRATED OUTPUT SECTION*, REF NODE=*n*,
REF NODE MOTION=AVERAGE

Abaqus/CAE Usage: Step module: integrated output section editor: **Anchor at reference point: Edit**: select reference point: **Point motion: Average translation and rotation** or **Average translation**

The Integrated Output Section Local Coordinate System

You can define a local coordinate system on an integrated output section and associate the section with an integrated output request to express the integrated output variables in the local coordinate system. You can specify an orientation as the local coordinate system and, possibly, further project it onto the surface. Alternatively, you can form a local coordinate system by projecting the global coordinate system onto the surface following the Abaqus conventions (see *Conventions*). If a local system is not defined explicitly, the local system is initialized to the global coordinate system.

The initial coordinate system, whether explicitly defined or initialized to the global coordinate system, will rotate with the deformation if a reference node is specified and that reference node has active rotational degrees of freedom. If the reference node is not connected to the rest of the model and its motion is based on both the average translation and rotation of the surface, the rotational and translational degrees of freedom are activated at the reference node.

Input File Usage: Use the following option to define the initial coordinate system for the section:

**INTEGRATED OUTPUT SECTION*, ORIENTATION=*orientation_name*

Abaqus/CAE Usage: Step module: integrated output section editor: **CSYS: Edit**: select orientation

Projecting the Coordinate System onto the Section Surface

Either the coordinate system defined by the specified orientation or the global coordinate system can be projected onto the section surface to obtain a local coordinate system. Projection onto the surface is based on the average normal of the surface; the local 1-direction is formed perpendicular to the surface (see *Figure 1*).

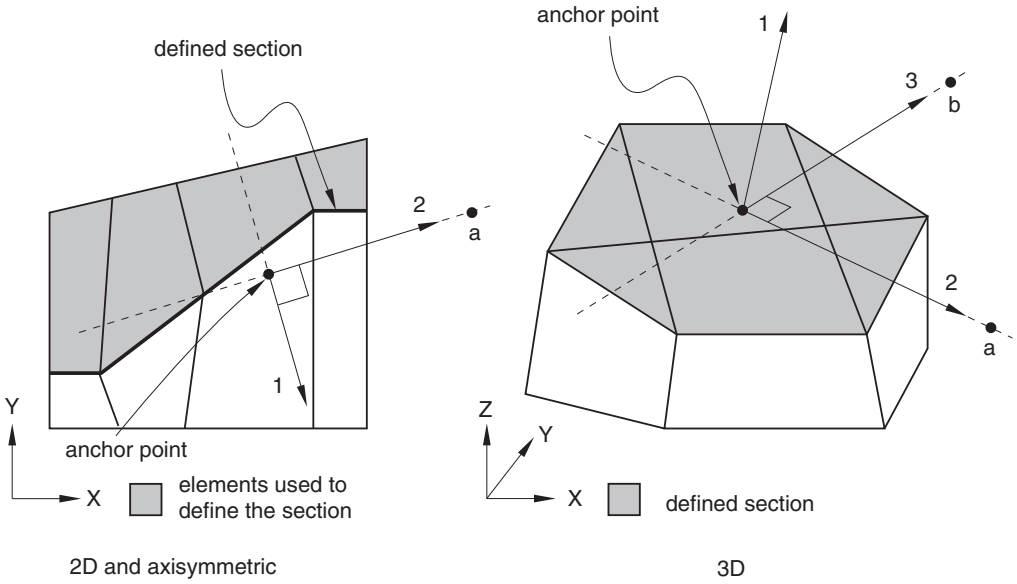


Figure 1: User-defined local coordinate system.

Input File Usage: Use the following option to project the coordinate system onto the section surface:

**INTEGRATED OUTPUT SECTION*, PROJECT ORIENTATION=YES

Abaqus/CAE Usage: Step module: integrated output section editor: **Project orientation onto surface**

Associating an Integrated Output Section with an Integrated Output Request

An integrated output request is used to obtain history output of variables such as total force transmitted across a surface (see *Integrated Output*). Such a request might refer to an integrated output section definition to identify the surface where output is needed and to provide a local coordinate system and/or a reference node as a point about which the total moment across the surface is computed.

Input File Usage: Use both of the following options to associate an integrated output section with an integrated output request:

**INTEGRATED OUTPUT SECTION*, NAME=*section_name*
**INTEGRATED OUTPUT*, SECTION=*section_name*

Abaqus/CAE Usage: Step module:

Output->Integrated Output Sections->Create: **Name:** *section_name*

History output request editor: **Domain: Integrated output section:** *section_name*

Limitations

Integrated output sections are subject to the following limitations:

- The surface associated with an integrated output section cannot be an analytical rigid surface.
- The surface associated with an integrated output section can contain facets over rigid or axisymmetric elements. However, such an integrated output section cannot be associated with an integrated output request (see *Output to the Output Database*).

Mass Adjustment

Products: Abaqus/Explicit

References:

- *Density*
- *Point Masses*
- *Nonstructural Mass Definition*
- *Mass Scaling*
- **MASS ADJUST*

Overview

Mass adjustment:

- is useful to set the net mass of one or more components in the model to a known value;
- is useful to scale the net mass of one or more components in the model (without the need to know the mass of the component);
- is useful to account for any errors in mass due to modeling approximations;
- is useful to account for mass from nonstructural features otherwise omitted from the model, such as paint;
- can be applied over all element types that have mass;
- adjusts the mass of the individual elements in an element set in proportion to their pre-adjusted mass including any nonstructural mass, so as to meet the specified target value for the set;
- can be used to redistribute mass among elements in the set to raise the minimum stable time increment to a target value;
- can be used to redistribute mass to maximize the stable time increment of the elements in the element set;
- can be specified only once in an Abaqus/Explicit analysis during the model definition; and
- can be applied in a hierarchical fashion to adjust the mass for individual parts first and then for an assembly of these parts.

Defining Mass Adjustment

To adjust or scale the total mass of one or more components in the model, you first identify the corresponding element sets. If you specify multiple elements sets, the mass is adjusted or scaled in the order in which the element sets are specified. For element sets that share elements, you must determine the order in which to specify the element sets to obtain the desired results.

Input File Usage: **MASS ADJUST*

Adjusting the Total Mass of an Element Set to a Known Value

The mass of a component in a numerical model may differ from its actual value for a number of reasons including modeling approximations and omission of minor features from the model. You can specify mass adjustment in

the numerical model for such components by identifying the element sets defining these components and their respective total mass values. For a given element set, the mass is adjusted at the start of the analysis such that the adjustment in each element in that set is in proportion to the pre-adjusted mass of that element, thus preserving the center of mass and the principal directions of the rotary inertia. The pre-adjusted mass of an element includes the mass due to any associated material density; any mass directly specified on the section definition as in the case of beam, pipe, shell, membrane, rigid, and surface elements; and any nonstructural mass applied directly to that element. *Knee bolster impact with general contact* is an example of setting the total mass of an element set using mass adjustment.

When mass is adjusted for an element with active rotational degrees of freedom, the rotary inertia contribution from that element is also modified proportionally to correspond with the scaling in the element mass from mass adjustment, thus preserving the principal directions of the rotary inertia. The adjusted mass value is considered when calculating the stable time increment of an element. Loads such as mass proportional damping (see *Material Damping*) and gravity take the adjusted mass into account.

Mass adjustment can be applied in a hierarchical fashion to adjust the mass for individual parts first and then for an assembly of these parts. In this scenario, the mass adjustment defined over the assembly may further modify the adjusted mass of the individual parts. You must associate all of the mass-adjusted element sets in the desired order with a single mass adjustment definition.

Abaqus/Explicit automatically calculates the mass, center of mass, and rotary inertia of each element set and prints the results to the data (`.dat`) file if model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). The contributions from mass adjustment are also listed in these tables. Element output variable MASSADJUST can be requested as output to the output database (`.odb`) file, and it will indicate how the mass of the set is adjusted or redistributed to each element included in the set (see *Abaqus/Explicit Output Variable Identifiers*). This output variable is available as field output (contour plots) in the first output frame of the first analysis step.

Mass adjustment contributions applied to an element set are always included when transferring model data between Abaqus/Explicit analyses (see *Transferring Results from One Abaqus/Explicit Analysis to Another*). There is no need to redefine these contributions in the import analysis unless different mass adjustment is required for the element set.

Input File Usage: Use the following option to define total mass for an element set without altering its center of mass:

**MASS ADJUST*
elem_set_name, elem_set_mass

Redistribution of Mass to Raise the Minimum Stable Time Increment to a Target Value

You can increase the minimum stable time increment in the initial configuration for an element set to a specified target value by redistributing mass among the elements in that set. The redistribution of mass to affect the stable time increment and adjustment of mass to achieve a target total mass can be requested independently of each other. If both options are requested for a given element set, the mass is first adjusted to meet the target total mass for the set and then redistributed among the elements to achieve the target time increment.

You can set a default target time increment that is applicable for all of the mass-adjusted element sets as well as specific targets for any of the individual element sets. Within each set, the mass is transferred to the elements with time increments below the target value from the remaining elements. Abaqus/Explicit prints the amount of mass available for redistribution along with the percentage of this amount that is redistributed to the data (`.dat`) file if model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). If a sufficient amount of mass is not available to meet the specified target time increment, the analysis terminates with an error message. *Impact of a water-filled bottle* is an example of maintaining the target stable time increment of an element set using mass adjustment.

When compared to the fixed mass scaling functionality, the redistribution feature above does not alter the total mass of the set. However, both features affect the center of mass and the principal directions of rotary inertia. The redistribution feature is performed only in the initial configuration at the start of the analysis; whereas the fixed mass scaling is performed in the configuration at the start of the step requesting that mass scaling. When you specify mass adjustment and mass scaling, the mass scaling adds mass as necessary on top of the adjusted mass.

Input File Usage: Use the following option to raise the time increment and specify the total mass:

**MASS ADJUST*, TARGET DT=*min_stable_time_increment*
elem_set_name, elem_set_mass, elem_set_min_stable_time_increment

Use the following option to raise the time increment without altering the total mass:

**MASS ADJUST*, TARGET DT=*min_stable_time_increment*
elem_set_name, CURRENT, *elem_set_min_stable_time_increment*

Redistribution of Mass Using a Scale Factor

An alternative method to adjust the mass of an element set is to specify a scale factor. When the mass adjustment is defined using a scale factor, you can maximize the minimum stable time increment in the initial configuration for an element set by redistributing either the total mass or only the added mass among the elements in that set. The redistribution of the total mass to maximize the time increment is done iteratively by taking mass from each element with a stable time increment greater than the average and adding it to the elements with lower time increments—this iterative process stops when all elements have the same time increment within a tolerance.

The redistribution of only the added mass to maximize the time increment is done in a similar fashion, but only the added mass is transferred between elements. The elements may not have the same time increment after the redistribution is complete. In this case the scale factor should be greater than one. Both choices of mass redistribution described here affect the center of mass and the principal directions of rotary inertia, but redistributing only the added mass with a relatively small scale factor may have a lesser impact.

Alternatively, you can scale the mass to redistribute it uniformly (in proportion with the pre-adjusted mass) without affecting the center of mass and the principal directions of rotary inertia.

