SIMULIA

3DEXPERIENCE

ABAQUS 2025 FD04
CONSTRAINTS GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....4
Abaqus Constraints Guide....5
What's New....6
Kinematic Constraints....9
About Kinematic Constraints....10
Constraint Enforcement and Overlapping Constraints....12
Constraint Enforcement....12
Multiple Kinematic Constraints at a Node....12
Computational Cost for Constraint Solution in Abaqus/Explicit....14
Initial Conditions at Constrained Nodes....16
Multi-Point Constraints....17
Linear Constraint Equations....18
Defining a Linear Constraint Equation....18
Use with Transformed Coordinate Systems....19
Use within a Part....19
Prescribing a Nonhomogeneous Constraint....19
Constraint Forces and Global Equilibrium....20
Obtaining the Constraint Force....21
Defining a Constraint in a Deformed State....21
Reading the Data from an Alternate Input File....22
General Multi-Point Constraints....23
Identifying the Nodes Involved in the MPC....23
Use with Transformed Coordinate Systems....24
Defining Multiple Multi-Point Constraints at a Point....24
Using MPCS in Implicit Dynamic Analysis....24
Using Nonlinear MPCS in Geometrically Linear Abaqus/Standard Analysis....25
Defining MPCS in a User Subroutine....25
Reading the Data from an Alternate Input File....26
MPCS for Mesh Refinement....26
MPCS for Connections and Joints....32
MPCS for Transitions....45
Kinematic Coupling Constraints....50
Typical Applications....50
Defining the Constraint....51
Constraint Directions and Finite Rotations....52
Thermal Expansion of a Kinematic Coupling....52
Surface-Based Constraints....53
Mesh Tie Constraints....54
Defining a Tie Constraint for a Pair of Surfaces....54
Defining the Surfaces to Be Constrained....55
Specifying the Subset of Secondary Nodes to Be Constrained....56
Constraining the Rotational Degrees of Freedom....60
Constraining the Faces of a Cyclic Symmetric Structure in Abaqus/Standard....60

The Surface-Based Tie Constraint Formulation....61
Adjusting the Surfaces and Considering Offsets....63
Constraining a Surface to a Three-Dimensional Beam....67
Use of Tie Constraints in Nonmechanical Simulations....67
Tie Constraints Versus Tied Contact in Abaqus/Standard....69
Overlapping Constraints....69
Limitations....70
Coupling Constraints....71
Surface-Based Coupling Definitions....71
Typical Applications....72
Defining a Surface-Based Coupling Constraint....73
Specifying a Region of Influence....73
Kinematic Coupling Constraints....74
Distributing Coupling Constraints....76
Converting a Distributing Coupling Attached to a Single Element Facet to a Tie Constraint....82
Uniform Coupling Constraints....83
Limitations....83
Shell-to-Solid Coupling....84
Shell-to-Solid Coupling....84
Defining Shell-to-Solid Coupling....86
Computation of the Internal Coupling Constraints....87
Comments, Restrictions, and Modeling Recommendations for Shell-to-Solid Coupling....89
Mesh-Independent Fasteners....91
Introduction....91
Fastener Interactions....92
Specifying the Positioning Points, Projection Method, and Fastening Points....96
Specifying the Surfaces to Be Fastened....99
Defining the Radius of Influence....101
Defining the Weighting Method....101
Defining the Fastener Orientation....102
Coupling Fasteners to Surfaces....105
Defining Fastener Properties....106
Overconstraints in Fasteners Modeled with BEAMMPCS....108
Output....108
Embedded Elements....109
Introduction....109
Available Embedded Element Types....110
Specifying the Host Elements....110
Specifying the Embedded Elements....111
Specifying the Embedded Nodes....111
Defining Geometric Tolerances....112
Adjusting the Positions of Embedded Nodes....113
Use with Other Multiple Kinematic Constraints....113
Defining Surfaces on Embedded Elements....113
Limitations....113
Example....114
Template....115
Element End Release....116

Introduction....116
Equivalent MPCS....116
Identifying the Element End Involved in the Release....117
Identifying the Local Rotational Degrees of Freedom Involved in the Release....117
Use with Transformed Coordinate Systems....118
Reading the Data from an Alternate Input File....118
Overconstraint Checks....119
Overconstraints: General Remarks....120
Overconstraints Detected in the Model Preprocessor....122
Intersecting Tie Constraints....122
Tie Constraint inside a Rigid Body Constraint....122
Tie Constraint between Two Rigid Bodies....123
Tie Constraint between a Deformable and a Rigid Body....123
Intersecting Rigid Bodies....123
Tie Constraints and Boundary Conditions....124
Rigid Body Constraints and Boundary Conditions....125
Connector Elements and Rigid Bodies....126
Coupling Constraints and Rigid Bodies....126
Coupling Constraints and Boundary Conditions....127
Spot Welds and Rigid Bodies....127
Overconstraints Detected and Resolved during Analysis....127
Contact Interactions and Tie Constraints....128
Contact Interactions and Prescribed Boundary Conditions....129
Overconstraints Detected in the Equation Solver....130
Correcting an Overconstrained Model....133
Controlling the Overconstraint Checks....133
Bypassing the Overconstraint Checks....133
Preventing Automatic Redundant Constraint Resolution....133
Changing the Frequency of the Overconstraint Checks....134
Stopping the Analysis When Overconstraints Are Detected....134

Trademarks and Legal Notices

Trademarks

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

Legal Notices

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

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

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

© Dassault Systèmes Americas Corp., 2025.

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

Abaqus Constraints Guide

The Abaqus Constraints Guide describes how to model kinematic relationships between points and surfaces.

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.

You can define the following types of kinematic constraints:

- Linear multi-point constraints in the form of equations.
- Multi-point constraints that specify linear or nonlinear constraints between nodes.
- In Abaqus/Standard kinematic coupling constraints that allow general node-by-node specification of constrained degrees of freedom.
- Surface-based tie constraints that tie two surfaces together.
- Surface-based coupling constraints that constrain a group of nodes located on a surface to a reference node.
- Surface-based shell-to-solid coupling couples an edge-based surface on a three-dimensional shell element mesh to an element- or node-based surface on a three-dimensional solid mesh.
- Mesh-independent fasteners (such as spot welds) can bond together two or more surfaces.
- Embedded elements that lie embedded in a group of "host" elements in a model.
- In Abaqus/Standard local rotational degree of freedom or a combination of local rotational degrees of freedom can be released at one or both ends of a beam element.

What's New

This page describes recent changes in Abaqus Constraints.

2025 FD01

Coupling of Reference Node Rotations for Distributing Coupling Constraints

For distributed couplings Abaqus/Explicit can now automatically determine whether to couple the reference node rotations to cloud node translations only or to both cloud node translations and rotations.

Benefits: This enhancement simplifies the use of distributed couplings and facilitates accurate, efficient solutions.

In earlier releases, Abaqus/Explicit always coupled reference node rotations to cloud node translations only, an approach that has undesired characteristics if all cloud nodes are positioned along a line. In this release, Abaqus can form valid coupling constraints even if cloud nodes are collinear. The new setting takes control of which terms are used in these couplings by default.

Now, Abaqus/Explicit can form valid coupling constraints if cloud nodes have rotational degrees of freedom regardless of the configuration of the cloud nodes. The new default setting takes control of which terms are used in these couplings by default.

The distributing coupling constraint in Abaqus/Explicit has been enhanced such that rotation of the reference node is coupled to cloud node rotations (if they exist) if the cloud nodes are nearly collinear. This capability already existed in Abaqus/Standard.

The left image below shows the results in earlier releases of Abaqus/Explicit. Abaqus/Explicit did not form a valid coupling between the reference node rotation and cloud node translations because of collinearity of the cloud nodes. Therefore, the four cloud nodes and the adjacent shell mesh do not rotate even though the reference node is rotating. The right image illustrates the new behavior where Abaqus is able to form a valid coupling between the reference node rotation and the cloud system. Here the cloud node rotations are included in this coupling. The shell mesh spins around the cloud nodes consistent with the reference node rotation, which is the desired behavior.

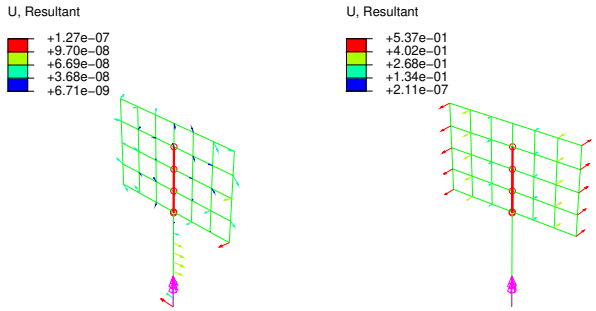


Figure 1: Comparison of rotational coupling behavior in earlier releases (left) with 2025 FD01 (right).

By default, cloud node rotations in both Abaqus/Standard and Abaqus/Explicit now automatically influence reference node rotations only if the cloud nodes are deemed to be nearly collinear. You can dictate whether or not the reference node rotations are influenced by the cloud node rotations. You need to take care in choosing the appropriate behavior. Including these extra terms when they are not required to form a valid constraint can

increase computation time without having a significant effect on results. However, not including these extra terms when they are required to form a valid constraint can lead to poor results.

For more information, see *Controlling Whether Cloud Node Rotations Influence Reference Node Rotation* and **DISTRIBUTING*.

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 Constraints Guide*.

2024 FD01

Conversion of Select Distributed Couplings to Tie Constraints

Abaqus/Explicit now converts distributing couplings to a tie constraint by default when the distributing couplings have cloud nodes associated with a single element facet and when the reference node of the coupling is not a node of a connector element.

Benefits: This enhancement avoids the creation of overlapping distributed couplings (which introduce a small amount of artificial mass for numerical purposes) when modeling fasteners using solid elements.

Some workflows involve fastening two surfaces together with the stiffness of the connection represented by a hexahedral element instead of a connector element, as shown in *Figure 2*. Each node of the hexahedral element acts as a reference node of a distributing coupling whose cloud nodes are associated with a single shell element, as shown in *Figure 3*. This approach results in some shell nodes participating in two or four distributing couplings. Abaqus/Explicit generally introduces artificial mass to nodes acting as clouds nodes in multiple distributing couplings. By changing the default behavior for automatic conversion of distributing couplings to tie constraints, Abaqus/Explicit reduces the likelihood of adding artificial mass to a model.

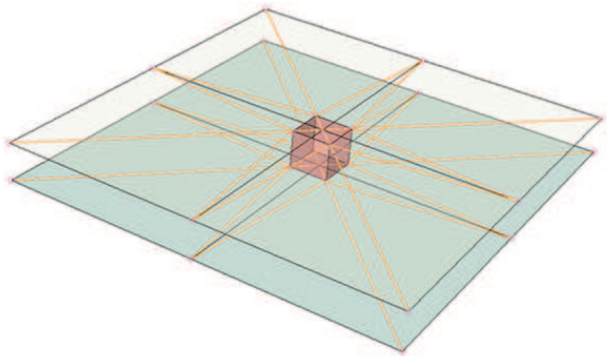


Figure 2: A fastener construct involving a hexahedral element representing the faster stiffness.

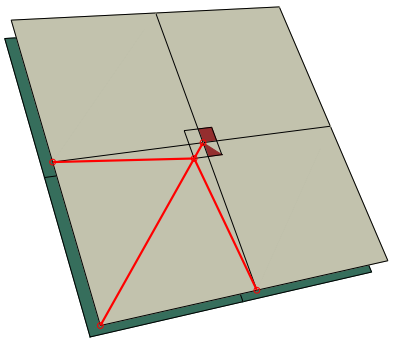


Figure 3: Highlighted connectivity for one of eight user-defined distributing couplings associated with a fastener connection.

Constraint coefficients are somewhat different for distributed couplings than they are for tie constraints in such cases. *Figure 4* provides perspective on these differences. A downward unit force is applied at node 101. Node 101 is connected to a single square face with a distributed coupling for the case on the left versus a tie constraint for the case on the right. The four nodes of the square face are fixed with boundary conditions. Node 101 corresponds to the position one-tenth along the diagonal line from nodes 1 to 4. The vertical reaction forces (RF3) for nodes 1 through 4 are 0.65, 0.25, 0.25, and -0.15 for the distributing coupling case; and for the tie constraint case these reaction forces are 0.81, 0.09, 0.09, and 0.01, respectively. These sets of reaction forces are statically equivalent (both sets of reaction forces sum to 1.0 and do not generate a moment about the location of node 101). Because the constraint coefficients are different in the two approaches, some differences in the local behavior around the constraint are expected due to this conversion.