Input File Usage: Use one of the following options to maximize the time increment by redistributing the total mass of the element set:

**MASS ADJUST*, TARGET DT=MAXIMIZE
elem_set_name, , , scale_factor, REDIST

**MASS ADJUST*
elem_set_name, , MAXIMIZE, *scale_factor*, REDIST

Use one of the following options to redistribute only the added mass:

**MASS ADJUST*, TARGET DT=MAXIMIZE
elem_set_name, , , scale_factor, ADD

**MASS ADJUST*
elem_set_name, , MAXIMIZE, *scale_factor*, ADD

Use the following option to scale the mass for an element set without altering its center of mass:

**MASS ADJUST*
elem_set_name, , UNIFORM, *scale_factor*

Nonstructural Mass Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Point Masses*
- *Density*
- *Mass Adjustment*
- **NONSTRUCTURAL MASS*
- *Defining nonstructural mass*

Overview

A nonstructural mass:

- is a contribution to the model mass from features that have negligible structural stiffness (such as paint on sheet metal panels in a car);
- can be used to bring the net mass of one or more components in the model up to a known value;
- can be positive to add mass to the model and negative to remove mass from the model, with the corresponding increase or decrease in the element stable time increment in an Abaqus/Explicit analysis;
- can be specified in the form of a total mass of the nonstructural features to be distributed over one or more components in the model;
- can be specified in the form of an increase in density over the smeared region;
- can be specified in the form of mass per unit area to be applied over a smeared region consisting of shells, membranes, and/or surface elements; and
- can be specified in the form of mass per unit length to be applied over a smeared region consisting of beam, pipe, and/or truss elements.

Nonstructural Mass

The mass contribution from nonstructural features can be included in the model even if the features themselves are omitted. The nonstructural mass is smeared over an element set that is typically adjacent to the nonstructural feature. This element set can contain solid, shell, membrane, surface, beam, pipe, or truss elements. The nonstructural mass can be specified in the following forms:

- a total mass value,
- a mass per volume,
- a mass per area (for element sets that contain conventional shell, membrane, and/or surface elements), or
- a mass per length (for element sets that contain beam, pipe, and/or truss elements).

When a total mass is spread over an element set region, it can be distributed either in proportion to the underlying element “structural” mass or in proportion to the element volume in the initial configuration.

A “structural” mass is defined as the sum of all the mass contributions to an element outside of the nonstructural features. This might include the mass due to any material definitions associated with the element; any “mass per

area” given on the section definition for shell, membrane, and surface elements; mass from any rebars included in shell, membrane, and surface elements; and any additional inertia given on the section definition of beam/pipe elements. A nonstructural mass contribution to an element is not allowed if that element has no structural mass.

A given element in the model can have contributions from multiple nonstructural mass specifications. The nonstructural mass in a given element will participate in any mass proportional distributed loads, such as gravity loading, defined on that element. When a nonstructural mass is added to a shell, beam, or pipe element with active rotational degrees of freedom, the nonstructural contribution affects both the element mass and the element rotary inertia. The element stable time increment increases with a positive nonstructural mass and decreases with a negative nonstructural mass. In general, it is easier to use a nonstructural mass definition to bring an additional mass into the model than to do the same with a group of point masses. It is also more beneficial in an Abaqus/Explicit analysis due to a possibly higher time increment.

Any mass proportional damping specified as part of the material definition (see *Material Damping*) will also apply to the nonstructural mass contribution assigned to the element or element set using that material definition.

Nonstructural mass contributions associated with an element set are not imported when transferring model data between Abaqus analyses (see *About Transferring Results between Abaqus Analyses*). These contributions need to be redefined in the import analysis if they are to be included in the model.

Defining Nonstructural Mass

To define a nonstructural mass contribution to the model mass, you must first identify the region over which the contribution must be added. You then specify the value of the nonstructural mass using the appropriate units and, if the total mass from the nonstructural features is known, determine how the nonstructural mass is distributed over the region.

Input File Usage: **NONSTRUCTURAL MASS*, ELSET=*element_set_name*

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region

Specifying the Units of the Nonstructural Mass

The nonstructural mass can be specified in different types of units, depending on the types of elements contained in the specified region.

Specifying Units of Total Mass

A total nonstructural mass with units of “total mass” can be spread over a region containing solid, shell, membrane, beam, pipe, and/or truss elements.

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=TOTAL MASS
total mass of the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Total Mass**: **Magnitude:** *total mass of the nonstructural feature*

Specifying Units of Mass per Volume

A nonstructural mass with units of “mass per volume” can be spread over a region containing solid, shell, membrane, beam, pipe, and/or truss elements.

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=MASS PER VOLUME
added density due to the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Mass per Volume**: **Magnitude:** *added density due to the nonstructural feature*

Specifying Units of Mass per Area

A nonstructural mass with units of "mass per area" can be spread over a region containing conventional shells, membranes, and/or surface elements.

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=MASS PER AREA
added mass per area due to the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Mass per Area**: **Magnitude:** *added mass per area due to the nonstructural feature*

Specifying Units of Mass per Length

A nonstructural mass with units of "mass per length" can be spread over a region containing beam, pipe, and/or truss elements.

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=MASS PER LENGTH
added mass per length due to the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Mass per Length**: **Magnitude:** *added mass per length due to the nonstructural feature*

Controlling the Distribution of the Total Mass from Nonstructural Features

There are two methods available for distributing the nonstructural mass over the region when the total mass from the nonstructural features is known.

Distributing the Nonstructural Mass in Proportion to the Element Structural Mass

If you do not want to change the center of mass for the region, distribute the nonstructural mass in proportion to the element structural mass. This method results in a uniform scaling of the structural density of the region. Abaqus uses mass proportional distribution by default.

The element structural mass in shell, membrane, and surface elements includes any mass contribution from rebar provided that the rebar are defined as a rebar layer (see *Defining Reinforcement*).

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=TOTAL MASS, DISTRIBUTION=MASS PROPORTIONAL
total mass of the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Total Mass**: **Magnitude:** *total mass of the nonstructural feature*: **Distribution: Mass Proportional**

Distributing the Nonstructural Mass in Proportion to the Element Volume

Alternatively, you can distribute the nonstructural mass in proportion to the element volume in the initial configuration. This method results in a uniform value added to the underlying structural density over the region. Therefore, the center of mass for the region may be altered if the region has nonuniform structural density.

Input File Usage: **NONSTRUCTURAL MASS*, UNITS=TOTAL MASS, DISTRIBUTION=VOLUME PROPORTIONAL
total mass of the nonstructural feature

Abaqus/CAE Usage: Property or Interaction module: **Special->Inertia->Create**: **Nonstructural mass**: select region: **Units: Total Mass**: **Magnitude:** *total mass of the nonstructural feature*: **Distribution: Volume Proportional**

Distribution Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Orientations*
- *About the Material Library*
- *Material Data Definition*
- *Combining Material Behaviors*
- *Density*
- *Linear Elastic Behavior*
- *Thermal Expansion*
- *Solid (Continuum) Elements*
- *Membrane Elements*
- *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*
- *Using a General Shell Section to Define the Section Behavior*
- *Using a Beam Section Integrated during the Analysis to Define the Section Behavior*
- *About Connectors*
- *Contact Initialization for General Contact in Abaqus/Explicit*
- *Mean-Field Homogenization*
- **DISTRIBUTION*
- **DISTRIBUTION TABLE*
- *The Discrete Field toolset*

Overview

A distribution:

- is a spatially varying field defined over elements, nodes, or element faces in an Abaqus model;
- can be used to define shell thicknesses for shell elements with displacement degrees of freedom;
- can be used to define beam radii for beam elements with a solid circular section when numerical integration across the section is required;
- can be used to define shell stiffness;
- can be used to define local coordinate systems on solid continuum and shell elements that have displacement degrees of freedom;
- can be used to define local material directions or fiber directions on solid continuum and shell elements with anisotropic materials (such as anisotropic hyperelastic materials) or, in Abaqus/Explicit, fabric materials;
- can be used to define orientation angles on the layers of composite shell elements that have displacement degrees of freedom;
- can be used to define orientation angles for connector elements;
- can be used to define thicknesses on the layers of conventional composite shell elements;
- can be used to specify initial contact clearances;

- can be used to specify the volume fraction, aspect ratio, and second-order orientation tensor of a constituent in a multiscale material; and
- can be used in an adjoint design sensitivity analysis to specify scale factors and a quantity related to the scale factor derivatives for scaling the mass, stiffness, or design stress material attribute on an element-by-element basis; to specify scale factors for material thermal conductivity on an element-by-element basis; or to specify nodal adjustments (displacements) on a node-by-node basis;
- in an Abaqus/Standard analysis can be used to define mass density, linear elastic material behavior, and thermal expansion for solid continuum elements; shell offsets; orientation angles and thicknesses on the layers of composite solid continuum elements; local coordinate systems on membrane elements; and membrane thickness on an element-by-element basis; and
- can be used to define the particle radius and particle surface area per unit volume on an element-by-element basis in a battery analysis that uses porous electrode theory.

Distributions

A distribution is a spatial analogy of an amplitude definition (see *Amplitude Curves*). Amplitude definitions are used to provide arbitrary time variations of loads, displacements, and other prescribed variables. Distributions are used to specify arbitrary spatial variations of selected element properties, material properties, local coordinate systems, boundary conditions, and spatial variations of initial contact clearances.

The two main components of a distribution are its *location* and *field data*. The location identifies where the distribution is defined, either on elements, nodes, or element faces. Field data are a specified number of floating point values defined for each element, node, or element face in the distribution.

To define a distribution, you must assign it a unique name. You must also specify the number and physical dimension of each data value in the distribution by referring to a distribution table.

Input File Usage: **DISTRIBUTION*, NAME=*name*, TABLE=*distribution table name*

Abaqus/CAE Usage: Abaqus/CAE supports distributions using discrete fields.

Property, Interaction, or Load module: **Tools->Discrete Field->Create**

Specifying the Location of a Distribution

You can define a distribution on elements or nodes. Distributions on nodes are supported only for defining initial contact clearances as described in *Contact Initialization for General Contact in Abaqus/Explicit*. All other applications of distributions require distributions defined on elements.

There is no limit on the number of distributions to which a given element or node can belong. Elements and nodes cannot be combined within the same distribution definition.

Defining a Distribution on Elements

Defining a distribution on elements requires you to specify field data for each element or element set included in the distribution definition. All distributions on elements require that default data be defined. Default data are used for all elements that are not specifically assigned a value in the distribution.

Input File Usage: **DISTRIBUTION*, LOCATION=ELEMENT
blank space, *field data*
element set or element number, *field data*

Default data are defined by using a blank space instead of an element number or element set for the first data item on the first data line of a distribution definition. Only one set of default data can be defined for a distribution. If you specify only default data, all elements that reference that distribution use the default values.

If an element is specified more than once in a given distribution definition, the last specification given is used.

Abaqus/CAE Usage: Property, Interaction, or Load module: **Tools->Discrete Field->Create**: **Definition**: **Elements**

Defining a Distribution on Nodes

Defining a distribution on nodes requires you to specify field data for each node or node set included in the distribution definition.

Input File Usage: **DISTRIBUTION*, LOCATION=NODE
node set or node number, *field data*

If a node is specified more than once in a given distribution definition, the last specification given is used.

Abaqus/CAE Usage: Defining a distribution on nodes for initial contact clearances is not supported in Abaqus/CAE.

Defining a Distribution Table

Every distribution definition must refer to a distribution table. A distribution table defines the number of field data items needed for each element or node in a distribution. The distribution table also defines the physical dimension of each data value in a distribution. A distribution table can be referred to as many times as needed by different distributions. The distribution table consists of a list of predefined labels shown in *Table 1*. The combination of labels needed for a given distribution is determined by how the distribution is applied.

Table 1: Distribution table labels—Abaqus/Standard and Abaqus/Explicit.

Label	Physical dimension	Number of data items per label
ANGLE	angle in degrees	1
COORD3D	(L, L, L)	3
DENSITY	ML^{-3}	1
DIR3D	dimensionless	3
DISP3D	(L, L, L)	3
EXPANSION	θ^{-1}	1
LENGTH	L	1
MODULUS	FL^{-2}	1
ORIENTS	dimensionless	6
ORITENS	dimensionless	6
RATIO	dimensionless	1

Label	Physical dimension	Number of data items per label
SHELLSTIFF1	FL^{-1}	1
SHELLSTIFF2	F	1
SHELLSTIFF3	FL	1
PARTICLE1RADIUS	M	1
PARTICLE2RADIUS	M	1
PARTICLE3RADIUS	M	1
PARTICLE1AS	M^{-1}	1
PARTICLE2AS	M^{-1}	1
PARTICLE3AS	M^{-1}	1

Input File Usage: Use the following option to define a distribution table:

**DISTRIBUTION TABLE*, NAME=*distribution table name*
list of labels

Abaqus/CAE Usage: Abaqus/CAE creates a distribution table when you specify a distribution by selecting a discrete field.

Defining Distributions by Importing Field Data from an Output Database File

For two- and three-dimensional continuum elements and three-dimensional conventional shell elements, you can define distribution data using output variables from a particular step and increment or a user-specified time in the output database (`.sim`) file from a previous analysis. For more information, see *General Capability for Importing External Fields*.

- If the previous analysis is performed with third-party software, the results file must be converted to the `.sim` file format.
- You can use any appropriate result from the output database by specifying an output variable identifier (see *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers*, for available output variable identifiers).
- There are some variables that can be read from the output database (`.sim`) file for which you cannot request output. For example, you cannot request output for the variable ORIENT; it is written automatically to the output database (`.sim`) file by Abaqus/Standardand Abaqus/Explicit if an orientation is associated with the results data in the analysis.
- The location of the results can be either at nodes or elements. Results data requested at integration points are mapped to the centroid of the target element, and only data at the first section point in the shell element are imported. No interpolation or averaging is performed through the thickness.
- When importing orientations from composite shells, you should import output variable SORIENT (if available) instead of variable ORIENT. If you import ORIENT from a composite shell and you specify the rotation angle as a single scalar value in the first layer, the imported orientation is the orientation at the first section point rotated to the section orientation using the user-specified rotation angle.
- When importing orientations from shell elements, you must specify the 3-axis as the axis for additional rotation in the orientation definition.
- When importing tensor or vector fields, the field data are transformed to the global system before mapping; mapped data are then transformed to the local system if specified in the target element.

You can specify a source region (node or element set in the previous model) if data are imported only from a subset of the previous model. Sometimes a source region is also specified to eliminate ambiguity during mapping. You can specify a target region if data are specified only on a subset of the current model.

You must specify the full name of the output database file including the file extension `.sim`.

You can import only results data requested on two- and three-dimensional continuum elements and three-dimensional conventional shell elements.

You can specify mapping tolerances and special tensor averaging methods if mapping is performed. If the model in the previous analysis is repositioned in the current analysis, you must specify the translation and rotation so that the source region can be repositioned before data are imported, except in the following cases:

- Scalar data are imported from a matching mesh.
- Tensor data are imported from a matching mesh, and there is no rotation between the source region and the target region.

Input File Usage: Use the following options to define a distribution from a user-specified output database (`.sim`) file with repositioning (data are imported from the entire model to the entire model):

**DISTRIBUTION*, NAME=*distribution table name*
blank space, *field data*
**EXTERNAL FIELD*, FILE=*file name*
, , , , , *outputvar*, *mapcontrol*, *fieldop*

**FIELD MAPPER CONTROLS*, NAME=*mapcontrol*
, , 1.0

**FIELD OPERATIONS*, NAME=*fieldop*
x, *y*, *z*
x1, *y1*, *z1*, *x2*, *y2*, *z2*, *rotation*

Abaqus/CAE Usage: Defining distributions by importing field data from an output database file is not supported in Abaqus/CAE.

Applying Distributions

The data defined in a distribution are not used in an Abaqus analysis unless the distribution is referred to by name by a feature that supports distributions, and the distribution is applied only to the elements, nodes, or element faces that are associated with the referenced feature. In addition, a distribution definition can be referenced more than one time in a given model. These points are illustrated in the examples below.

If an element in an Abaqus/Standard or Abaqus/Explicit analysis is declared rigid (see *Rigid Body Definition*) any distributions used to define element properties, material properties (with the exception of density), or local coordinate systems are ignored.

Examples

The simple examples below illustrate how distributions are defined. A large number of illustrative example problems using distributions can be found in *Spatially varying element properties*.

Example 1

A distribution for shell thickness is defined and applied to two different shell section definitions through the SHELL THICKNESS parameter—as noted above the distribution `dist0` would not be used if it is not referred to by a feature that supports distributions. See *Using a Shell Section Integrated during the Analysis to Define the Section Behavior* for more details. The distribution table defines both the number of data values (one) and the physical dimension (LENGTH) of the thickness data. The thicknesses defined in distribution `dist0` are assigned only to shell elements that belong to the element set `elset1` or `elset2`. The default thickness (t_0) defined in the first data line of `dist0` will be assigned to all elements in `elset1` and `elset2` that are not explicitly assigned a thickness in `dist0`.

```
*DISTRIBUTION TABLE, NAME=tab0
 LENGTH
*DISTRIBUTION, NAME=dist0, LOCATION=element, TABLE=tab0
                       , t0
 element set or number, t1
 element set or number, t2
 …
*SHELL SECTION, ELSET=elset1, SHELL THICKNESS=dist0
*SHELL SECTION, ELSET=elset2, SHELL THICKNESS=dist0
```

Example 2

A distribution for spatially varying isotropic elastic material behavior is defined and applied to a material definition (*Linear Elastic Behavior*). This material is then referred to by a solid section definition. This is important because like any material definition, a material defined by a distribution is not used unless it is referred to by a section definition, and then it is applied only to the elements associated with the section definition. The distribution table defines both the number of data values (two) and the physical dimensions (MODULUS and RATIO) of the isotropic elastic data. Other material behaviors (in this case plasticity) can also be included in the material definition. The default elastic constants (E_0, ν_0) in distribution `dist1` will be assigned to all elements in `elset3` that are not explicitly assigned elastic constants in `dist1`.

```
*DISTRIBUTION TABLE, NAME=tab1
 MODULUS, RATIO
*DISTRIBUTION, NAME=dist1, LOCATION=element, TABLE=tab1
                       , E0, ν0
 element set or number, E1, ν1
 element set or number, E2, ν2
 …
*MATERIAL, NAME=MAT
*ELASTIC
 dist1
*PLASTIC
 …
*SOLID SECTION, ELSET=elset3, MATERIAL=MAT
```

Example 3

A spatially varying local coordinate system (*Orientations*) is defined by specifying both spatially varying coordinates for points *a* and *b* as well as a spatially varying additional rotation angle. This orientation is then referred to by a general shell section definition. This is important because like any orientation definition, an orientation defined by a distribution is not used unless it is referred to by a section definition, and then it is applied only to the elements associated with the section definition. The distribution table for the coordinates

specifies COORD3D twice to indicate that data for two three-dimensional coordinates points must be specified for each element in the distribution.

```
*DISTRIBUTION TABLE, NAME=tab2
 COORD3D, COORD3D
*DISTRIBUTION, NAME=dist2, LOCATION=element, TABLE=tab2
                      , aX0,aY0,aZ0,bX0,bY0,bZ0
 element set or number, aX1,aY1,aZ1,bX1,bY1,bZ1
 element set or number, aX2,aY2,aZ2,bX2,bY2,bZ2
 …
*DISTRIBUTION TABLE, NAME=tab3
 ANGLE
*DISTRIBUTION, NAME=dist3, LOCATION=element, TABLE=tab3
                      , θ0
 element set or number, θ1
 element set or number, θ2
 …
*ORIENTATION, NAME=ORI, DEFINITION=COORDINATES
 dist2
 3, dist3
*SHELL GENERAL SECTION, ELSET=elset4, ORIENTATION=ORI
```

Example 4

Spatially varying thicknesses and orientation angles are defined on the layers of a composite shell element. The distribution table for the thicknesses specifies LENGTH, and the distribution table for the orientation angles specifies ANGLE. A distribution of thicknesses is used on layers 1 and 3, while a distribution of angles is used on layers 2 and 3.

```
*DISTRIBUTION TABLE, NAME=tableThick
 LENGTH
*DISTRIBUTION, NAME=thickPly1, LOCATION=element, TABLE=tableThick
                      , t0
 element set or number, t1
 element set or number, t2
 …
*DISTRIBUTION, NAME=thickPly3, LOCATION=element, TABLE=tableThick
                      , t0
 element set or number, t1
 element set or number, t2
 …
*DISTRIBUTION TABLE, NAME=tableOriAngle
 ANGLE
*DISTRIBUTION, NAME=oriAnglePly2, LOCATION=element,
TABLE=tableOriAngle
                      , ϕ0
 element set or number, ϕ1
 element set or number, ϕ2
 …
*DISTRIBUTION, NAME=oriAnglePly3, LOCATION=element,
TABLE=tableOriAngle
                      , ϕ0
 element set or number, ϕ1
 element set or number, ϕ2
 …
*SHELL SECTION, ELSET=elset1, COMPOSITE
```

```
 thickPly1, 3, mat1, 0.
        1., 3, mat2, oriAnglePly2
 thickPly3, 3, mat3, oriAnglePly3
```

Example 5

A distribution for a spatially varying volume fraction is defined in distribution `distVF`. A distribution for a spatially varying aspect ratio is defined in distribution `distAR`, and a distribution for the second-order orientation tensor is defined in distribution `distOriTens`. These distributions are then applied to the definition of a constituent, which is then used in the material definition of a multiscale material with mean-field homogenization (*Mean-Field Homogenization*). The distribution table for the volume fraction and aspect ratio specifies RATIO, and the distribution table for the orientation tensor specifies ORITENS.

```
*DISTRIBUTION TABLE, NAME=vfTable
 RATIO
*DISTRIBUTION TABLE, NAME=arTable
 RATIO
*DISTRIBUTION TABLE, NAME=oriTable
 ORITENS
*DISTRIBUTION, NAME=distVF, LOCATION=element, TABLE=vfTable
 element set or number, v_f
*DISTRIBUTION, NAME=distAR, LOCATION=element, TABLE=arTable
 element set or number, ar
*DISTRIBUTION, NAME=distOriTens, LOCATION=element, TABLE=oriTable
 element set or number, a_11,a_22,a_33,a_12,a_13,a_23
 …
*MATERIAL, NAME=MAT
*MEAN FIELD HOMOGENIZATION
*CONSTITUENT
 distVF, distAR, distOriTens
```

Example 6

Orientations defined as part of the connector section can be defined as distributions.

```
*DISTRIBUTION, NAME=distOri, LOCATION=element, TABLE=oriTable
 element set or number, a_x,a_y,a_z,b_x,b_y,b_z
*ORIENTATION, NAME=Ori
distOri
 *CONNECTOR SECTION
 connection type
Ori, Ori
```

Display Body Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- **DISPLAY BODY*
- *Defining display body constraints*

Overview

A display body:

- can be two-dimensional planar, axisymmetric, or three-dimensional;
- is associated with a part instance and up to three reference nodes, such that the motion of the part instance is governed by the motion of the reference nodes;
- is used for display purposes only and does not take part in the analysis;
- can be used to make the analysis more efficient while improving visualization of analysis results; and
- is especially useful for mechanism or multibody dynamic analyses.

What Is a Display Body?