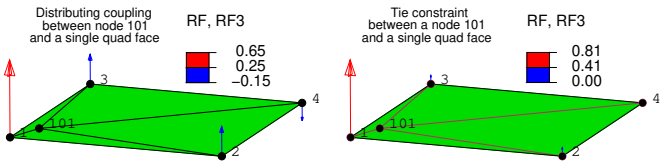


Figure 4: Comparison of reaction forces for a distributed coupling versus a tie constraint acting between node 101 and a single quad face, with unit downward load applied at node 101 and boundary conditions acting at nodes 1 through 4.

You can override this conversion from distributed couplings to tie constraints if desired.

The recommended technique for modeling compliant fasteners is to use connector elements instead of hexahedral "nuggets." This enables the use of larger numbers of cloud nodes per coupling, which reduces mesh sensitivity without having nodes act as cloud nodes in multiple overlapping distributed couplings.

For more information, see *Converting a Distributing Coupling Attached to a Single Element Facet to a Tie Constraint*.

Kinematic Constraints

You can define several types of kinematic constraints, including multi-point constraints, kinematic coupling constraints, and tie constraints.

In this section:

- *About Kinematic Constraints*
- *Constraint Enforcement and Overlapping Constraints*
- *Initial Conditions at Constrained Nodes*

About Kinematic Constraints

Overview

You can define several types of kinematic constraints, including multi-point constraints, kinematic coupling constraints, and tie constraints.

The following types of kinematic constraints can be defined:

Equations

Linear multi-point constraints can be given in the form of an equation (see *Linear Constraint Equations*).

Multi-point constraints

Multi-point constraints (MPCs) specify linear or nonlinear constraints between nodes. These relations between nodes can be the default types that are provided in Abaqus or, in Abaqus/Standard, can be coded in the form of a user subroutine. *General Multi-Point Constraints* explains the use of MPCs and lists the available default constraints.

Kinematic coupling

In Abaqus/Standard a node or group of nodes can be constrained to a reference node. Similar to multi-point constraints, the kinematic coupling constraint allows general node-by-node specification of constrained degrees of freedom (see *Kinematic Coupling Constraints*).

Surface-based tie constraints

Two surfaces can be tied together. Each node on the first surface (the secondary surface) will have the same values for its degrees of freedom as the point on the second surface (the main surface) to which it is closest (see *Mesh Tie Constraints*). In the case of surface elements tied to a beam surface, the offset distances between the surface elements and the beam are used in the definition of constraints, which include the rotational degrees of freedom of the beam.

Surface-based coupling constraints

A group of nodes located on a surface can be constrained to a reference node. This constraint may be kinematic, in which the group of coupling nodes can be constrained to the rigid body motion defined by the reference node, or distributing, in which the group of coupling nodes can be constrained to the rigid body motion defined by the reference node in an average sense (see *Coupling Constraints*).

Surface-based shell-to-solid coupling

An edge-based surface on a three-dimensional shell element mesh can be coupled to an element- or node-based surface on a three-dimensional solid mesh. The coupling is enforced by the creation of an internal set of distributing coupling constraints (see *Shell-to-Solid Coupling*).

Mesh-independent spot welds

Two or more surfaces can be bonded together using fasteners such as spot welds (see *Mesh-Independent Fasteners*). Distributed coupling constraints are created on each of the connected surfaces. The connection is modeled independent of the mesh.

Embedded elements

An element or a group of elements can be embedded in a group of host elements (see *Embedded Elements*). Abaqus will search for the geometric relationships between nodes on the embedded elements and the host elements. If a node on an embedded element lies within a host element, the degrees of freedom at the node will be eliminated by constraining them to the interpolated values of the degrees of freedom of the host element. Host elements cannot be embedded themselves.

Release

In Abaqus/Standard a local rotational degree of freedom or a combination of local rotational degrees of freedom can be released at one or both ends of a beam element (see *Element End Release*).

Boundary conditions are also a type of kinematic constraint in stress analysis because they define the support of the structure or give fixed displacements at nodal points. Specification of boundary conditions is discussed in *Boundary Conditions*.

Connector elements can be used to impose element-based kinematic constraints for mechanism-type analysis. See *About Connectors*.

Contact interactions, described in *About Contact Interactions* can be used to enforce constraints between bodies that come into contact. Contact interactions can be used in mechanical as well as coupled thermomechanical, coupled thermal-electrical-structural, and coupled pore fluid-mechanical analysis.

References:

- *Constraint Enforcement and Overlapping Constraints*

Constraint Enforcement and Overlapping Constraints

Overview

You can define several types of kinematic constraints, including multi-point constraints, kinematic coupling constraints, and tie constraints.

References:

- *About Kinematic Constraints*

Constraint Enforcement

Abaqus enforces constraints on incremental motion at every increment during the solution procedure. A variety of constraint enforcement methods are used by Abaqus. To successfully run simulations, you do not need to be aware of which constraint methods are used and how they work. However, a basic understanding of the constraint methods used might offer you some context for the diagnostic information that Abaqus provides.

Abaqus typically uses a penalty method to enforce contact constraints, although Lagrange multiplier and augmented Lagrange constraint methods are also available to enforce contact constraints in Abaqus/Standard (see *Contact Constraint Enforcement Methods in Abaqus/Standard* and *Contact Constraint Enforcement Methods in Abaqus/Explicit*). Contact penalty stiffness is treated by the central-difference time integration in Abaqus/Explicit without triggering implicit corrections (see *Central-Difference Time Integration with Implicit Corrections*).

Abaqus typically uses the elimination method to strictly enforce noncontact constraints. The elimination method eliminates degrees of freedom of secondary nodes in favor of degrees of freedom of main nodes. The elimination method in Abaqus/Explicit involves the implicit correction phase of central-difference time integration. Exceptions for which noncontact constraints use other constraint methods include the following:

- Distributed coupling constraints in Abaqus/Standard use a Lagrange multiplier constraint method to strictly enforce that a reference node moves according to the weighted average motion of cloud nodes. Boundary conditions or other constraints also typically involve the reference node, so using an elimination method to enforce distributed couplings is undesirable.
- Distributed coupling constraints in Abaqus/Explicit use a specialized form of implicit correction equations because distributed coupling constraints often participate in other boundary conditions, constraints, or connector elements.
- Conflicting constraint requirements cause Abaqus/Explicit to introduce stiff penalty springs. These springs are treated during the implicit correction phase of central-difference time integration, such that the stiff springs do not influence the size of the stable time increment. Large constraint systems involving conflicting constraints can degrade Abaqus/Explicit performance. For further discussion related to overlapping constraints, see *Multiple Constraints in Abaqus/Standard*, *Multiple Constraints in Abaqus/Explicit*, and *Computational Cost for Constraint Solution in Abaqus/Explicit*.

Multiple Kinematic Constraints at a Node

Overconstraint Checks describes the overconstraint checks and the automatic resolution of some overconstraints performed in Abaqus/Standard.

It is possible to use a single node in several multi-point constraints, kinematic coupling constraints, tie constraints, and constraint equations. However, the constraint dependencies are handled differently in Abaqus/Standard and Abaqus/Explicit.

Multiple Constraints in Abaqus/Standard

In Abaqus/Standard kinematic constraints are usually imposed by eliminating degrees of freedom at the dependent (secondary) nodes. Once a variable has been eliminated, it cannot be referenced in any boundary condition or in any subsequent multi-point constraint, kinematic coupling constraint, tie constraint, or constraint equation. Abaqus/Standard automatically sorts MPC types BEAM, CYCLSYM, LINK, PIN, REVOLUTE, TIE, and UNIVERSAL to obtain a proper elimination order. For other constraint types, you must order constraint definitions such that constraints for which a variable (nodal degree of freedom) act as a main variable precede a constraint definition involving that variable acting as a secondary variable. Automated or manual sorting cannot overcome circular dependences or overconstraints associated with a variable acting as a secondary variable in multiple constraints.

Excessive chaining of multi-point constraints, kinematic coupling constraints, and constraint equations is not recommended and might result in a degradation in performance during analysis preprocessing. Whenever possible, it is best to relate the behavior of several nodes (grouped into a node set) to a single node by using one multi-point constraint, kinematic coupling constraint, or constraint equation.

Multiple Constraints in Abaqus/Explicit

Kinematic constraints in Abaqus/Explicit can be defined in any order without regard to constraint dependencies. Less stringent requirements pertaining to constraint ordering and overconstraints in Abaqus/Explicit compared to Abaqus/Standard are due to two factors: a more extensive automated constraint reordering algorithm for Abaqus/Explicit, and automatically switching to a penalty enforcement method to avoid overconstraints.

With the exception of constraints arising from kinematic contact pairs, Abaqus/Explicit solves for all kinematic constraints simultaneously. Thus, nodes involved in a combination of multi-point constraints, constraint equations, connector element kinematic constraints, rigid body constraints, and constraints due to boundary conditions will simultaneously satisfy these constraints as long as they are not conflicting. Redundant and closed loop constraints are acceptable but might lead to a system of constraint equations being solved each increment. This process to solve systems of equations can degrade computational performance if many nodes are involved in the overlapping system of constraints, as discussed in *Constraint Enforcement and Overlapping Constraints*.

Because the above constraints are enforced independent of contact constraints, you should use the penalty contact algorithm for nodes involved in both kinematic constraints and contact pair definitions. The penalty contact algorithm introduces numerical softening through the use of penalty springs and does not interfere with kinematic constraints. If a node that participates in a kinematic constraint is used in a kinematic contact pair, the contact constraint overrides the kinematic constraint in most situations. Except for rigid bodies, Abaqus/Explicit does not prevent you from defining these conditions; however, the results cannot be guaranteed. If a kinematic constraint is defined for a node on a rigid body, the penalty contact algorithm is required for all contact pairs involving the rigid body.

To obtain accurate reaction force and moment output from Abaqus/Explicit at nodes that are constrained by boundary conditions in addition to one or more of the kinematic constraints described above, it might be necessary to run the analysis in double precision. In such a situation, a double precision run also yields a better estimate of the work done by the reaction forces and moments, thereby providing a more accurate value of the energy due to the external work reported by Abaqus/Explicit.

Abaqus/Explicit uses a penalty method to solve for constraints in situations involving overconstraints. The penalties are weighted based on the masses of nodes participating in the constraint and the stable time increment. The penalty formulation attempts to satisfy the constraint approximately (that is, a very small lack of compliance exists after imposition of the constraint). One situation in which the penalty approach is used to solve the constraint is when secondary nodes of a tie constraint participate in other constraints such as multi-point constraints,

kinematic coupling constraints, constraint equations, connector elements, rigid body constraints, or constraints due to boundary conditions. In this case, the lack of compliance in the tie constraint is not carried across step boundaries; thus, noisy accelerations and energy imbalance might be observed at step boundaries for certain problems. An alternative modeling approach (such as reversing the main and secondary surfaces in the tie constraint) might switch to a different solution approach and resolve the inaccuracies.

In Abaqus/Explicit, when there are two or more overlapping distributed coupling constraints or overlapping distributed coupling and tie constraints and the elements underlying the participating surfaces have very low densities, the lack of compliance might result in an inaccurate solution. Specifying reasonable density values for underlying elements might reduce the lack of compliance and improve solution accuracy.

Abaqus/Explicit always uses a geometrically nonlinear formulation for the enforcement of kinematic constraints. This is the case even when you have designated a particular analysis step as being geometrically linear. Thus, results in these geometrically linear analyses could be hard to interpret, particularly when the deformations are large and a geometrically nonlinear formulation should have been used for the step.

Computational Cost for Constraint Solution in Abaqus/Explicit

In Abaqus/Explicit some constraints and actions might be enforced through implicit solution by forming an operator matrix. Several of the constraints and actions that require such an approach include:

- Connector elements, except those that have no behavior defined.
- Constraints where secondary nodes have conflicting constraints; for example, a secondary node of a tie constraint that is also a secondary node in another constraint.
- A secondary node of a constraint that has boundary conditions applied.
- Releasing some degrees of freedoms in surface-based constraints.

Each of these constraints or actions might involve only a few nodes and small operator sizes. However, a chain of such constraints or actions can include many nodes requiring large operators.

Abaqus/Explicit examines all such systems and categorizes them according to the number of operations required for the solution. For moderate-sized systems that do not cause any appreciable performance degradation, an informational message is output to the status file. Nodes that participate in such systems are output using a node set `InfoNsetSystNodes`*N*, where *N* is the ordinal position of the system in a list of moderately sized operators. Nodes that most probably caused the implicit solution to be invoked are output using the node set `InfoNsetSystNodes`*N*`_Trigger`. The node sets are output to the output database, and a list of node set names is tabulated in the status file. The table also contains an estimate of the number of operations required for solution and the number of variables involved in the operator.

For systems of intermediate sizes, a warning message is printed to the status file. Node sets `WarnNsetSystNodes`*N* and `WarnNsetSystNodes`*N*`_Trigger` are tabulated to the status file along with an estimate of the number of operations and the number of variables. The node sets are also output to the output database.