A display body is a part instance that is used for display only. None of the nodes or elements of the instance take part in the analysis, but they are still available during postprocessing. The motion of the display body is governed by the motion of the associated reference nodes, if any. It behaves like a rigid body since the relative positions of the nodes and elements of the part instance remain constant throughout a simulation. The nodes and elements of the part instance cannot be used to define prescribed conditions, interactions, constraints, etc. Section properties do not have to be assigned to the elements.

A display body is useful in cases where the physical model is different from the idealized model used for the analysis. An idealized model may be difficult to visualize; it may help to include more details in the model for realistic postprocessing purposes. Display bodies allow this without increasing the analysis time.

Display bodies are especially useful in mechanism or multibody dynamics problems where rigid parts interact with each other via connectors. In such cases a part can be represented by a very simple rigid body and a more complex display body. In this case, the rigid body can be as simple as just a node, along with mass and rotary inertia elements attached to that node.

Display bodies can also be used to model stationary objects that are not involved in the analysis but aid in visualization.

Creating a Display Body

You must specify the part instance to be made a display body.

Input File Usage: **DISPLAY BODY*, INSTANCE=*name*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Display body**: select part instance

The Reference Nodes

If the display body is not associated with any reference nodes, it will remain fixed in space during the analysis. However, you can specify that the motion of the display body should be governed by the motion of selected reference nodes. These nodes must belong to another part instance in the assembly. They cannot belong to another display body definition. If you specify only one reference node, the display body will translate and rotate based on the translations and rotations of that node during the analysis. If the reference node has no rotational degrees of freedom, the display body will not rotate during the analysis.

If you specify three reference nodes, the display body will translate and rotate based on the translations of all three nodes. The new position of the part instance at any time will be calculated from the new position and orientation of the coordinate system defined by the three reference nodes: the first node will be the origin, the second will be a point in the *x*-direction, and the third node will be a point in the *X*–*Y* plane. Care should be taken when specifying the three nodes so that they do not become colinear at any stage of the analysis. If this occurs, the position of the part instance may change abruptly through that increment.

Input File Usage: **DISPLAY BODY*, INSTANCE=*name*
first reference node number, second reference node number, third reference node number

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Display body**: select part instance, choose **Follow single point** or **Follow three points**, click **Edit**, and select the reference points

Using Display Bodies with Connectors

Display bodies can be used effectively in models containing rigid part instances that interact with each other using connector elements. Such models need both rigid bodies and display bodies. The rigid body should contain any nodes used by connectors, used to define mass and inertia properties, and used to apply loads or boundary conditions. The display body should contain the nodes and elements representing the physical part. Care should be taken to ensure that the nodes in the rigid body are not part of the display body. The reference node of the display body will typically be the same as the rigid body reference node.

Figure 1(a) illustrates a model containing rigid bodies and a display body.

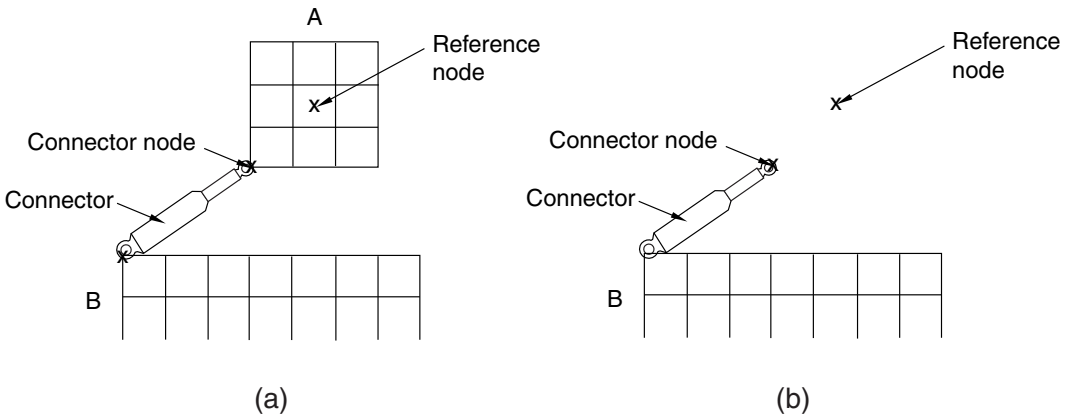


Figure 1: Example of a display body.

Part instance A is included in a display body definition. *Figure 1*(b) shows the same model without the display body. This model will actually be involved in the analysis. The connector node and reference node form a rigid

body that represents the analysis version of part instance A. Both these nodes are assembly-level nodes and are not included in the display body.

Input File Template

The following input shows how display bodies can be used in a model with rigid part instances and connectors:

```
*ASSEMBLY
...
*INSTANCE, NAME=INST1
...
*END INSTANCE
*NODE, NSET=INST1-REFNODE
1001, -10, 0, 0
*NODE, NSET=INST1-CONNECTOR-NODE
1002, -5, -5, 0
*RIGID BODY, TIE NSET=INST1-CONNECTOR-NODE,
REF NODE=INST1-REFNODE
*DISPLAY BODY, INSTANCE=INST1
1001
...
*END ASSEMBLY
```

Assembly Definition

Products: Abaqus/Standard Abaqus/Explicit

References:

- **ASSEMBLY*
- **INSTANCE*
- **PART*

Overview

A finite element model in Abaqus can be defined as an assembly of part instances. The organization of such a model:

- is consistent with models generated by Abaqus/CAE and displayed in the Visualization module (Abaqus/Viewer); and
- allows reuse of part definitions, which is valuable for creating large, complex models.

By default, input files written by Abaqus/CAE are written in terms of an assembly of part instances. For input files not written by Abaqus/CAE, the use of part and assembly definitions in the input file is currently optional. However, since the Visualization module displays results in terms of an assembly of part instances, an assembly and at least one part instance will be created automatically by the analysis input file processor if they are not defined in the input file.

Introduction

A physical model is typically created by assembling various components. The assembly interface in Abaqus allows analysts to create a finite element mesh using an organizational scheme that parallels the physical assembly. In Abaqus the components that are assembled together are called *part instances*. This section explains how to organize an Abaqus finite element model in terms of an assembly of part instances.

The mesh is created by defining parts, then assembling instances of each part. Each part can be used (instanced) one or more times, and each part instance has its own position within the assembly. This organization of the model definition matches the way models are created in Abaqus/CAE, where the assembly can be created interactively or imported from an input file (see the *Introduction*).

Terminology

1. Placeholder for glosslist element

Example

A hinge can be modeled using two flanges and a pin, as shown in *Figure 1*. The flange geometry is defined by creating a part, which is instanced twice inside the hinge assembly. Another part, the pin, is created and instanced once. The pin is modeled as a rigid body created from an analytical surface (see *Analytical Rigid Surface Definition*).

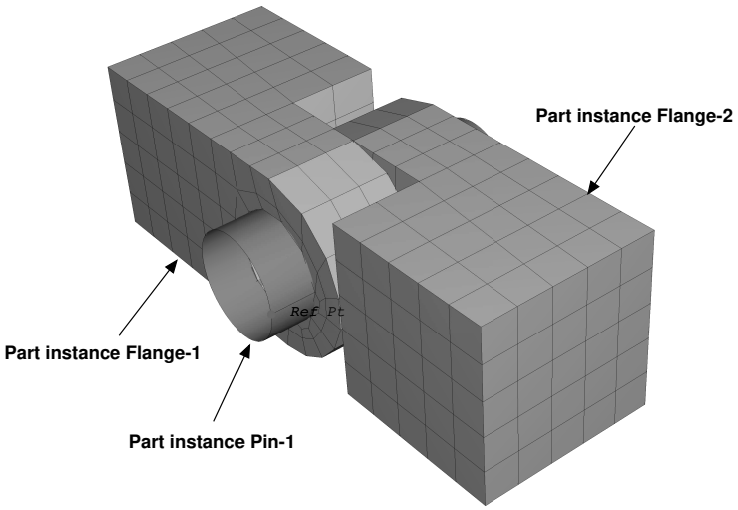


Figure 1: The hinge assembly.

This hinge example is used throughout this section to illustrate the keyword interface for parts and assemblies. This example is also used to illustrate the interactive assembly process (see *Getting Started with Abaqus/CAE*).

Defining Parts, Part Instances, and the Assembly

Everything defined within a part, instance, or the assembly is local to that part, instance, or the assembly. This means that node/element identifiers and names (like set and surface names) need not be unique throughout a model; they need only be unique within the part, instance, or assembly where they are being defined (see *Viewing Part and Assembly Information in the Data File*). Names should not use an underscore to join part instance names to element set, node set, orientation names, or distribution names because the names may conflict with internal names used by Abaqus.

For example, consider *Figure 2*. In this model the assembly (`Hinge`) contains three part instances (`Flange-1`, `Flange-2`, and `Pin-1`). Multiple sets named `top` can be defined: in this case one is defined within the assembly and one is defined within each of the `Flange` part instances. The set name `top` can be reused, and each set named `top` is independent from the others.

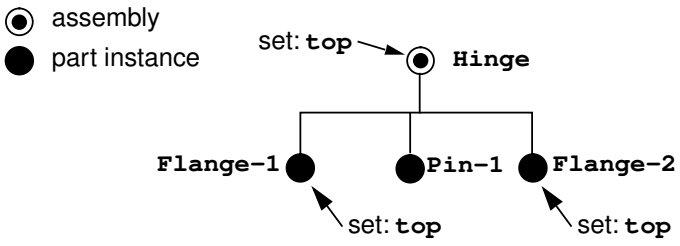


Figure 2: The organization of the `Hinge` assembly.

Input File Usage: Use the following options to begin and end each part, instance, and assembly definition:

**PART/*END PART*
**INSTANCE/*END INSTANCE*
**ASSEMBLY/*END ASSEMBLY*

If any one of these options appears in an input file, they must all appear except when you import a part instance from a previous analysis; in this case **PART* and **END PART* are not required. The model must be consistently defined as an assembly of part instances.

Defining a Part

A part definition must appear outside the assembly definition. Multiple parts can be defined in a model; each part must have a unique name.

Input File Usage: Use the following options to define a part:

**PART*, NAME=*PartName*
Node, element, section, set, and surface definitions
**END PART*

Defining Part Instances

A part instance definition must appear within the assembly definition. If the part instance is not imported from a previous analysis, each part instance must have a unique name and refer to a part name. A part instance name of `Assembly` is not allowed. In addition, you can specify data that are used to position the instance within the assembly. Give a translation and rotation for the part instance relative to the origin of the assembly (global) coordinate system.

If the part instance is to be imported from a previous analysis, each part instance must specify the name of the instance to be imported. For more information on defining part instances for use with the import capability, see *About Transferring Results between Abaqus Analyses*.

Additional sets and surfaces can be defined at the instance level, as explained later in this section.

Input File Usage: Use the following options to instance a part that is not imported from a previous analysis:

**INSTANCE*, NAME=*InstanceName*, PART=*PartName*
<positioning data>
Additional set and surface definitions (optional)
**END INSTANCE*

Repeat these options, each time referring to the same part name, to instance a part multiple times.

Use the following options to import a part instance from a previous analysis:

**INSTANCE*, INSTANCE=*instance-name*
Additional set and surface definitions (optional)
**IMPORT*
**END INSTANCE*

Defining the Assembly

Only one assembly can be defined in a model. All part instance definitions must appear within the assembly definition.

Sets and surfaces can be defined at the assembly level by including the appropriate definitions within the assembly definition.

Input File Usage: Use the following options to create an assembly:

**ASSEMBLY*, NAME=*name*
Part instance definitions
Set and surface definitions
Connector and constraint definitions
Rigid body definitions
**END ASSEMBLY*

Example

The hinge assembly shown in *Figure 1* can be defined using the following syntax in the input file:

```
*PART, NAME=Flange
   *NODE, NSET=Flange
    1, ...
    2, ...
    ...
    360, ...
   *ELEMENT, ELSET=Flange
    1, ...
    2, ...
    ...
    200, ...
   *SOLID SECTION, ELSET=Flange, MATERIAL=Steel
   *ELSET, ELSET=Flat, GENERATE
    176, 200, 1
   *SURFACE, NAME=Flat
    Flat, S1
*END PART
*PART, NAME=Pin
   *NODE, NSET=RefPt
    1, ...
   *SURFACE, TYPE=REVOLUTION, NAME=Pin
    ...
   *RIGID BODY, REF NODE=1, ANALYTICAL SURFACE=Pin
*END PART
*ASSEMBLY, NAME=Hinge
   *INSTANCE, NAME=Flange-1, PART=Flange
    <positioning data>
   *END INSTANCE
   *INSTANCE, NAME=Flange-2, PART=Flange
    <positioning data>
   *END INSTANCE
   *INSTANCE, NAME=Pin-1, PART=Pin
    <positioning data>
   *END INSTANCE
   *ELSET, ELSET=Top
    ...
   *NSET, NSET=Output
    ...
*END ASSEMBLY
*MATERIAL, NAME=Steel
 ...
```

Notes

- All of the nodes and elements that describe the `Flange` part are defined between the **PART* and **END PART* options. The section definition (**SOLID SECTION*) must also appear within the part definition.
- At least one element set must be defined within the `Flange` part so that the section definition can refer to it. Additional node and element sets can also be defined in the part.
- The `Flange` part is instanced twice in the `Hinge` assembly. Therefore, the model contains two element sets named `Flat`: one belongs to part instance `Flange-1`, and the other belongs to part instance `Flange-2`.
- When a meshed part is instanced, the node and element numbers are repeated in each part instance.
- The `Pin` part is instanced once. It is a rigid body created from an analytical surface (see *Analytical Rigid Surface Definition*).
- Keywords can be indented to help clarify the definition of each part, part instance, and assembly.

Organizing the Model Definition

In a traditional Abaqus model without an assembly definition, the components of the model fall into one of two categories: model data (step independent) and history data (step dependent). In an Abaqus model that is organized into an assembly of part instances, all components are further categorized and must fall within the proper level: part, assembly, instance, step, or model. Step-level components correspond to history data; all step-dependent component definitions must appear within a step definition (see *Defining an Analysis*). Model-level data include everything that does not fall into part-, assembly-, instance-, or step-level data (for example, material definitions; see *Figure 3*). The proper level within which a keyword option must appear in the input file is indicated at the top of each section in the *Abaqus Keywords Guide*.

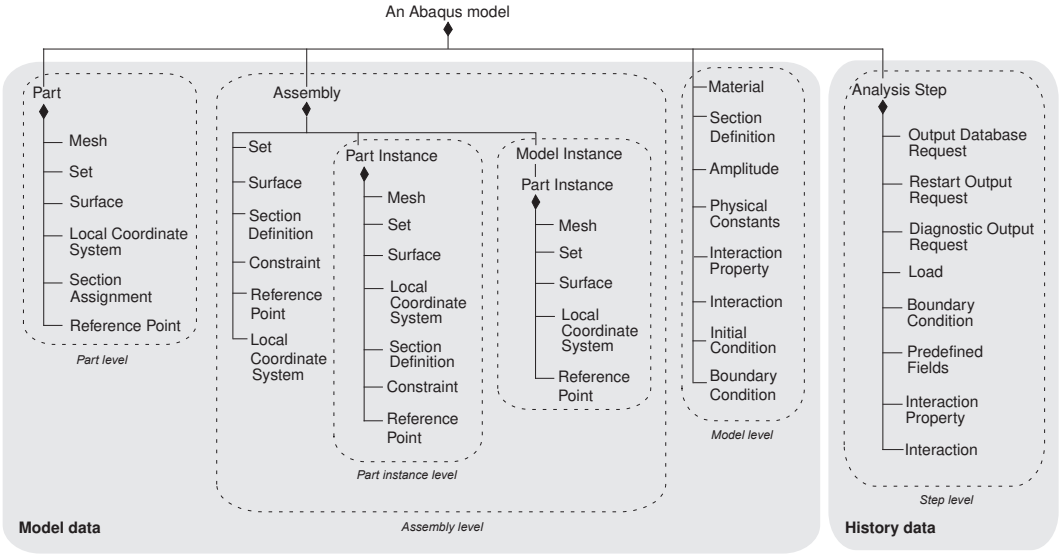


Figure 3: Organization of a model defined in terms of an assembly of part instances.

Rules for Defining an Assembly

The organization shown in *Figure 3* is achieved by following a few basic rules.

Referring to Items between Levels

When creating a model, it is often necessary to refer to something outside of the current level; for example, a section definition within a part must refer to a material, which is defined at the model level. Loads defined within a step must refer to sets within the assembly. But some references between levels are not allowed; for example, a set in one part instance cannot refer to nodes in another part instance. The following references are allowed:

A definition within:	Can refer to items within:
the assembly	an instance
	the model
an instance	the model
a part	the model
a step	the assembly
	an instance
	the model

These rules are illustrated in *Figure 4*.

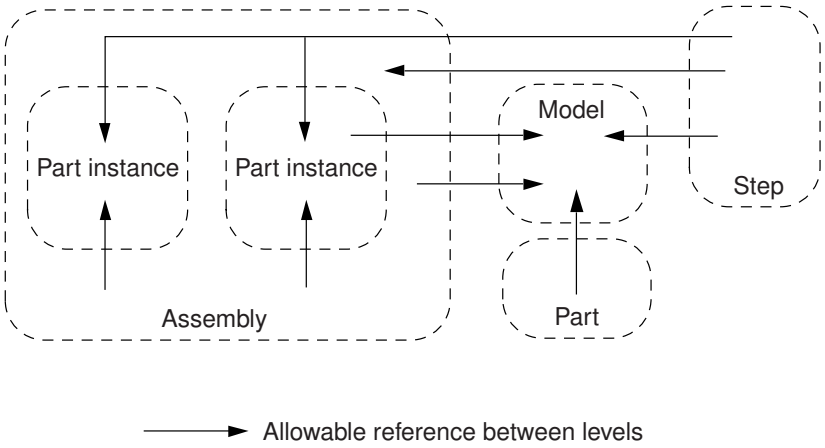


Figure 4: Allowable references between levels.

Naming Conventions

The Abaqus naming conventions allow for a model that contains an assembly. When something is defined within a part, instance, or the assembly and is referred to from outside its level, the complete name must be used to identify it (set `Flat` of instance `Flange-2` in assembly `Hinge`, for example). A complete name is given in the input file using "dot" notation: each name in the hierarchy is separated by a "." (period). For example, some complete names in the `Hinge` assembly are

`Hinge.Flange-2.Flat`	An element set that belongs to part instance `Flange-2`.
`Hinge.Output`	A node set that belongs to assembly `Hinge`.

Such names would be used to refer to the sets from outside the assembly. The same syntax is used to refer to individual nodes or elements.

`Hinge.Flange-1.3`	A node or element that belongs to part instance `Flange-1`.
`Hinge.Flange-2.11`	A node or element that belongs to part instance `Flange-2`.

As always, the context determines whether a node or element is being referred to. The "." has special meaning; it is used to separate the individual names in a complete name. Therefore, the "." cannot be used in labels such as set and surface names. For example,

`ELSETAssign elements to an element set., ELSET=Set.1`	Error
`ELSETAssign elements to an element set., ELSET=Set1`	OK

Complete names are limited to 80 characters, including the periods.

However, when referring to a name in an input file that is not defined in terms of an assembly of part instances, the "." in the name should be replaced by underscores. Such a situation can occur, for example, when an element set from a previous analysis is referred to by the current analysis but the current input file is not defined in terms of an assembly of part instances.

Quoted Labels

Labels for set and surface names can be defined by enclosing the label in quotation marks (see *Input Syntax Rules*). Any subsequent use of the label in a complete name must be enclosed in quotation marks as well. For example,

```
*PART, NAME=Flange
 ...
*ELSET, ELSET="Set 1"
 ...
*END PART
 ...
*ELEMENT OUTPUT, ELSET=Hinge.Flange-1."Set 1"
```

Example

An assembly node set `Top` can be defined by the following syntax:

```
*ASSEMBLY, NAME=Hinge
   ...
   *NSET, NSET=Top
    Flange-1.2, Flange-1.5, ...
    Flange-2.1, Flange-2.4, ...
*END ASSEMBLY
```

Since the node set is defined within the assembly level, `Hinge.` is not part of the complete names given on the data lines. However, the prefix `Hinge.` would be required to request output for this node set, since the output request exists within the step definition, which is outside the assembly level.

```
*STEP
     ...
     *NODE OUTPUT, NSET=Hinge.Top
*END STEP
```

Similarly, a boundary condition could be applied to a set defined for part instance `Flange-2`.

```
*STEP
     ...
     *BOUNDARY
      Hinge.Flange-2.FixedEnd, 1, 3
*END STEP
```

The Mesh (Nodes and Elements)

- The mesh can be defined either on a part or on an instance of that part (not both). Typically, parts are meshed and instances inherit that mesh, but it is not required. If, for example, you want to use fully integrated elements

for one part instance and reduced-integration elements for another, or if you want to define a more refined mesh on one part instance than on another, you must mesh the instances separately.

- If the mesh is defined on a part, it is inherited by every instance of that part.
- If the mesh is defined on a part, it cannot be redefined (overridden) on an instance of that part. In other words, if the node and element definitions appear within the part definition, they cannot appear within the instance definition for that part.
- If a mesh is not defined on a part, it must be defined on every instance of that part.

- A part definition is required even if no mesh is defined on it. In such cases the empty part definition is used only to relate various instances to each other via the instance definitions. This allows the Visualization module to group information by part.
- Rebar must be defined within a part along with the elements that are being reinforced.
- Reference nodes can be created at the assembly level.
- Only mass, rotary inertia, capacitance, connector, spring, and dashpot elements can be created at the part or the assembly level. All other element types must be defined within a part (or part instance). To define assembly-level elements that refer to part-level nodes, include the part instance name when defining the element connectivity. For example:

```
*ELEMENT, TYPE=MASS
1, Instance-1.10
```

Section Definitions

- Sections must be assigned where the mesh is defined (either within a part definition or within each instance of the part).
- If a part is meshed, all instances of that part have the same element types and are made of the same materials.
- The set referred to by a section definition must be created at the same level as the mesh and section definition.
- If the part is meshed, the section assignment cannot be overridden at the instance level.

Sets and Surfaces

- Sets and surfaces (rigid or deformable) can be created within a part, part instance, or the assembly.
  - Sets and surfaces can be created on a part if a mesh is defined on the part.
  - Sets and surfaces defined on a part are inherited by each instance of that part.
  - Assembly-level sets and, in Abaqus/Standard, secondary surfaces can span part instances.
- If an element set or node set definition with the same name appears more than once at the same level, the new members are appended to the set.
- A surface definition cannot appear more than once with the same surface name within the same level.
- New sets and surfaces can be created on a part instance. If a set or surface is defined on a part instance and a set or surface with that name was not defined on the part, the set or surface is added to the instance.
- Sets and surfaces cannot be redefined on a part instance. If a set or surface is defined on a part instance and a set or surface with that name was also defined on the part, an error will be generated.
- Sets and surfaces are not step dependent. All sets and surfaces must be defined within a part, part instance, or the assembly.