For large systems that cause severe performance degradation, `ErrNsetSystNodes`*N* and `ErrNsetSystNodes`*N*`_Trigger` are output along with an error message in similar fashion. You can change the error message to a warning message.

Figure 1 shows an example of a model with an implicit system of constraint equations associated with the highlighted nodes. This region has redundant surface-based tie specifications with opposite main and secondary surface roles.

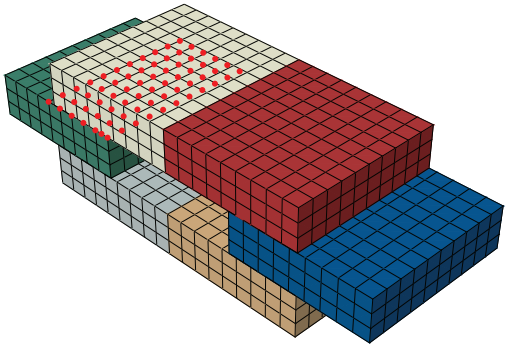


Figure 1: Highlighted diagnostic node set.

After running this job, the status file shows the following output:

```
------------------------------------------------------------------------
CONSTRAINT SYSTEM DIAGNOSTIC OUTPUT
------------------------------------------------------------------------

***WARNING: Abaqus/Explicit has formed at least one moderately-sized implicit
            system of equations associated with enforcing constraints. Solving
            this system of equations would  moderately degrade performance.
            Please refer to Abaqus/Explicit constraints documentation about
            implicit systems of equations for suggestions on how to reduce the
            size of, or avoid, these systems of equations.  Additional details
            on individual systems of equations follows:

System              Solve           Number of       Node set             TriggerNodeSet
 Rank             Operations        Variables
-------------------------------------------------------------------------------------------------
    1             140499              201          InfoNsetSystNodes1   InfoNsetSystNodes1_Trigger
```

This output informs you about a moderately sized implicit system that has 201 degrees of freedom and would require 140,499 operations to solve during each iteration of the solution. The diagnostic message for this implicit system of equations is informational (rather than a warning or error message) because the number of computational operations to solve the system of equations is moderate. The nodes highlighted in *Figure 1* are associated with an automatically generated node set referred to by the diagnostic message. The `TriggerNodeSet` is a set of nodes that most likely have constraints that require implicit solution. In general, studying constraints and connections associated with such node sets might help you understand the source of overlap associated with the implicit system of equations. Avoiding redundant tie specifications enable this simulation to run about 30% faster. Larger speedups can occur for larger implicit systems of constraint equations.

Input File Usage: Use the following option to change the error message to a warning message:

**DIAGNOSTICS*, LARGE IMPLICIT SYSTEM=WARNING

Initial Conditions at Constrained Nodes

Overview

When you prescribe initial conditions at a set of nodes that are constrained kinematically, Abaqus processes the prescribed values to determine an initial value.

You should not think of initial conditions as boundary conditions at the beginning of the analysis. When you prescribe initial conditions at a set of nodes that are constrained kinematically, Abaqus processes the prescribed values to determine an initial value that is then redistributed to the nodes involved in the constraints in a kinematically consistent manner using a “mass” weighted averaging method: the initial value prescribed at each node involved in the constraint is weighted with the corresponding “mass” at the node. Consequently, the values of the initial conditions that you specified at the nodes are recomputed, and in many cases the output of the prescribed quantity at these nodes at the beginning of the analysis will be different from the values that you have specified. Correct modeling practices consist of specifying initial conditions at all nodes involved in the constraints in a manner consistent with constraint itself.

This behavior is probably best understood with a simple example. Consider a model consisting of two nodes each with a mass of 1.0 constrained by boundary conditions in global directions 2 and 3 and allowed to move freely along the global 1-direction while their relative motions is also constrained via a rigid connection such as a BEAM connector. Assume that you have specified an initial translational velocity along the global 1-direction only at the first node of 10.0 units and you have not specified initial conditions at the second node. Consequently, Abaqus will consider that the initial velocity is 0.0 at the second node. This initial velocity field is inconsistent with the kinematic constraint enforced by the BEAM connector because the constraint would be violated if the initial conditions were to be enforced even for an infinitesimally short period of time. The outcome is that Abaqus will compute an initial velocity field that would redistribute the momentum of the first node in a manner consistent with the constraint. In this particular example, the net effect is that both nodes will end up with an initial velocity of 5.0 units along the global 1-direction. Most likely, this is not what you intended. Correct modeling practice in this case would be to specify an initial velocity of 10.0 units at both nodes involved in the constraint. In this case Abaqus still recomputes the initial values, but the outcome would be an initial velocity of 10.0 units at both nodes, as intended.

The same principle applies in more complicated modeling situations. For example, if you prescribe initial translational velocities at the nodes of the kinematic constraint, an average translational velocity of the constrained nodes is computed by calculating a mass weighted average of the velocities at the individual nodes. Depending on the nature of the kinematic constraint, initial translational velocities at the nodes of a constraint might also give rise to an average rotational velocity about the center of mass of the constraint. The velocity of each individual node of the constraint is then recomputed from the average translational and rotational velocities at the center of mass of the constraint. The “mass”-type quantity used in the weighting varies depending on the nature of the prescribed quantity: if the initial condition is prescribed on the rotational velocities, the rotary inertia at the nodes is used in the weighting; if temperature initial conditions are prescribed, the thermal capacitance at the nodes is used in the weighting; and so on.

In all cases, you should specify initial conditions at all nodes involved in the constraint that are consistent with the constraint. This is typically accomplished by specifying the same initial conditions at all nodes involved in the constraint.

Multi-Point Constraints

You can model kinematic relationships between points in the form of equations and as linear or nonlinear constraints between nodes, and, in Abaqus/Standard, by coupling a group of nodes to a reference node.

In this section:

- *Linear Constraint Equations*
- *General Multi-Point Constraints*
- *Kinematic Coupling Constraints*

Linear Constraint Equations

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Kinematic Constraints*
- **EQUATION*
- *Defining equation constraints*

Overview

Linear multi-point constraints can be given in the form of a linear equation involving nodal degrees of freedom.

A linear multi-point constraint requires that a linear combination of nodal variables is equal to zero; that is,

$A_1 u_i^P + A_2 u_j^Q + \cdots + A_N u_k^R = 0$, where u_i^P is a nodal variable at node P, degree of freedom i; and the A_n are coefficients that define the relative motion of the nodes.

In Abaqus/Explicit linear constraint equations can be used only to constrain mechanical degrees of freedom.

Defining a Linear Constraint Equation

A linear constraint equation is defined in Abaqus by specifying:

- the number of terms in the equation, N;
- the nodes, P, and the degrees of freedom, i, corresponding to the nodal variables u_i^P; and
- the coefficients, A_n.

For example, to impose the equation

$$u_3^5 = u_1^6 - u_3^{1000},$$

you would first write the equation in the standard form,

$$u_3^5 - u_1^6 + u_3^{1000} = 0.$$

There are three terms in this equation (N=3). P=5, i=3, A_1=1.0, Q=6, j=1, A_2=–1.0, R=1000, k=3, and A_3=1.0.

In Abaqus/Standard the first nodal variable specified (u_i^P corresponding to A_1) will be eliminated to impose the constraint (in the above equation constraint, degree of freedom 3 at node 5 will be eliminated); therefore, it should not be used to apply boundary conditions, nor should it be used in any subsequent multi-point constraint, kinematic coupling constraint, tie constraint, or equation constraint (see *About Kinematic Constraints*). In addition, the coefficient A_1 should not be set to zero. These restrictions do not apply in Abaqus/Explicit.

In Abaqus/Standard a linear multi-point constraint cannot be used to connect two rigid bodies at nodes other than the reference nodes, since multi-point constraints use degree-of-freedom elimination and the other nodes on a rigid body do not have independent degrees of freedom. In Abaqus/Explicit a rigid body reference node or any other node on a rigid body can be used in an equation constraint definition.

Input File Usage: **EQUATION*
N
P, *i*, A_1, *Q*, *j*, A_2, *etc.*

For example, the following input could be used to define the equation constraint above:

```
*EQUATION
 3
 5, 3, 1.0, 6, 1, -1.0, 1000, 3, 1.0
```

Either node sets or individual nodes can be specified as input. If node sets are used, corresponding set entries are matched to each other. If sorted node sets are given as input, you must ensure that the nodes are numbered such that they match up with each other correctly once sorted. The nodes in an unsorted node set will be used in the order that they are given in defining the set (see *Node Definition*).

If the first entry is a single node, subsequent entries must be single nodes. If the first entry is a node set, subsequent entries can be either node sets or single nodes. The latter option is useful if a degree of freedom at each of a set of nodes depends on a degree of freedom of a single node, such as might occur in certain symmetry conditions or in the simulation of a rigid body.

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Equation**

The nodes must be specified as sets. The first set can contain one or more nodes. Subsequent sets can contain multiple nodes if all the sets contain an equal number of nodes.

Use with Transformed Coordinate Systems

If a local coordinate system (*Transformed Coordinate Systems*) is defined for any node involved in the equation, the variables at that node appear in the equation in the local system.

Use within a Part

If an equation constraint is defined at the part (or part instance) level, the nodal variables are transformed initially according to the positioning data given for each instance of the part (see *Assembly Definition*).



Note:

Equation constraints cannot be defined at the part (or part instance) level in Abaqus/CAE.

Prescribing a Nonhomogeneous Constraint

It is sometimes necessary to impose a constraint in the form

$$A_1 u_i^P + A_2 u_j^Q + \cdots + A_N u_k^R = \hat{u},$$

where $\hat{u}(t)$ is a prescribed value that may vary with time, *t*. This is easily done by rewriting the equation as

$$A_1 u_i^P + A_2 u_j^Q + \cdots + A_N u_k^R - \hat{u}_m^Z = 0$$

and introducing a node, *Z*, that is not attached to any element in the model. Choosing $\hat{u}_m^Z$ to be some convenient degree of freedom *m* at node *Z* allows the prescribed value $\hat{u}(t)$ to be imposed through a boundary condition specification. If necessary, an amplitude reference can be provided to give the variation with time (see *Boundary Conditions*); such an amplitude reference is required in Abaqus/Explicit for prescribed displacements.

For example, assume that node 1000 in the example above is a "dummy" node that appears only in this equation and is not attached to any other part of the model. Defining a boundary condition to constrain degree of freedom 3 at node 1000 to −12.5 would impose the constraint

$$u_3^5 - u_1^6 = 12.5.$$

Constraint Forces and Global Equilibrium

Linear constraint equations introduce constraint forces at all degrees of freedom appearing in the equations. These forces are considered external, but they are not included in reaction force output. Therefore, the totals provided at the end of the reaction force output tables might reflect an incomplete measure of global equilibrium.

To illustrate this behavior, consider a spring-supported beam subjected to a concentrated load as shown in *Figure 1*. The static reaction forces are $R_y^C = -3$ and $R_y^D = -6$. In *Figure 2* the same structure is subjected to the additional linear constraint equation $u_y^A - u_y^B = 0$, which constrains the beam to remain horizontal. This introduces constraint forces $F_y^A = 1.5$ and $F_y^B = -1.5$, and the new reaction forces are $R_y^C = R_y^D = -4.5$. These reaction forces produce a global force balance in the Y-direction, but since the constraint forces are not included in reaction force output, the global moment balance about point A cannot be verified.

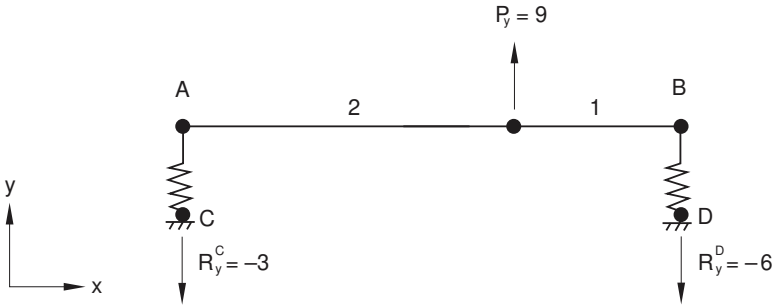


Figure 1: Beam with no linear constraints.

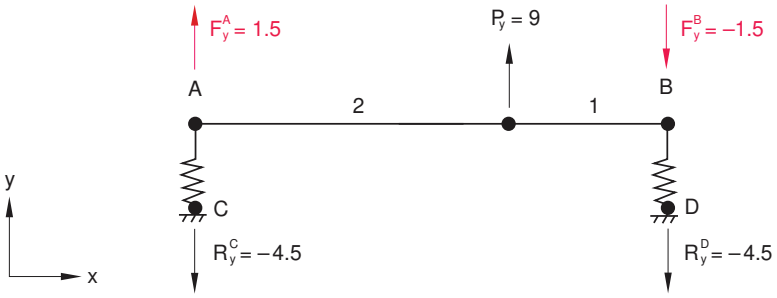


Figure 2: Beam with linear constraint $u_y^A - u_y^B = 0$. Constraint forces F_y^A and F_y^B are not included in reaction force output.