Defining Assembly-Level Sets

You can refer to a part instance from an element set or node set definition as a shortcut to using the complete name when defining assembly-level sets. Specify the name of the instance that contains the specified elements or nodes. To add elements or nodes from more than one instance to the set, repeat the element set or node set definition (see *Node Definition* and *Element Definition* for more details).

Input File Usage: Use the following options to define assembly-level sets:

**NSET*, NSET=*NsetName*, INSTANCE=*InstanceName*
**ELSET*, ELSET=*ElsetName*, INSTANCE=*InstanceName*

Adding Sets and Surfaces on Restart

- Existing sets and surfaces cannot be redefined on restart.
- Analytical surfaces cannot be created on restart.
- New sets and surfaces (excluding analytical surfaces) can be added to part instances or the assembly on restart. To add a set or surface, give the complete name. As in the original analysis, you can refer to the part instance name from the element set or node set definition to define an assembly-level set in the restart analysis. For example,

```
*HEADING
*RESTART, READ, STEP=1
** Add element set "Bottom" to assembly "Hinge":
*ELSET, ELSET=Hinge.Bottom
 Flange-1.40, Flange-2.99
** Add node set "Top" to assembly "Hinge":
*NSET, NSET=Hinge.Top, Instance=Flange-1
 21, 22, 23, 24, 26, 28, 31
*NSET, NSET=Hinge.Top, Instance=Flange-2
 21, 22, 23, 24, 26, 28, 31
**
** Add element set "Right" to part instance "Flange-2":
*ELSET, ELSET=Hinge.Flange-2.Right
 16, 18, 20, 29
**
** Add surface "surfR" to part instance "Flange-2":
*SURFACE, TYPE=ELEMENT, NAME=Hinge.Flange-2.surfR
Right, S1
**
*STEP
 ...
*END STEP
```

Rigid Bodies

Rigid bodies can be defined at the part or assembly level.

- To define a rigid body at the part level, include the rigid body and rigid body reference node definitions within the part definition.
  - Rigid elements, deformable elements, and analytical surfaces cannot be combined within a part.
  - If a rigid body is defined within a part, all deformable, rigid, or connector elements in the part must belong to the rigid body.

  - Mass, rotary inertia, spring, dashpot, and heat capacitance elements can be included in a part that contains a rigid body definition, but these elements cannot belong to the rigid body.
  - To create a part-level rigid body from an analytical surface, include the surface definition within the part definition. Only one analytical surface is allowed per part.

- To define a rigid body at the assembly level, include the rigid body and reference node definitions within the assembly definition.
  - A rigid body can be created at the assembly level from any combination of rigid elements, deformable elements, and up to one analytical surface.
  - The rigid body definition can refer to assembly-level or part-level sets.
  - A part that contains a rigid body definition cannot be included in an assembly-level rigid body.
- You can define a discrete surface at the part or assembly level independent from the rigid body definition.
- An analytical surface definition can appear only within a part definition, even if the rigid body is defined at the assembly level.

Materials

- Materials are defined at the model level so that they can be reused. The material definition cannot appear within a part, part instance, or the assembly.
- All materials in a model must have unique names.

Interactions

An interaction is a relationship between surfaces or between a surface and its environment. Interactions in Abaqus include contact, radiation, film conditions, and element foundations.

- Interactions are defined at the model level in Abaqus/Standard and at the model level or within steps in Abaqus/Explicit; they cannot be defined within a part, assembly, or instance.

Constraints

Constraints are inflexible coupling mechanisms such as MPCs and equations (see *About Kinematic Constraints*).

- Constraints can be defined within a part or the assembly. They can be defined within a part instance if the mesh is defined within the part instance. Constraints should be defined at the assembly level if they constrain the motion of one part instance relative to another.
- Constraints are translated and rotated according to the positioning data given for a part instance.

Distributions

Distributions are used to specify arbitrary spatial variations of selected element properties, material properties, local coordinate systems, and spatial variations of initial contact clearances (see *Distribution Definition*).

- Distributions should be defined at the level at which they are used. For example, if a distribution is used to define shell thicknesses, the distribution should be defined at the same level as the section definition that refers to it. If a distribution is used to define a material property, it should be defined at the model level with the material definition.

Examples

In the following examples most parameters and data lines are omitted for clarity.

Example 1		Notes
**PART*, NAME=PartA		
**NODE* ...		The mesh is defined on the part.
**ELEMENT* ...		
**SOLID SECTION*, ELSET=setA, MATERIAL=Mat1		Section assignment must appear within the part level if the mesh is defined on the part.
**SURFACE*, NAME=surf1 setB, ...	error	Element set `setB` is not defined at the part level.
**ELSET*, ELSET=setA		Sets and surfaces can be defined on the part since the mesh is defined on the part.
**NSET*, NSET=setA		
**SURFACE*, NAME=surf2 setA, ...		
**END PART*		
**ASSEMBLY*, NAME=Assembly-1		
**INSTANCE*, NAME=I1, PART=PartA		
**NODE*	error	Mesh and section assignment cannot be defined on the instance if they are defined on the part.
**ELEMENT*	error	
**SOLID SECTION*	error	
**ELSET*, ELSET=setA	error	Sets and surfaces cannot be redefined on the instance.
**NSET*, NSET=setA	error	
**SURFACE*, NAME=surf2	error	
**ELSET*, ELSET=setB		New sets and surfaces can be defined on the instance.
**NSET*, NSET=setB		
**SURFACE*, NAME=surf3 setA, ...		Set and surface definitions can refer to inherited sets.
**END INSTANCE*		
**END ASSEMBLY*		

In the second example the instances are meshed.

Example 2		Notes
**PART*, NAME=PartB		The **PART* and **END PART* options are required, even when the instance is meshed.
**END PART*		
**PART*, NAME=PartC		Section cannot be defined on the part if mesh is not defined on the part.
**SOLID SECTION*, ...	error	
**END PART*		
**ASSEMBLY*, NAME=Assembly-1		
**INSTANCE*, NAME=I1, PART=PartB		
**NODE* ...		The mesh is defined on the part instance.

Example 2		Notes
**ELEMENT ...*		
**SOLID SECTION*, ELSET=setA, MATERIAL=Mat1		Section assignment must appear within the same level as the mesh definition.
**ELSET*, ELSET=setA		Sets and surfaces are defined on the instance since the mesh is defined on the instance.
**NSET*, NSET=setA		
**SURFACE*, NAME=surf2 setA, ...		
**END INSTANCE*		
**INSTANCE*, NAME=I3, PART=PartC *<positioning data>*	error	The mesh and section must be defined for each instance since the part is not meshed.
**END INSTANCE*		
**END ASSEMBLY*		

Coordinate System Definitions

Abaqus provides several methods for defining local coordinate systems.

Nodal coordinate systems

You can define nodal coordinates in a local coordinate system (see *Specifying a Local Coordinate System in Which to Define Nodes*). The coordinate system can be defined within a part definition to define the nodes in that part. The nodal coordinate system definition remains in effect until another nodal coordinate system is defined within the same level or until the level ends.

Nodal transformations

A nodal transformation is used for applying loads and boundary conditions (see *Transformed Coordinate Systems*). It can be defined at the part or assembly level to define a local coordinate system for application of loads and boundary conditions or for the definition of linear constraint equations.

User-defined orientations

A user-defined orientation is used for defining material properties, coupling, connectors, and rebar (see *Orientations*). It can be defined at the part level for reference from a section, connector, rebar, or coupling definition. An orientation definition can also be used at the assembly level for reference from a connector or coupling definition.

Distributions

Distributions can be used to specify arbitrary spatial variations of local coordinate systems for continuum and shell elements (see *Orientations*). A distribution used by an orientation should be defined at the level in which the orientation is defined.

Normal definitions at nodes

Normals can be defined at nodes as part of the node definition for beam, pipe, and shell elements or with a user-specified normal definition (see *Normal Definitions at Nodes*). These normals can be defined at the part or assembly level.

A local coordinate system defined for a part using any of these methods is inherited by all instances of the part.

Translating and Rotating a Part Instance

The assembly's coordinate system is the global coordinate system. You can position part instances within the assembly by giving a translation and/or rotation relative to the global origin. Specify a translation by giving a translation vector. Specify a rotation by giving two points, *a* and *b*, to define a rotation axis plus a right-handed angular rotation around that axis.

Local coordinate systems defined within a part or part instance will be translated and rotated according to the specified positioning data, as shown in *Figure 5*. (In this figure details such as element and section definitions are omitted for clarity.) Results given in a local coordinate system are output in the transformed local system. Equations will also be translated and rotated according to the positioning data for an instance. All data within a part (or part instance) definition are defined relative to the part's local coordinate system; positioning data are applied to a part instance after everything within that instance is defined.

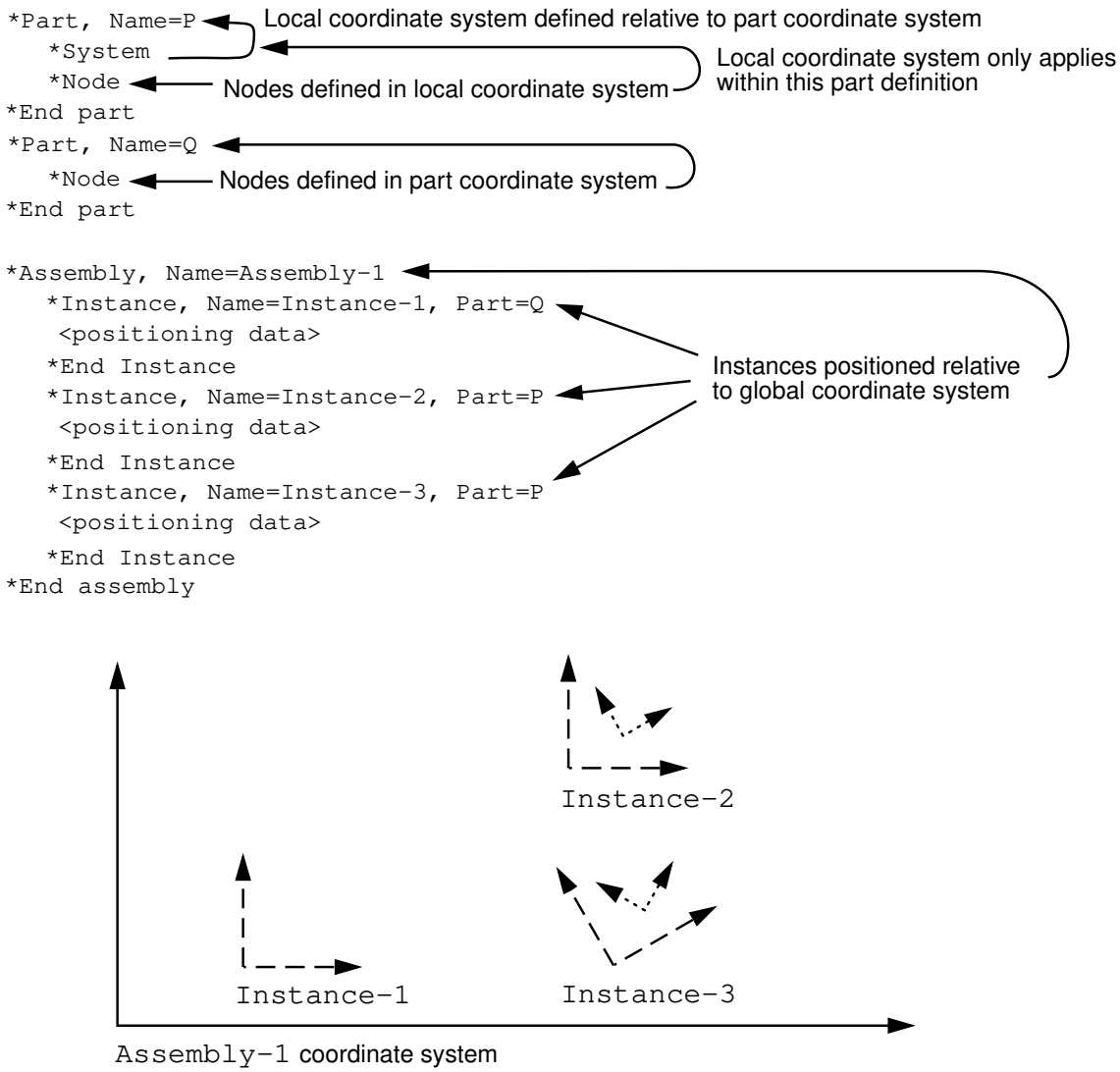


Figure 5: Defining local coordinate systems.

Limitations

The following capabilities are not supported in a model defined in terms of an assembly of part instances:

- *Mapping a Set of Nodes from One Coordinate System to Another*
- *Using Auxiliary Analyses to Generate Shape Variations*
- *Symmetric Model Generation*
- *Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*

- *Reading the Element Matrices from an Abaqus/Standard Results File*

The substructure database is not organized in terms of an assembly of part instances, so substructures cannot be generated from models that have an assembly defined. None of the substructure options are supported in models that have an assembly defined.

Input File Template

This template shows an input file that is written in terms of parts and assemblies with the part instances defined in this analysis. For templates that show how to import a part instance from a previous analysis to transfer model data and results, see *Transferring Results between Abaqus/Explicit and Abaqus/Standard* and *Transferring Results from One Abaqus/Standard Analysis to Another*.

```
*HEADING
*PART, NAME=Part-1
   Node, element, section, set, and surface definitions
   Connector and constraint definitions
*END PART
*PART, NAME=Part-2
   **The instance is meshed, so the part definition is empty
*END PART
*MATERIAL, NAME=mat1
   Suboptions and data lines to define this material
*ASSEMBLY, NAME=Assembly-1
   *INSTANCE, NAME=i1, PART=Part-1
    <positioning data>
      Additional set and surface definitions (optional)
   *END INSTANCE
   *INSTANCE, NAME=i2, PART=Part-2
    <positioning data>
      Node, element, section, set, and surface definitions
      Connector and constraint definitions
   *END INSTANCE
   Assembly-level set and surface definitions
   Assembly-level connectors and constraints
   Assembly-level reference node definitions
   Assembly-level rigid body definitions
*END ASSEMBLY
*MATERIAL, NAME=mat2
   Suboptions and data lines to define this material
*AMPLITUDE
*INITIAL CONDITIONS
*BOUNDARY
   Zero-valued boundary conditions
*PHYSICAL CONSTANTS
*CONNECTOR BEHAVIOR
   Suboptions and data lines to define this connector behavior
Interaction and interaction property definitions in Abaqus/Standard or Abaqus/Explicit
*STEP
   Loads and boundary conditions
   Predefined field definitions
   Output requests
   Contact interaction definitions in Abaqus/Explicit
*END STEP
```

Probability Density Function

Products: Abaqus/Explicit Abaqus/Viewer

References:

- *Discrete Element Method*
- *Discrete Particle Elements*
- **PARTICLE GENERATOR*
- **PARTICLE GENERATOR MIXTURE*
- **PROBABILITY DENSITY FUNCTION*
- **DISCRETE SECTION*

Overview

A probability density function:

- is used to define the statistical distributions of a continuous random variable; and
- can be defined for uniform, normal, log-normal, piecewise linear, and discrete distributions.

Introduction

There are many examples of randomness associated with data. Particle sizes in a granular media such as gravel are an example. Randomness observed in data can be described by statistical distributions. Pseudo-random numbers that are generated based on statistical distributions are used to capture randomness in data in a numerical simulation.

Applications

The size distribution of particle species generated by a particle generator can be described by statistical distributions.

Probability Density Function

A probability density function (PDF) describes the probability of the value of a continuous random variable falling within a range. If the random variable can only have specific values (like throwing dice), a probability mass function (PMF) would be used to describe the probabilities of the outcomes. The plot on the left in *Figure 1* shows a PDF for the random variable x. The probability that the random variable has a value in the range x and $x + dx$ is $f(x)\,dx$. The probability that the random variable x will be in the range $a \le x \le b$ is given by:

$$\Pr[a \le x \le b] = \int_a^b f(x)\,dx.$$

The probability that the random variable x is in the range $-\infty$ and ∞ is one; i.e.,

$$\int_{-\infty}^{\infty} f(x)\,dx = 1.$$

The area under the PDF curve is, therefore, always unity.

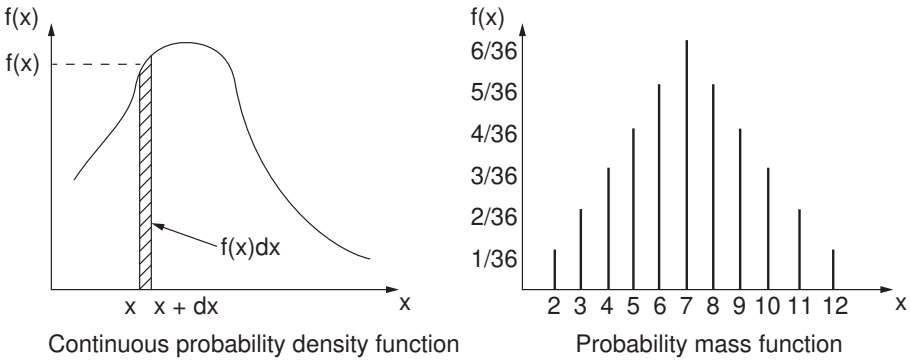


Figure 1: Probability distributions of continuous and discrete variables.

The plot on the right in *Figure 1* shows a PMF where the horizontal axis shows the specific values of the random variable and the vertical axis shows the corresponding probabilities.

Abaqus/Explicit supports uniform, normal (Gaussian), log-normal, piecewise linear, and discrete probability density functions. To define a probability density function, you must assign it a name and specify its type.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, NAME=*PDF_name*, TYPE=*PDF_type*

Uniform Probability Density Function

Uniform distributions (shown in *Figure 2*) have many applications, particularly in the numerical simulation of random processes. The following function describes a uniform probability density function for a random variable x between $x_{\min}$ and $x_{\max}$:

$$f(x) = \begin{cases} \dfrac{1}{x_{\max} - x_{\min}} & x_{\min} \le x \le x_{\max} \\ 0 & \text{otherwise} \end{cases}.$$

The mean is $\mu = \frac{x_{\min}+x_{\max}}{2}$, and the variance is $\sigma^2 = \frac{(x_{\max}-x_{\min})^2}{12}$. You specify $x_{\min}$ and $x_{\max}$ for the uniform distribution.

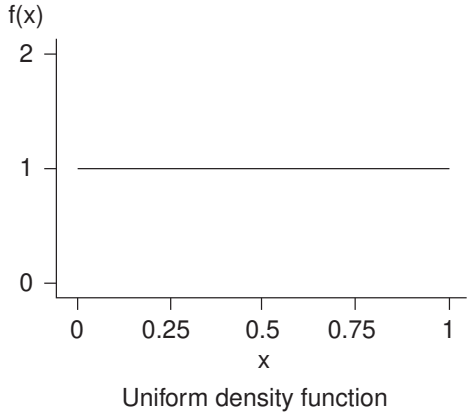


Figure 2: Uniform PDF.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, TYPE=UNIFORM

Normal Probability Density Function

Normal distributions (shown in *Figure 3*) have many applications in science and engineering; for example, errors in experimental measurements are often assumed to have a normal distribution. The following function describes a normal probability density function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]}.$$

The mean is $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$, and the variance is $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$. You specify the mean, μ, and standard deviation, σ, for the normal distribution.

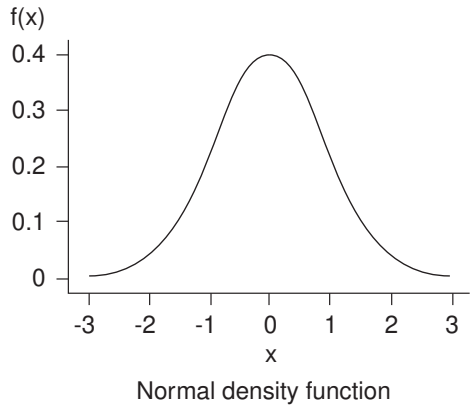


Figure 3: Normal PDF.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, TYPE=NORMAL

Log-Normal Probability Density Function

Log-normal distributions (shown in *Figure 4*) are used in describing many natural phenomena. They are commonly used to describe particle size distributions in soils. The following function describes a log-normal probability density function:

$$f(x) = \begin{cases} \dfrac{1}{\sqrt{2\pi}\sigma x} e^{\left[\frac{-[\ln(x)-\mu]^2}{2\sigma^2}\right]} & x > 0 \\ 0 & \text{otherwise} \end{cases}.$$

The mean, $\hat{\mu}$, and standard deviation, $\hat{\sigma}$, in the x space are related to μ and σ as follows:

$$\hat{\mu} = e^{\left[\mu + \frac{1}{2}\sigma^2\right]}$$

$$\widehat{\sigma^2} = e^{2\mu} e^{\sigma^2}\left(e^{\sigma^2} - 1\right),$$

where the parameters μ and σ are the mean and standard deviation in the $\ln(x)$ space; given by $\mu = \frac{1}{N}\sum_{i=1}^{N}\ln(x_i)$ and $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\ln(x_i) - \mu)^2}$, respectively. You specify the mean, $\hat{\mu}$, and the standard deviation, $\hat{\sigma}$, in the x space for the log-normal distribution.

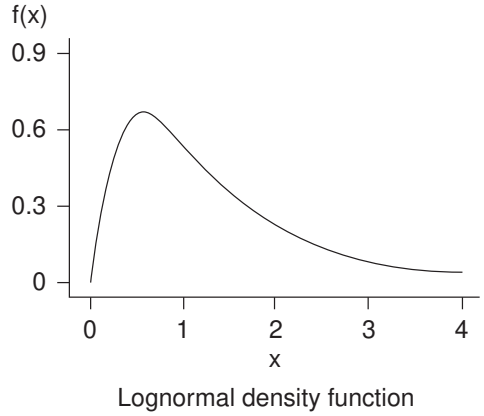


Figure 4: Log-normal PDF.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, TYPE=LOGNORMAL

Piecewise Linear Probability Density Function

A piecewise linear probability density function can be used to approximate general distributions that are not well represented by the other PDF forms discussed above. With a piecewise linear probability density function, you specify PDF values at discrete points. Abaqus/Explicit considers linear variations in the PDF between these points, as shown in *Figure 5*. The PDF is zero below the first data point and above the last data point.

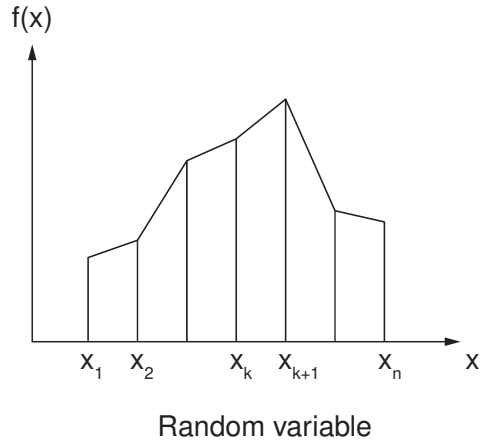


Figure 5: Piecewise linear PDF.