The global force balance can also be incomplete. This is demonstrated in *Figure 3*, where a pulley connection between nodes A and B is represented by the linear constraint equation $u_y^A - u_x^B = 0$. The constraint forces at

the pulley, F_x and F_y, are not included in the reaction force output, producing incomplete global force balances in both the *X*- and *Y*-directions.

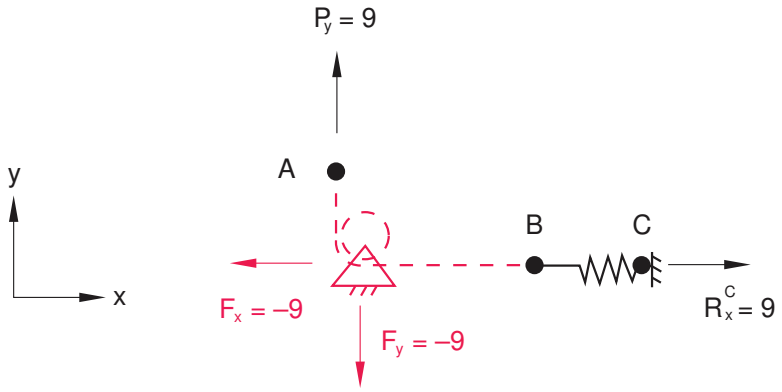


Figure 3: Pulley connection represented by the linear constraint $u_y^A - u_x^B = 0$. Constraint forces F_x and F_y are not included in reaction force output.

Obtaining the Constraint Force

The linear constraint generates constraint forces at all the degrees of freedom involved in the equation. For a given constraint equation these forces are proportional to their respective coefficients. To find the constraint forces, introduce a node *Z* that is not attached to any element in the model; rewrite the constraint equation as

$$A_1 u_i^P + A_2 u_j^Q + \cdots + A_N u_k^R - A_1 \hat{u}_m^Z = 0;$$

and specify a zero displacement boundary condition at degree of freedom *m* of node *Z*. The reaction force obtained at node *Z* will be equal to the constraint force acting at node *P* in degree of freedom *i*. The constraint force in any term with coefficient A_K in the constraint equation is obtained by multiplying the constraint force at node *P* in degree of freedom *i* with the ratio A_K/A_1. For example, if the equation is

$$u_3^5 - u_3^6 = 0$$

and the forces in the constraint are needed, the equation can be rewritten as

$$u_3^5 - u_3^6 - u_3^{1000} = 0,$$

where node 1000 is the fixed “dummy” node. Since the coefficient of u_3^5 is the opposite of the coefficient of u_3^{1000}, the constraint force at node 5 is the same as the reaction force at node 1000. Since the coefficient of u_3^6 is the same as the coefficient of u_3^{1000}, the constraint force at node 6 is the opposite of the reaction force at node 1000.

Defining a Constraint in a Deformed State

You might want to impose an equation starting at a certain point in the analysis:

$$A_1 \Delta u_i^P + A_2 \Delta u_j^Q + \ldots + A_N \Delta u_k^R = 0,$$

where Δu represents the change in displacement after time t_0. The equation can be rewritten as

$$A_1 u_i^P + A_2 u_j^Q + \ldots + A_N u_k^R - \hat{u}_m^Z = 0,$$

where, again, node Z is not attached to any element in the model. Prior to time t_0 (which is assumed to be at the end of a step), degree of freedom m of node Z is left unrestrained. After time t_0 further changes in $\hat{u}_m^Z$ are restrained in Abaqus/Standard by applying a boundary condition fixing the degree of freedom at its current values at the start of the step.

Reading the Data from an Alternate Input File

The input for a linear constraint equation can be contained in a separate input file.

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

If the INPUT parameter is omitted, it is assumed that the data lines follow the keyword line.

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Equation**: click mouse button 3 while holding the cursor over the data table, and select **Read from File**

General Multi-Point Constraints

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Kinematic Constraints*
- **MPC*
- *Defining MPC constraints*
- *Connectors*

Overview

Multi-point constraints (MPCs) allow constraints to be imposed between different degrees of freedom of the model and can be quite general (nonlinear and nonhomogeneous).

The most commonly required constraints are available directly by choosing an MPC type and giving the associated data. The available MPC types are described below; MPCs that are available only in Abaqus/Standard are designated with an $^{(S)}$.

In Abaqus/Standard, the constraints can also be given by the user subroutine *MPC*.

Linear constraints can be given directly by defining a linear constraint equation (see *Linear Constraint Equations*).

In Abaqus/Explicit some multi-point constraints can be modeled more effectively using rigid bodies (see *Rigid Body Definition*).

Several MPC types are also available with connector elements (*Connector Elements*). Although the connector elements impose the same kinematic constraint, connectors do not eliminate degrees of freedom.

MPC constraint forces are not available as output quantities. Therefore, to output the forces required to enforce the constraint specified in an MPC, you should use an equivalent connector element. Connector element force, moment, and kinematic output is readily available and is defined in *Connector Element Library*.

Identifying the Nodes Involved in the MPC

For any MPC type, either node sets or individual nodes can be given as input. If the first entry is a node, subsequent entries must be nodes. If the first entry is a node set, subsequent entries can be either node sets or single nodes. The latter option is useful if a degree of freedom at each of a set of nodes depends on a degree of freedom of a single node, such as might occur in certain symmetry conditions or in the simulation of a rigid body.

If node sets are used, corresponding set entries will be constrained to each other. If sorted node sets are given as input, you must ensure that the nodes are numbered such that they will match up correctly when sorted. The nodes in an unsorted node set (see *Node Definition*) will be used in the order that they are given in defining the set.

In Abaqus/Standard multi-point constraints cannot be used to connect two rigid bodies at nodes other than the reference nodes, since multi-point constraints use degree-of-freedom elimination and the other nodes on a rigid body do not have independent degrees of freedom. In Abaqus/Explicit a rigid body reference node or any other node on a rigid body can be used in a multi-point constraint definition.

Abaqus/CAE uses connectors to define multi-point constraints between two points and constraints to define multi-point constraints between a point and secondary nodes in a region. Set-to-set multi-point constraints and unsorted node sets are not supported in Abaqus/CAE.

Input File Usage: **MPC*

Abaqus/CAE Usage: Use the following options to define a multi-point constraint between two points:

Interaction module:
Connector->Geometry->Create Wire Feature
Connector->Section->Create: **Connection Category**: **MPC**,
MPC type: select type
Connector->Assignment->Create: select wires: **Section**: select MPC connector section

Use the following options to define a multi-point constraint between a point and secondary nodes in a region:

Interaction module:
Constraint->Create: **MPC Constraint**: select control point and region; **MPC type**: select type

Use with Transformed Coordinate Systems

Local coordinate systems (see *Transformed Coordinate Systems*) can be defined for any nodes connected to MPCs. Some special considerations apply for user-defined MPCs, as described in *MPC*.

Defining Multiple Multi-Point Constraints at a Point

See *About Kinematic Constraints* for details on how multiple kinematic constraints at a point are treated in Abaqus/Standard and Abaqus/Explicit.

In Abaqus/Standard MPCs are usually imposed by eliminating the degree of freedom at the first node given (the dependent degree of freedom). MPC types BEAM, CYCLSYM, LINK, PIN, REVOLUTE, TIE, and UNIVERSAL are sorted internally by Abaqus/Standard so that the MPC in which a node is used as a dependent node is the last MPC that uses this node. Therefore, groups of these MPCs can be given in any order. However, even for these MPCs, a node can be used only once as a dependent node. In other cases dependent degrees of freedom should not be used subsequently to impose kinematic constraints; this generally precludes the use of the first node in an MPC definition as an independent node in any subsequent multi-point constraint, equation constraint, kinematic coupling constraint, or tie constraint definition.

Using MPCS in Implicit Dynamic Analysis

In implicit dynamic analysis Abaqus/Standard enforces MPCs rigorously for the displacements. The velocities and accelerations are derived from the displacements with the relations defined by the dynamic integration operator (see *Implicit dynamic analysis*). For linear MPCs (such as PIN, TIE, and mesh refinement MPCs) and geometrically linear analysis the velocities obtained in this way satisfy the constraint exactly. However, the accelerations satisfy the constraint only approximately. If nonlinear MPCs (such as BEAM, LINK, and SLIDER) are used in geometrically nonlinear analysis, both the velocities and accelerations satisfy the constraint only approximately. In most cases the approximation is quite accurate, but in some cases high frequency oscillations might occur in the accelerations of the nodes involved in the MPC.

Using Nonlinear MPCS in Geometrically Linear Abaqus/Standard Analysis

If a nonlinear MPC is used in a geometrically linear Abaqus/Standard analysis (see *General and Perturbation Procedures*), the MPC is linearized. For example, if MPC LINK is used in a geometrically nonlinear Abaqus/Standard analysis, the distance between the two nodes of the link remains constant. If it is used in a geometrically linear Abaqus/Standard analysis, the distance between the two nodes is held constant after projection onto the direction of the line between the original positions of the nodes. The difference should be noticeable only if the magnitudes of the rotations and displacements are not small.

Defining MPCS in a User Subroutine

In Abaqus/Standard you can define multi-point constraints in user subroutine *MPC*.

Constraints defined in user subroutine *MPC* can only use degrees of freedom that also exist on an element somewhere in the same model. For example, if a model contains no elements with rotational degrees of freedom, user subroutine *MPC* cannot use degrees of freedom 4, 5, or 6. This limitation can be overcome by adding a suitable element somewhere in the model to introduce the required degrees of freedom. This element can be added so that it does not affect the response of the model.

Constraints defined in the user subroutine are applied to the transformed degrees of freedom. A boundary nonlinearity occurs in Abaqus/Standard when MPCs are activated/deactivated in a user subroutine.

Input File Usage: **MPC*, USER

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **User-defined** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **User-defined** as the **MPC Type**

Specifying the Version of User Subroutine `MPC`

You must specify whether the user subroutine will be coded in degree of freedom mode or in nodal mode.

Input File Usage: Use one of the following options:

**MPC*, USER, MODE=DOF
**MPC*, USER, MODE=NODE

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **User-defined** as the **MPC Type**, choose **DOF-by-DOF** or **Node-by-Node**

Interaction module: **Create Constraint**: **MPC Constraint**: select **User-defined** as the **MPC Type**, choose **DOF-by-DOF** or **Node-by-Node**

Reading the Data from an Alternate Input File

The input for an MPC definition can be contained in a separate input file.

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

If the INPUT parameter is omitted, it is assumed that the data lines follow the keyword line.

Abaqus/CAE Usage: Reading data from an alternate input file is not supported in Abaqus/CAE.

MPCS for Mesh Refinement

LINEAR

This MPC is a standard method for mesh refinement of first-order elements. It applies to all active degrees of freedom at the involved nodes including temperature, pressure, and electrical potential.

In Abaqus/Explicit it might be preferable to use a surface-based tie constraint (see *Mesh Tie Constraints*) for mesh refinement, particularly when one or more of the meshes to be constrained involve shell elements with thickness.

QUADRATIC(S)

This MPC is a standard method for mesh refinement of second-order elements. It applies to all active degrees of freedom at the involved nodes except for temperature degrees of freedom in coupled temperature-displacement analysis and coupled thermal-electrical-structural analysis and to pressure degrees of freedom in coupled pore pressure analysis. For refinement using second-order pore pressure or coupled-temperature displacement elements, the P LINEAR or T LINEAR MPC must be used with this MPC.

BILINEAR(S)

This MPC is a standard method for mesh refinement of first-order solid elements in three dimensions. It applies to all active degrees of freedom at the involved nodes including temperature, pressure, and electrical potential.

C BIQUAD(S)

This MPC is a standard method for mesh refinement of second-order solid elements in three dimensions. It applies to all active degrees of freedom at the involved nodes except for temperature degrees of freedom in coupled temperature-displacement analysis and coupled thermal-electrical-structural analysis and to pressure degrees of freedom in coupled pore pressure analysis. For refinement using pore pressure or coupled-temperature displacement elements in three dimensions, the P BILINEAR or T BILINEAR MPC must be used with this MPC.

P LINEAR(S)

This MPC can be used with the QUADRATIC MPC for mesh refinement of second-order, fully coupled pore fluid flow-displacement elements. It applies to pressure degrees of freedom only. For acoustic analysis it applies the same constraint as the LINEAR MPC.

T LINEAR(S)

This MPC can be used with the QUADRATIC MPC for mesh refinement of second-order, fully coupled temperature-displacement and fully coupled thermal-electrical-structural elements. It applies to temperature degrees of freedom only. For heat transfer analysis it applies the same constraint as the LINEAR MPC.

P BILINEAR(S)

This MPC can be used in the C BIQUAD MPC for mesh refinement of pore fluid flow-displacement elements in three dimensions. It applies to pressure degrees of freedom only. For acoustic analysis it applies the same constraint as the BILINEAR MPC.

T BILINEAR(S)

This MPC can be used in the C BIQUAD MPC for mesh refinement of fully coupled temperature-displacement and fully coupled thermal-electrical-structural elements in three dimensions. It applies to temperature degrees of freedom only. For heat transfer analysis it applies the same constraint as the BILINEAR MPC.

Using Mesh Refinement MPCS with Shell or Beam Elements

The Abaqus/Standard shell elements S4R5, S8R5, S9R5, and STRI65 use a penalty method to enforce transverse shear constraints on the edges of the element. The use of mesh refinement MPCs LINEAR and QUADRATIC might, therefore, lead to overconstraining or “shear locking” of the bending behavior. Graded meshes, using the triangular elements as necessary to create a transition zone, are recommended for mesh refinement with these elements.

The shear flexible beam elements in Abaqus/Standard such as B31 or B32 will also “lock” if used as stiffeners along a mesh line where the mesh refinement MPCs are used.

For shell elements in Abaqus/Explicit the rotational degrees of freedom are not constrained by the LINEAR MPC; therefore, a hinge is formed along the line defined by the constrained nodes.

Using MPC Type LINEAR

MPC type LINEAR is a standard method for mesh refinement of first-order elements. However, in Abaqus/Explicit it might be preferable to use a surface-based tie constraint (see *Mesh Tie Constraints*) for mesh refinement, particularly when one or more of the meshes to be constrained involve shell elements with thickness.

This MPC constrains each degree of freedom at node p to be interpolated linearly from the corresponding degrees of freedom at nodes a and b (see *Figure 1*).

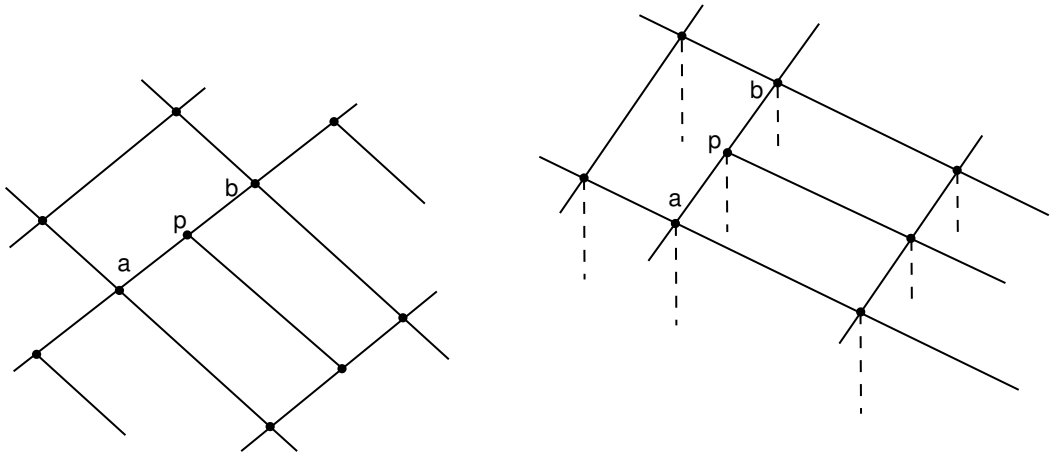


Figure 1: LINEAR type MPC.

Input Data

Give the nodes p, a, and b as shown in *Figure 1*.

Input File Usage: **MPC*
LINEAR, p, a, b

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type QUADRATIC

MPC type QUADRATIC is a standard method for mesh refinement of second-order elements. This MPC type is available only in Abaqus/Standard.

This MPC constrains each degree of freedom at node p (where p is either p_1 or p_2) to be interpolated quadratically from the corresponding degrees of freedom at nodes a, b, and c (*Figure 2*). For coupled temperature-displacement, coupled thermal-electrical-structural, or pore pressure elements, only the displacement degrees of freedom are constrained.

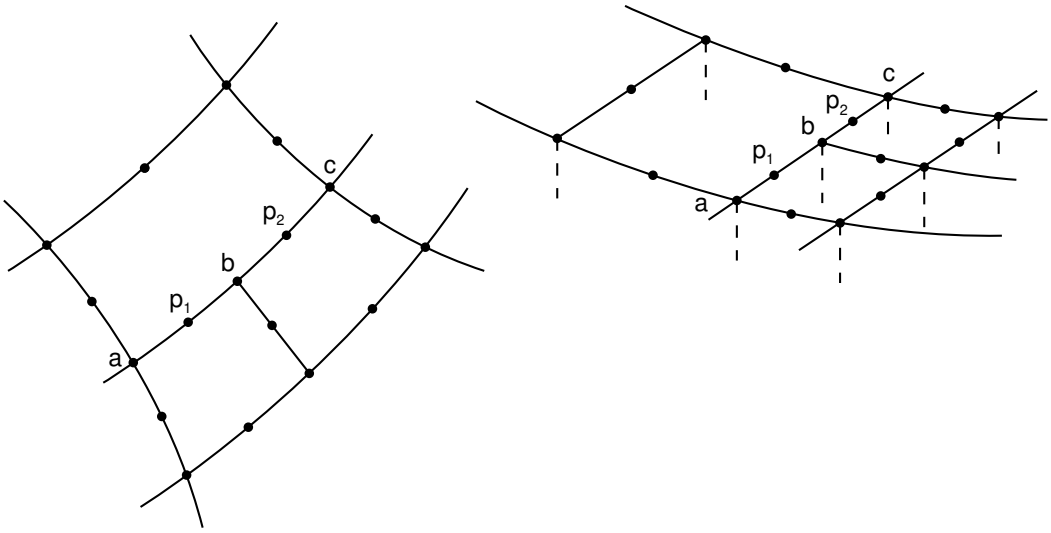


Figure 2: QUADRATIC type MPC.

Input Data

Give the nodes p, a, b, and c as shown in *Figure 2*, where p is either $\boldsymbol{p_1}$ or $\boldsymbol{p_2}$.

Input File Usage: **MPC*
QUADRATIC, p, a, b, c

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type BILINEAR

MPC type BILINEAR is a standard method for mesh refinement of first-order solid elements in three dimensions. This MPC type is available only in Abaqus/Standard.

This MPC constrains each degree of freedom at node p to be interpolated bilinearly from the corresponding degrees of freedom at nodes a, b, c, and d (*Figure 3*).

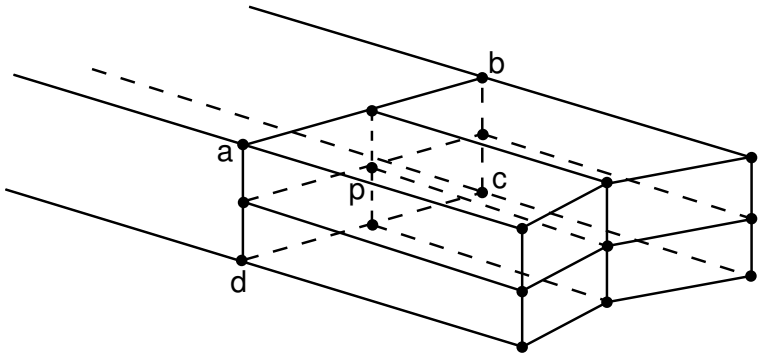


Figure 3: BILINEAR type MPC.

Input Data

Give the nodes p, a, b, c, and d as shown in *Figure 3*.

Input File Usage: **MPC*
BILINEAR, *p*, *a*, *b*, *c*, *d*

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type C BIQUAD

MPC type C BIQUAD is a standard method for mesh refinement of second-order solid elements in three dimensions. This MPC type is available only in Abaqus/Standard.

This MPC constrains each degree of freedom at node *p* to be interpolated by a constrained biquadratic from the corresponding degrees of freedom at the eight nodes *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h* (*Figure 4*). For coupled temperature-displacement, coupled thermal-electrical-structural, or pore pressure elements, only the displacement degrees of freedom are constrained.

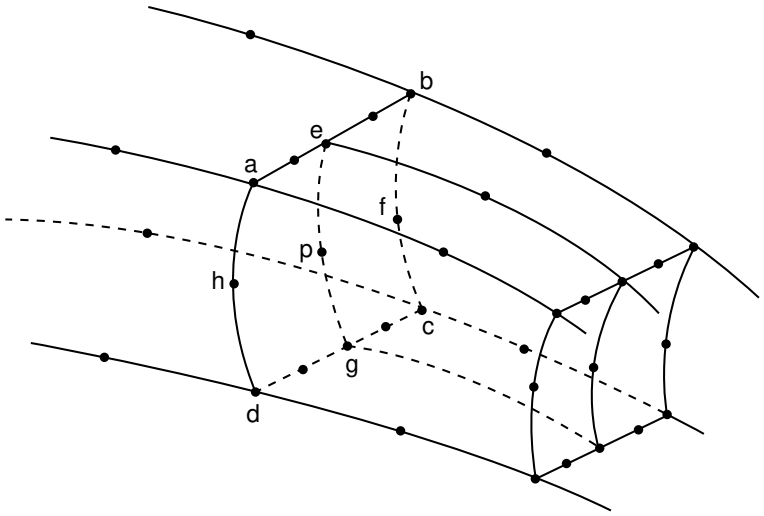


Figure 4: C BIQUAD type MPC.

Input Data

Give the nodes *p*, *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h* as shown in *Figure 4*.

Input File Usage: **MPC*
C BIQUAD, *p*, *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

Using MPC Types P LINEAR and T LINEAR

The P LINEAR MPC can be used in the QUADRATIC MPC for mesh refinement of second-order, fully coupled pore fluid flow-displacement elements.

The T LINEAR MPC can be used in the QUADRATIC MPC for mesh refinement of second-order, fully coupled temperature-displacement and fully coupled thermal-electrical-structural elements.

These MPC types are available only in Abaqus/Standard.

These MPCs constrain the pore pressure (P LINEAR) or temperature (T LINEAR) degree of freedom at node *p* to be interpolated linearly from the degrees of freedom at nodes *a* and *b* (*Figure 5*).

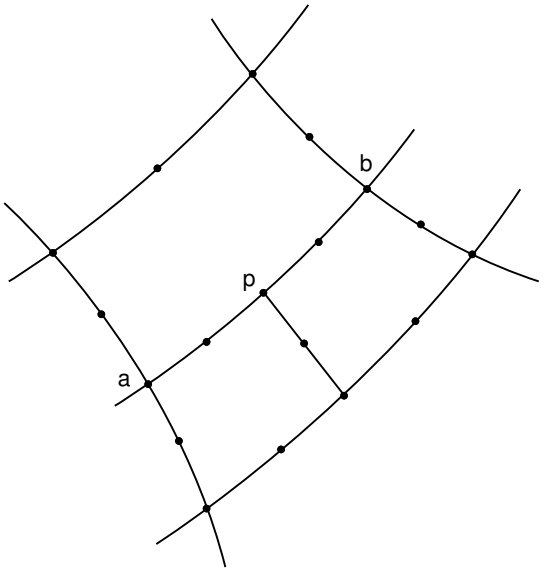


Figure 5: P LINEAR and T LINEAR MPCs.

Input Data

Give the nodes *p*, *a*, and *b* as shown in *Figure 5*.

Input File Usage: Use the following option to define a P LINEAR MPC:

**MPC*
P LINEAR, *p*, *a*, *b*

Use the following option to define a T LINEAR MPC:

**MPC*
T LINEAR, *p*, *a*, *b*

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

Using MPC Types P BILINEAR and T BILINEAR

The P BILINEAR MPC can be used in the C BIQUAD MPC for mesh refinement of pore fluid flow-displacement elements in three dimensions.

The T BILINEAR MPC can be used in the C BIQUAD MPC for mesh refinement of fully coupled temperature-displacement and fully coupled thermal-electrical-structural elements in three dimensions.

These MPC types are available only in Abaqus/Standard.

These MPCs constrain the pore pressure (P LINEAR) or temperature (T LINEAR) at node *p* to be interpolated bilinearly from the pore pressure or temperature at nodes *a*, *b*, *c*, and *d* (*Figure 6*).

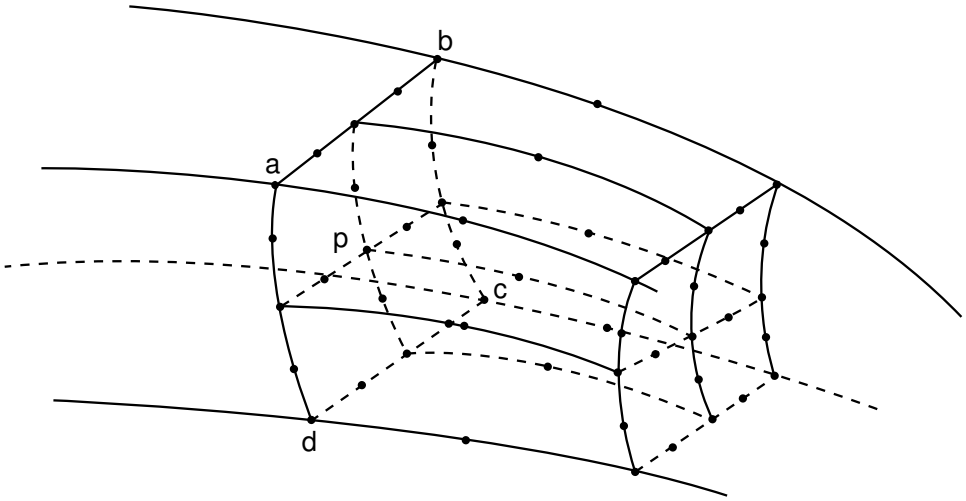


Figure 6: P BILINEAR and T BILINEAR MPCs.

Input Data

Give the nodes *p*, *a*, *b*, *c*, and *d* as shown in *Figure 6*.

Input File Usage: Use the following option to define a P BILINEAR MPC:

**MPC*
P BILINEAR, *p*, *a*, *b*, *c*, *d*

Use the following option to define a T BILINEAR MPC:

**MPC*
T BILINEAR, *p*, *a*, *b*, *c*, *d*

Abaqus/CAE Usage: Mesh refinement multi-point constraints are not supported in Abaqus/CAE.

MPCS for Connections and Joints

BEAM

Provide a rigid beam between two nodes to constrain the displacement and rotation at the first node to the displacement and rotation at the second node, corresponding to the presence of a rigid beam between the two nodes.

CYCLSYM(S)

Constrain nodes to impose cyclic symmetry in a model.

ELBOW(S)

Constrain two nodes of ELBOW31 or ELBOW32 elements together, where the cross-sectional direction, $\mathbf{a_2}$, changes (see *Pipes and Pipebends with Deforming Cross-Sections: Elbow Elements*).

LINK

Provide a pinned rigid link between two nodes to keep the distance between the two nodes constant. The displacements of the first node are modified to enforce this constraint. The rotations at the nodes, if they exist, are not involved in this constraint.

PIN

Provide a pinned joint between two nodes. This MPC makes the displacements equal but leaves the rotations, if they exist, independent of each other.

REVOLUTE(S)

Provide a revolute joint.

SLIDER

Keep a node on a straight line defined by two other nodes, but allow the possibility of moving along the line and allow the line to change length.

TIE

Make all active degrees of freedom equal at two nodes.

UNIVERSAL(S)

Provide a universal joint.

V LOCAL(S)

Allow the velocity at the constrained node to be expressed in terms of velocity components at the third node defined in a local, body axis system. These local velocity components can be constrained, thus providing prescribed velocity boundary conditions in a rotating, body axis system.

See *About Connectors* for element-based versions of several of these MPCs for connections and joints.

Using MPC Type BEAM

MPC type BEAM provides a rigid beam between two nodes to constrain the displacement and rotation at the first node to the displacement and rotation at the second node, corresponding to the presence of a rigid beam between the two nodes.

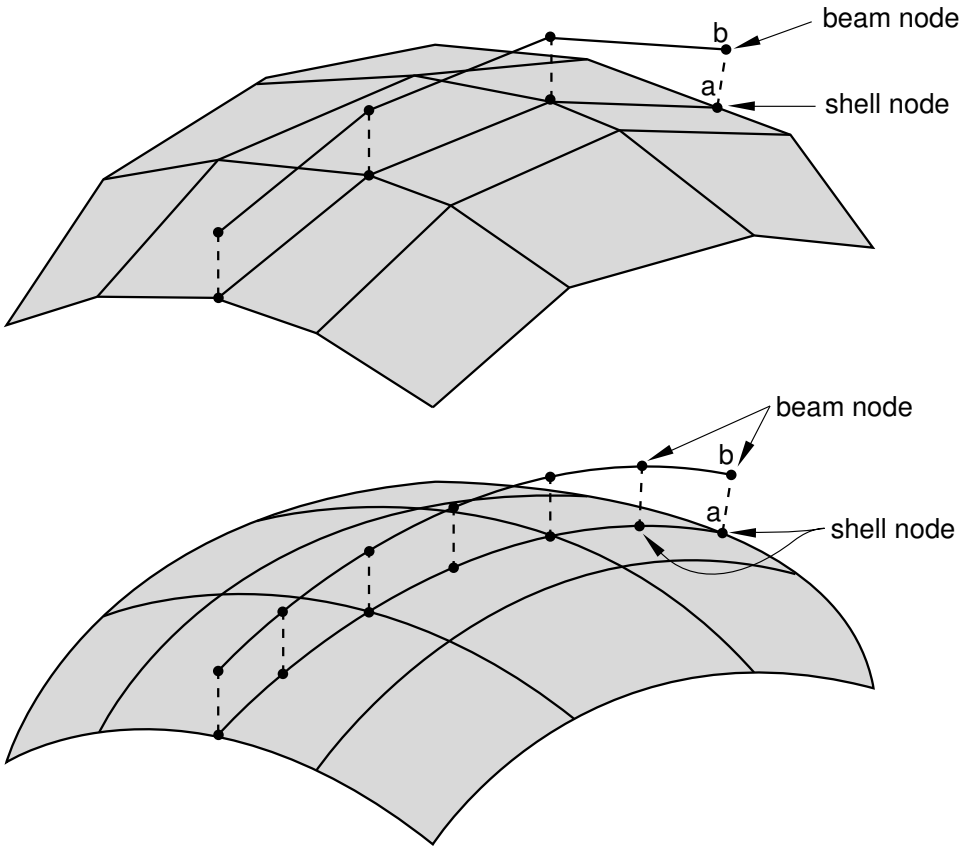


Figure 7: BEAM type MPC.

Input Data

Give the nodes *a* and *b* as shown in *Figure 7*.

Input File Usage: **MPC*
BEAM, *a*, *b*

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **Beam** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **Beam** as the **MPC Type**

Constraining a Beam Stiffener to a Shell

The general method of using a beam as a stiffener on a shell is to define the beam and shell elements with separate nodes. These nodes can then be constrained to each other using BEAM type MPCs.

A more economical way, when applicable, is to use the same node for the beam node and the shell node and then define the offset of the center of the cross-section of the beam in the beam section data. *Figure 8* shows a T-shaped stiffener attached to a shell, using the I-beam cross-section. You can define this configuration by setting *l* (see *Beam Cross-Section Library*) equal to the distance between the node and the underside of the lower flange

and setting the thickness of the top flange to zero. You can use this approach with all beam elements that use TRAPEZOID, I, CHANNEL, or ARBITRARY beam sections. Alternatively, you can define the offset directly as described in *Using a Beam Section Integrated during the Analysis to Define the Section Behavior* and *Using a General Beam Section to Define the Section Behavior*. You can use this approach for all beam cross-sections available in the beam cross section library (see *Beam Cross-Section Library*).

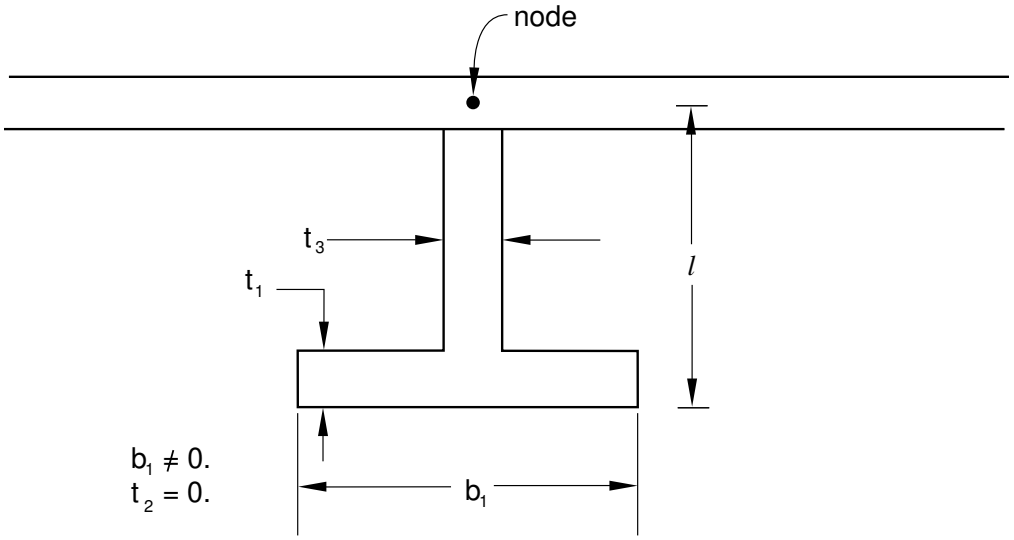


Figure 8: Stiffened shell.

Thermal Expansion with BEAM MPC

In Abaqus/Standard a BEAM MPC can experience expansion due to a temperature increase. The magnitude of the expansion depends on the distance between the nodes of the MPC. The temperature change for computing the expansion is the average of the temperature change at both the nodes of the MPC. 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. Thermal expansion can be used only when temperature is a field variable.

Input File Usage: **MPC*, ALPHA=*thermal_expansion_coeff*
BEAM, *a*, *b*

Abaqus/CAE Usage: Thermal expansion with BEAM type MPCs is not supported in Abaqus/CAE.

Using MPC Type CYCLSYM

MPC type CYCLSYM is used to enforce proper constraints on the radial faces bounding a segment of a cyclic symmetric structure (see *Figure 9*). This MPC type is available only in Abaqus/Standard.

MPC type CYCLSYM imposes the cyclic symmetry by equating radial, circumferential, and axial displacement components (and rotations, if active) at the two nodes (*a* and *b*). The symmetry axis can be defined by the original coordinates of two additional nodes (*c* and *d*) that do not need to be connected to any element in the structure. Scalar degrees of freedom (such as temperature) are made equal.

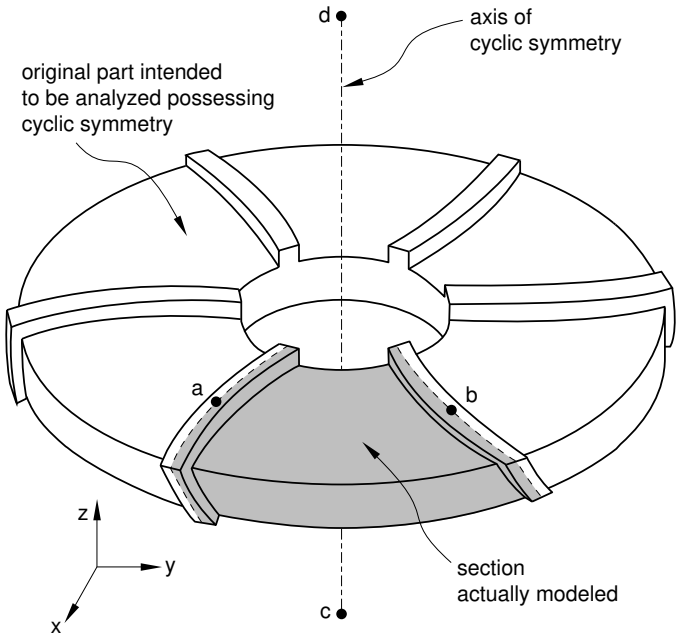


Figure 9: MPC type CYCLSYM.

Input Data

Give the nodes *a*, *b*, and (optionally) node *c* and/or *d* that define the axis of symmetry as shown in *Figure 9*. Node set names can be used instead of the nodes *a* and *b*. If neither *c* nor *d* is given, the global *z*-axis is taken to be the axis of cyclic symmetry. If only node *c* is given, the symmetry axis passes through *c* and is parallel to the global *z*-axis. Thus, node *d* is not needed in two-dimensional cases.

Input File Usage: **MPC*
CYCLSYM, *a*, *b*, *c*, *d*

Abaqus/CAE Usage: Cyclic symmetry multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type ELBOW

MPC type ELBOW constrains two nodes of ELBOW31 or ELBOW32 elements together, where the cross-sectional direction, $\mathbf{a}_2$, changes (see *Pipes and Pipebends with Deforming Cross-Sections: Elbow Elements*). This MPC type is available only in Abaqus/Standard.

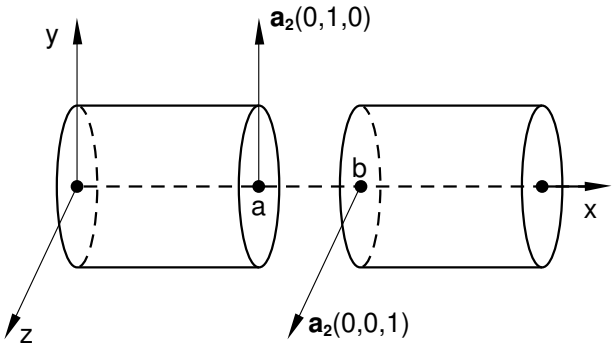


Figure 10: ELBOW type MPC.

Input Data

Give the nodes *a* and *b* as shown in *Figure 10*.

Input File Usage: **MPC*
ELBOW, *a*, *b*

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **Elbow** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **Elbow** as the **MPC Type**

Using MPC Type LINK

MPC type LINK provides a pinned rigid link between two nodes to keep the distance between the nodes constant, as shown in *Figure 11*. The displacements of the first node are modified to enforce this constraint. The rotations at the nodes, if they exist, are not involved in this constraint.

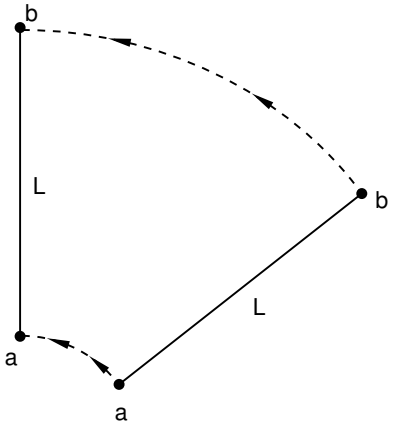


Figure 11: MPC type LINK.

Input Data

Give the nodes *a* and *b* as shown in *Figure 11*.

Input File Usage: **MPC*
LINK, *a*, *b*

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **Link** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **Link** as the **MPC Type**

Thermal Expansion with LINK MPC

In Abaqus/Standard a LINK MPC can experience expansion due to a temperature increase. The magnitude of the expansion depends on the distance between the nodes of the MPC. The temperature change for computing the expansion is the average of the temperature change at both the nodes of the MPC. 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. Thermal expansion can be used only when temperature is a field variable.

Input File Usage: **MPC*, ALPHA=*thermal_expansion_coeff*
LINK, *a*, *b*

Abaqus/CAE Usage: Thermal expansion with LINK type MPCs is not supported in Abaqus/CAE.

Using MPC Type PIN

MPC type PIN provides a pinned joint between two nodes. This MPC makes the global displacements equal but leaves the rotations, if they exist, independent of each other, as shown in *Figure 12*.

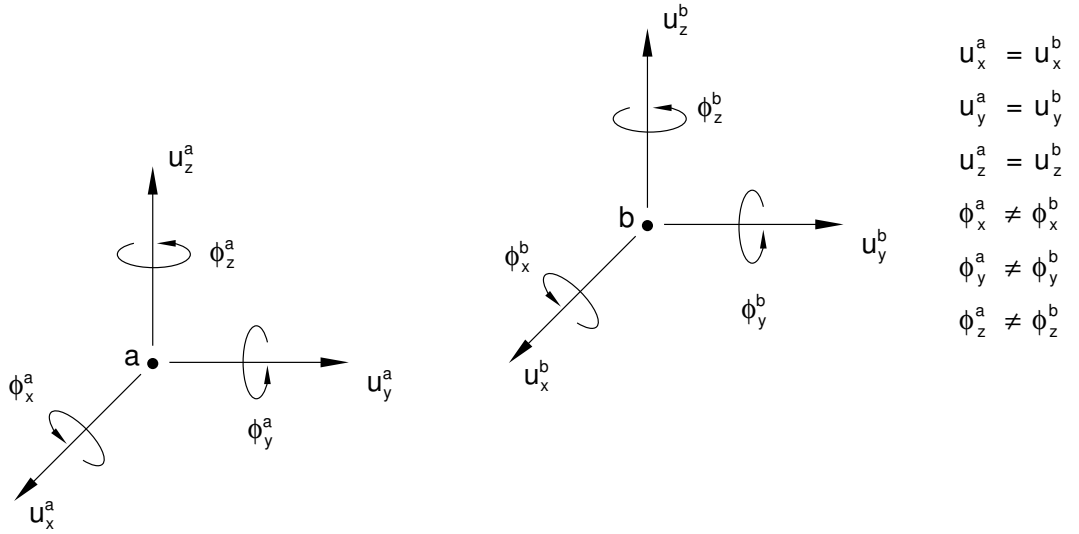


Figure 12: MPC type PIN.

Input Data

Give the nodes *a* and *b* as shown in *Figure 12*.

Input File Usage: **MPC*
PIN, *a*, *b*

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **Pin** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **Pin** as the **MPC Type**

Using MPC Type REVOLUTE

This MPC type is available only in Abaqus/Standard.

A revolute joint is a joint in which relative rotation is allowed between two nodes about an axis that rotates during the motion (see *Figure 13*). The axis of the joint is defined in the initial configuration as the line from node *b* to node *c*. If these nodes are coincident, the axis is assumed to be the global *z*-axis. The rotation of the joint axis is that of node *b*.

The relative rotation in the joint is a single variable and is stored as degree of freedom 6 at node *c*. This degree of freedom can be used with other members in the model, but caution should be used because of the nonstandard use of degree of freedom 6. For example, a SPRING1 element (a spring to ground) might be attached to this degree of freedom. Since the degree of freedom measures a *relative* rotation, this spring would then be a torsional spring between nodes *a* and *b*.

The displacements at node *a* are not constrained by the REVOLUTE MPC to be the same as the displacements at node *b*. Thus, the joint definition must usually be completed either by using a PIN type MPC between nodes *a* and *b* or by using suitable stiffness members between these two nodes.

An example of a revolute joint and application of the REVOLUTE MPC is provided in *Revolute MPC verification: rotation of a crank*. See *Revolute joint* for more details on revolute joints.

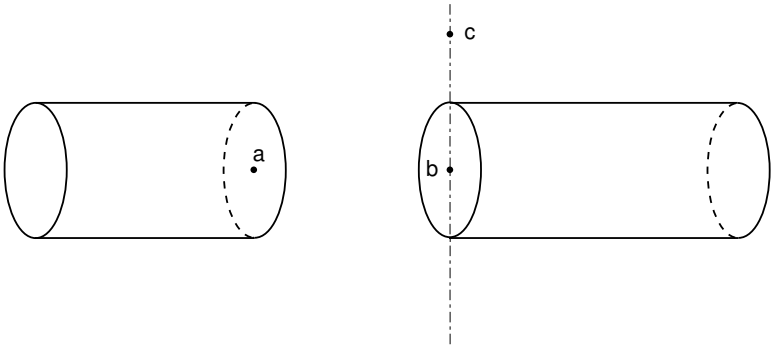


Figure 13: Revolute joint.

Input Data

Give the nodes *a*, *b*, and *c* as shown in *Figure 13*. Degree of freedom 6 at node *c* defines the *relative* rotation between nodes *a* and *b*; therefore, this degree of freedom does not obey the standard convention for degrees of freedom in Abaqus.

Input File Usage: **MPC*
REVOLUTE, *a*, *b*, *c*

Abaqus/CAE Usage: Revolute joint multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type SLIDER

MPC type SLIDER keeps a node on a straight line defined by two other nodes but allows the possibility of moving along the line and allows the line to change length.

When transitioning from multiple layers of solid elements to shells, it is often desirable to constrain the nodes on the free edge of the solid elements to remain in a straight line. (This constraint is consistent with shell theory.) The SLIDER MPC can perform this function without restraining the “thinning” behavior of the solid layers. The SS LINEAR MPC is then used to attach the shell element to this edge.

In Abaqus/Standard when a SLIDER MPC is used with one of the shell-solid MPCs—SS LINEAR, SS BILINEAR, or SSF BILINEAR—it must be given following the shell-solid MPCs.

Input Data

For each node *p* shown in *Figure 14* and *Figure 15*, give the nodes *p*, *a*, and *b* for each line of nodes that should remain straight. For each node *q* shown in *Figure 14*, give the nodes *q*, *c*, and *d*, and so on for each line of nodes that should remain straight.

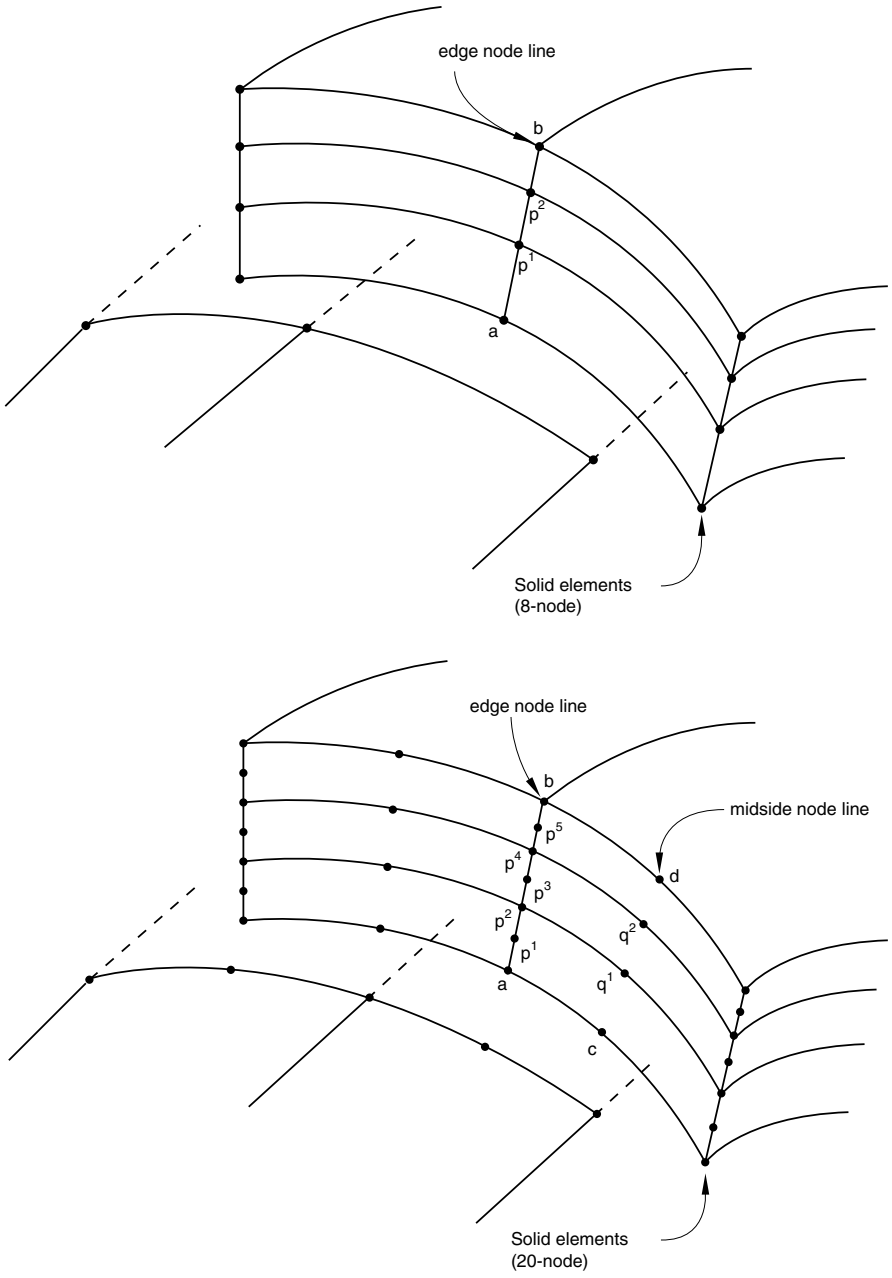


Figure 14: SLIDER type MPC used at a shell-solid intersection.

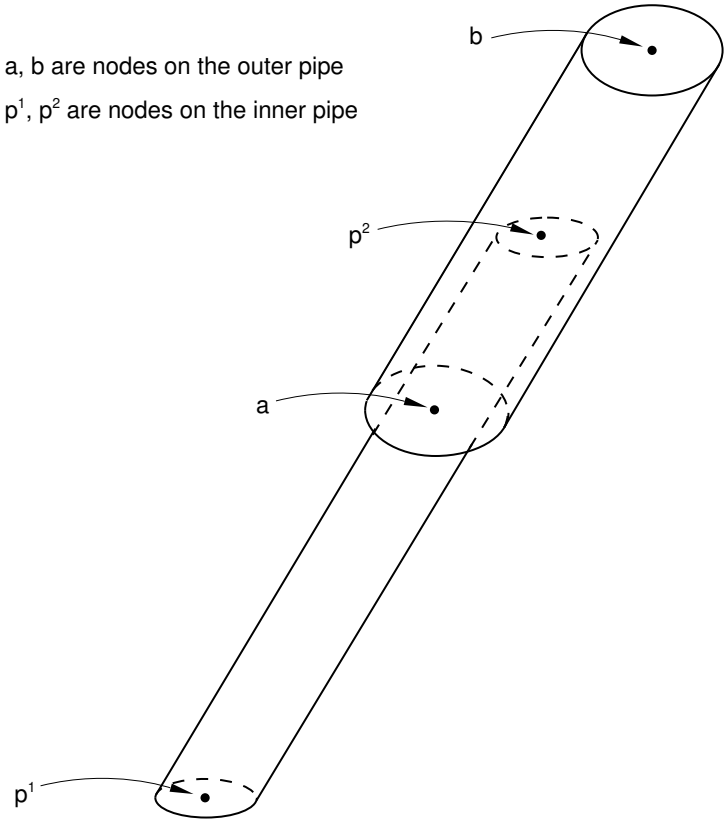


Figure 15: SLIDER type MPC used to model a telescoping beam.

Input File Usage: **MPC*
SLIDER, *p*, *a*, *b*
SLIDER, *q*, *c*, *d*

Abaqus/CAE Usage: Slider multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type TIE

MPC type TIE makes the global displacements and rotations as well as all other active degrees of freedom equal at two nodes. If there are different degrees of freedom active at the two nodes, only those in common will be constrained.

MPC type TIE is usually used to join two parts of a mesh when corresponding nodes on the two parts are to be fully connected ("zipping up" a mesh). For example, when a mesh is generated on a cylindrical body, the solution at the nodes at 0° and those at 360° must be the same. This can be done either by renumbering the nodes on one of the mesh extremes or by using this MPC for each pair of corresponding nodes, as shown in *Figure 16*.

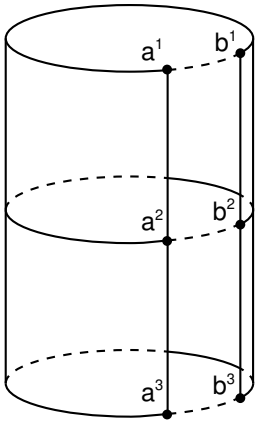


Figure 16: Example of use of TIEMPC.

Input Data

Give the nodes *a* and *b* as shown in *Figure 16*.

Input File Usage: **MPC*
TIE, *a*, *b*

Abaqus/CAE Usage: Use one of the following options:

Interaction module: **Create Connector Section**: select **MPC** as the **Connection Category** and **Tie** as the **MPC Type**

Interaction module: **Create Constraint**: **MPC Constraint**; select **Tie** as the **MPC Type**

Using MPC Type UNIVERSAL

This MPC type is available only in Abaqus/Standard.

A universal joint is a joint in which relative rotation is allowed between two nodes, about two axes that are connected rigidly, and each of which rotates with the rotation of one end of the joint (see *Figure 17*). Such a joint might be used to couple two shafts that have an angular misalignment. The first axis of the joint, which is attached to node *b*, is defined in the initial configuration as the line from node *b* to node *c*. If these nodes are coincident, the axis is assumed to be the global *z*-axis. The second axis of the joint is at right angles to the first axis and is in the plane defined by the first axis and node *d*.

The relative rotations in the joint are stored as degree of freedom 6 at the nodes *c* and *d*. These degrees of freedom can be used with other members in the model, but caution should be used because of the nonstandard use of degree of freedom 6. For example, a SPRING1 element (a spring to ground) might be attached to one of these degrees of freedom. Since the degree of freedom measures a *relative* rotation, this spring would then be a torsional spring, restraining that component of relative rotation.

The displacements at node *a* are not constrained by the UNIVERSAL MPC to be the same as the displacements at node *b*. Thus, the joint definition must usually be completed either by using a PIN type MPC between nodes *a* and *b* or by using suitable stiffness members between these two nodes.

See *Universal joint* for more details on universal joints.

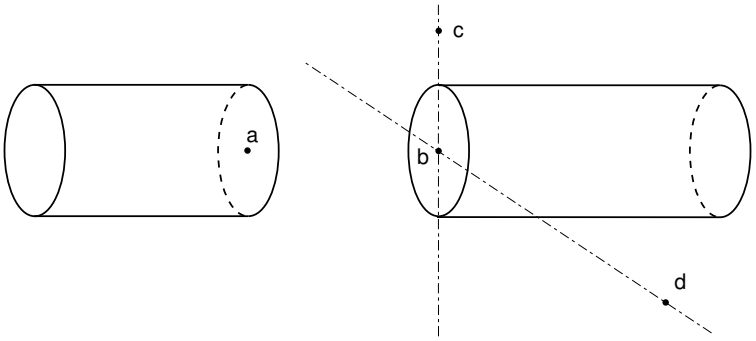


Figure 17: Universal joint.

Input Data

Give the nodes *a*, *b*, *c*, and *d* as shown in *Figure 17*. Degrees of freedom 6 at nodes *c* and *d* define the *relative* rotation in the joint; therefore, these degrees of freedom do not obey the standard convention for degrees of freedom in Abaqus.

Input File Usage: **MPC*
UNIVERSAL, *a*, *b*, *c*, *d*

Abaqus/CAE Usage: Universal joint multi-point constraints are not supported in Abaqus/CAE.

Using MPC Type V LOCAL

This MPC type is available only in Abaqus/Standard.

As shown in *Figure 18*, MPC type V LOCAL constrains the velocity components associated with degrees of freedom 1, 2, and 3 at a first node (*a*) to be equal to the velocity components at a third node (*c*) along local, rotating directions. These local directions rotate according to the rotation at a second node (*b*). In the initial configuration, the first local direction is from the second to the third node of the MPC (from *b* to *c*, as indicated by the arrows in *Figure 18*), or it is the global *z*-axis if these nodes coincide. The other local directions are then defined by the standard Abaqus convention for such directions (see *Conventions*). In *Figure 18* this MPC is applied to nodes *d*, *e*, and *f* in the same manner.

MPC type V LOCAL can be useful for defining a complex motion within a model. For example, the MPC can be used to model the steering of an automobile in a dynamic analysis for which the resulting inertial effects are of interest. See *Local velocity constraint* for more details on the local velocity constraint.

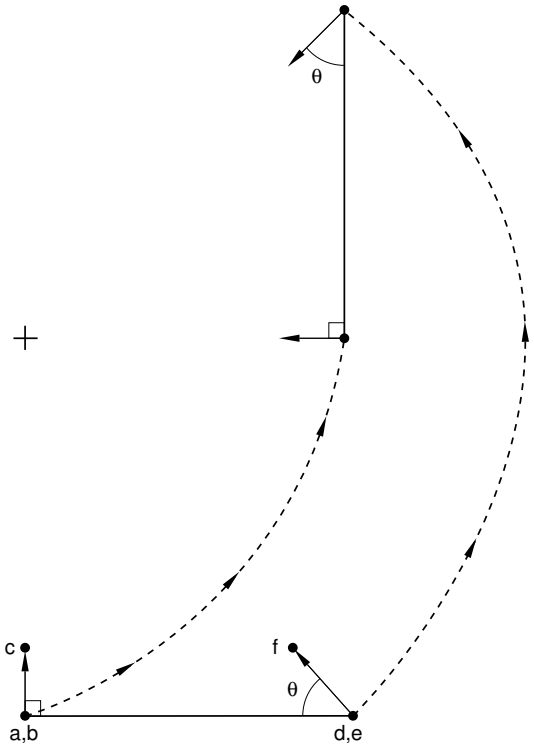


Figure 18: Local velocity constraint.

Input Data

Give the node whose velocity components are constrained (node *a* or *d* in *Figure 18*), the node whose rotation defines the rotation of the local directions (node *b* or *e* in *Figure 18*), and the node whose velocity components are in these local directions (node *c* or *f* in *Figure 18*). Nodes *a* and *b* (or *d* and *e*) can be the same.

Input File Usage: **MPC*
V LOCAL, *a*, *b*, *c*
V LOCAL, *d*, *e*, *f*

Abaqus/CAE Usage: Local velocity component multi-point constraints are not supported in Abaqus/CAE.

MPCS for Transitions

SS LINEAR

Constrain a shell node to a solid node line for linear elements (such as S4, S4R, S4R5, C3D8, C3D8R, SAX1, and CAX4).

SS BILINEAR(S)

Constrain a shell node to a solid node line for edge lines on quadratic elements (such as S8R, S8R5, C3D20, C3D20R, SAX2, and CAX8).

SSF BILINEAR(S)

Constrain a midside node of a quadratic shell element (such as S8R and S8R5) to midface lines on 20-node bricks (such as C3D20 and C3D20R).

Modeling a Shell-to-Solid Element Transition

The SLIDER, SS LINEAR, SS BILINEAR, and SSF BILINEAR MPCs allow for a transition from shell element modeling to solid element modeling on a shell surface. This modeling technique can be used to obtain solutions at shell-solid intersections or other discontinuities, where the local modeling should use full three-dimensional theory but the other parts of the structure can be modeled as shells. The shell-to-solid submodeling capability (*About Submodeling*) and the surface-based shell-to-solid coupling constraint (*Shell-to-Solid Coupling*) can also be used to obtain more accurate solutions in such cases, with considerably less modeling effort.

In Abaqus/Standard the MPC usage assumes that the interface between the shell and solid elements is a surface containing the normals to the shell along the line of intersection of the meshes, so that the lines of nodes on the solid mesh side of the interface in the normal direction to the surface are straight lines. (Line a, $\boldsymbol{p}^1$, $\boldsymbol{p}^2$, …, b in *Figure 14* and lines $\boldsymbol{p}^1$, $\boldsymbol{p}^2$, …, $\boldsymbol{p}^n$ in *Figure 19* to *Figure 20* should be straight lines.) It also assumes that the nodes of the solid elements are spaced uniformly on the interface surface as indicated in *Figure 14* and *Figure 19* to *Figure 20*. For each shell node on the edge use MPC type SS LINEAR, SS BILINEAR, or SSF BILINEAR, as appropriate, to constrain the shell node to the corresponding line or face of solid element nodes through the thickness. Then, use a SLIDER MPC to constrain each interior node on the line through the thickness to remain on the straight line defined by the bottom and top nodes of that line. For an example, see *Multi-point constraints*.

The SS BILINEAR and SSF BILINEAR MPCs are not intended for use with the variable node solid elements (C3D27, C3D27H, C3D27R, and C3D27RH).

In Abaqus/Standard MPCs SS LINEAR, SS BILINEAR, and SSF BILINEAR eliminate all displacement components and two of the rotation components at the shell node, and the SLIDER MPC eliminates two displacement components at each interior solid element node in the interface. Therefore, any boundary conditions needed at the interface (such as those required when the shell/solid interface intersects a symmetry plane) should be applied only to the top and bottom nodes on the solid element side of the interface.

Using MPC Type SS LINEAR

MPC type SS LINEAR constrains a shell corner node to a line of edge nodes on solid elements for linear elements (S4, S4R, or S4R5; C3D8, C3D8R; SAX1; CAX4; etc.).

The constrained nodes need not lie exactly on these lines, but it is suggested that they be in close proximity to the lines for meaningful results.

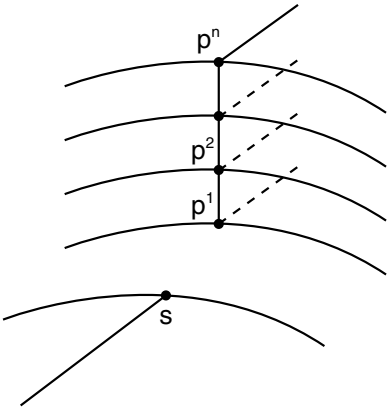


Figure 19: SS LINEAR type MPC. 4-node shells to 8-node bricks.

Input Data

Give the shell node, *S*, then the list of nodes along the corresponding line through the thickness in the solid element mesh. In Abaqus/Explicit only two solid nodes can be given. Referring to *Figure 19*, in Abaqus/Standard give *S*, $\boldsymbol{p}^1$, $\boldsymbol{p}^2$, …, $\boldsymbol{p}^n$, and in Abaqus/Explicit give *S*, $\boldsymbol{p}^1$, $\boldsymbol{p}^n$, where $\boldsymbol{n} \geq \mathbf{2}$. The shell node number must be different from the solid mesh node numbers.

Input File Usage: In Abaqus/Standard use the following option:

**MPC*

SS LINEAR, *S*, $\boldsymbol{p}^1$, $\boldsymbol{p}^2$, …, $\boldsymbol{p}^n$

In Abaqus/Explicit use the following option:

**MPC*

SS LINEAR, *S*, $\boldsymbol{p}^1$, $\boldsymbol{p}^n$

Abaqus/CAE Usage: Multi-point constraints for transitions are not supported in Abaqus/CAE.

Using MPC Type SS BILINEAR

MPC type SS BILINEAR constrains a corner node of a quadratic shell element (S8R, S8R5) to a line of edge nodes on 20-node bricks. This MPC type is available only in Abaqus/Standard.

The constrained node need not lie exactly on the line, but it is suggested that it be in close proximity to the line for meaningful results.