As mentioned earlier, the area under a PDF is unity. When the deviation of the area under the PDF is within 2% of unity, Abaqus/Explicit renormalizes the specified PDF data to achieve this requirement. This renormalization of data values allows you to specify relative PDF values that can be obtained from a histogram with small numerical errors. A histogram contains the data in the form of a table of random variable ranges and the percentage or number that falls within those ranges. The PDF is obtained from the histogram by normalizing the area under the histogram. As shown in *Figure 6*, you specify a table of the midpoint value of each range in the normalized histogram and the corresponding count:

$$\begin{Bmatrix} x_1 & c(x_1) \\ x_2 & c(x_2) \\ x_3 & c(x_3) \\ .. & .. \\ x_n & c(x_n) \end{Bmatrix}$$

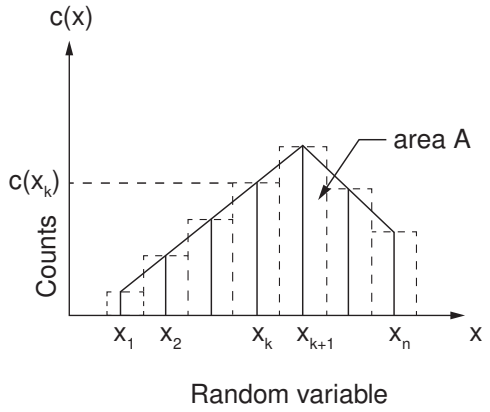


Figure 6: Histogram.

There may be situations where the random variable has continuous values over certain ranges and discrete values elsewhere. *Figure 7* shows the use of a piecewise linear probability density function to approximate such distributions where the discrete values are approximated by continuous random variables spanning a very narrow range of values (for example, the discrete value x_7 is approximated by the continuous range from x_5 to x_9).

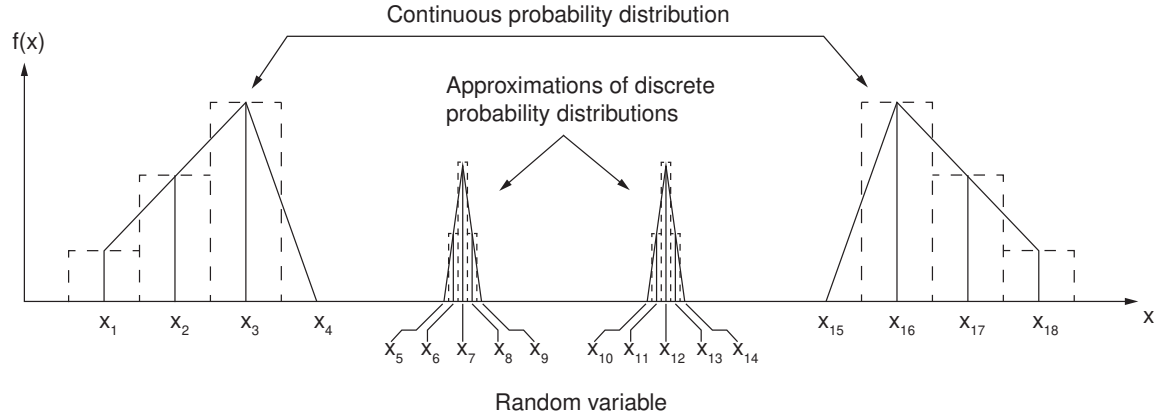


Figure 7: Approximating a discrete probability distribution using a piecewise linear PDF.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, TYPE=PIECEWISE LINEAR

Discrete Probability Density Function

Some applications have only certain specific outcomes. These applications can be represented by a discrete probability density function, as shown in *Figure 8*. A simple example is throwing of a pair of dice. Only the outcomes of 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12 are possible, with the probabilities of 1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, and 1/36, respectively. A very specific case of a discrete probability density function is the case when only one value occurs with the probability of 1. To specify a discrete probability density function, you provide a table of the specific values of the random variable along with the corresponding probability:

$$\begin{Bmatrix} x_1 & p_1 \\ x_2 & p_2 \\ x_3 & p_3 \\ \cdot\cdot & \cdot\cdot \\ x_n & p_n \end{Bmatrix}$$

The specified probabilities should sum up to 1. Small deviations can occur in some cases. When the sum of the probabilities is within two percent of unity, Abaqus/Explicit renormalizes the specified probabilities to ensure that they sum up to 1.

Input File Usage: **PROBABILITY DENSITY FUNCTION*, TYPE=DISCRETE

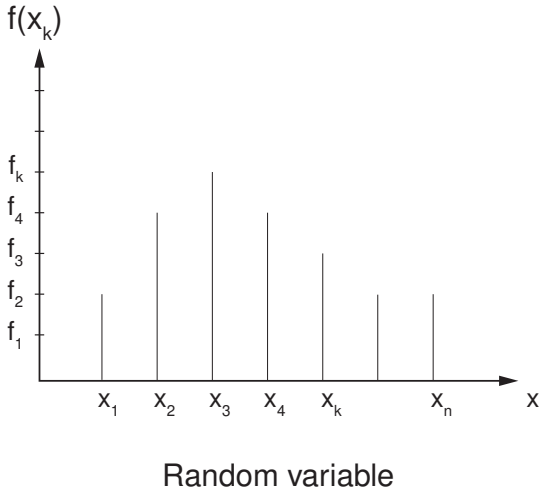


Figure 8: Discrete PDF.

Truncated Probability Density Function

The normal and log-normal probability density functions have open-ended characteristics. These PDFs can be truncated to enforce upper and lower bounds on the value of the random variable. *Figure 9* shows a truncated normal distribution $\hat{f}(x)$ where all values of the random variable $x < x_{min}$ and $x > x_{max}$ from the untruncated normal distribution $f(x)$ have been rejected.

$$\hat{f}(x) = \begin{cases} 0 & x < x_{min} \\ \dfrac{f(x)}{k} & x_{min} \le x \le x_{max} \\ 0 & x > x_{min} \end{cases},$$

where

$$k = \int_{x_{min}}^{x_{max}} f(x)\,dx.$$

The factor k represents the probability that the random variable is in the range from x_{min} to $x = x_{max}$ for the untruncated PDF.

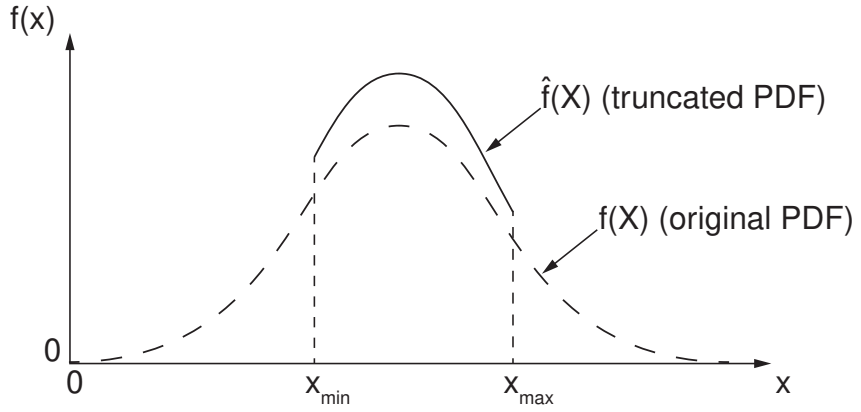


Figure 9: Truncated PDF.

You specify the lower and upper limits of the random variable along with the mean and standard deviation for these types of PDFs. The uniform and the piecewise linear distributions have lower and upper limits for the random variable built into the definition of the PDF and, therefore, do not require renormalization because of truncation.

Output

No output is available for probability density functions.

Limitations

Probability density functions are supported only for the size distributions of PD3D elements created using a particle generator.

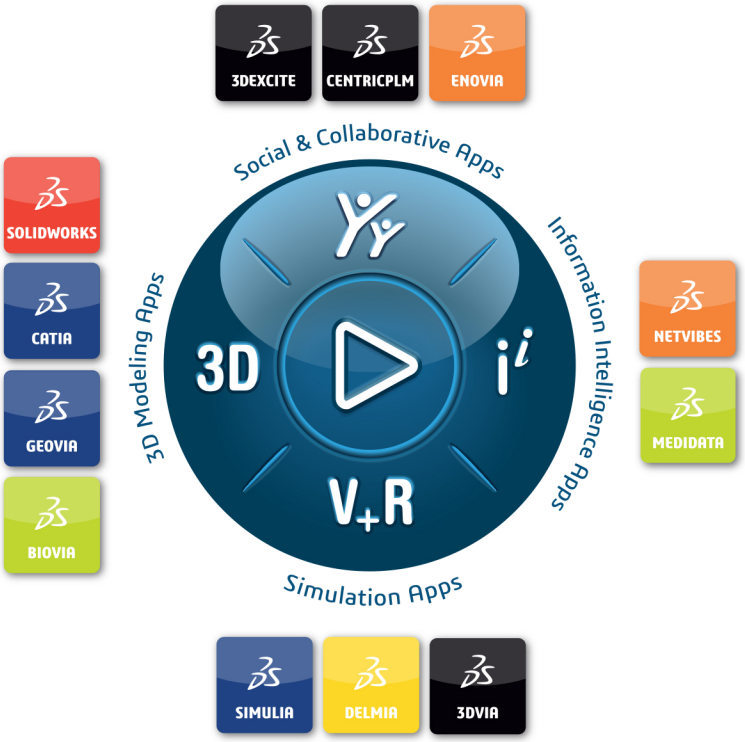
Input File Template

The following example illustrates the use of a probability density function for particle size distribution:

```
*HEADING
…
*PARTICLE GENERATOR, NAME=generator_name, TYPE=PD3D,
MAXIMUM NUMBER OF PARTICLES=number
**
*PARTICLE GENERATOR INLET, SURFACE=inlet_surf
**
*PARTICLE GENERATOR MIXTURE
gen_SET1, gen_SET2
**
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET1, TYPE=NORMAL
Data line to define PDF
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET2, TYPE=LOGNORMAL
Data line to define PDF
**
*DISCRETE SECTION, ELSET=gen_SET1
PDF_gen_SET1
*DISCRETE SECTION, ELSET=gen_SET2
PDF_gen_SET2
…
*END STEP
```

References

- Benjamin, J. R., and C. A. Cornell, "Probability, Statistics, and Decision for Civil Engineers," *McGraw-Hill*, 1970.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Numerical Recipes in Fortran 77, The Art of Scientific Computing," *University of Cambridge*, 1992.
- Saucier, R., "Computer Generation of Statistical Distributions," *Army Research Laboratory*, 2000.



Our **3D**EXPERIENCE® platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes is a catalyst for human progress. We provide business and people with collaborative virtual environments to imagine sustainable innovations. By creating virtual twin experiences of the real world with our **3D**EXPERIENCE platform and applications, our customers can redefine the creation, production and life-cycle-management processes of their offer and thus have a meaningful impact to make the world more sustainable. The beauty of the Experience Economy is that it is a human-centered economy for the benefit of all – consumers, patients and citizens.

Dassault Systèmes brings value to more than 300,000 customers of all sizes, in all industries, in more than 150 countries. For more information, visit **www.3ds.com**.

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes
17F, Foxconn Building,
No. 1366, Lujiazui Ring Road
Pilot Free Trade Zone, Shanghai 200120
China

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

DASSAULT SYSTEMES

©2026 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries.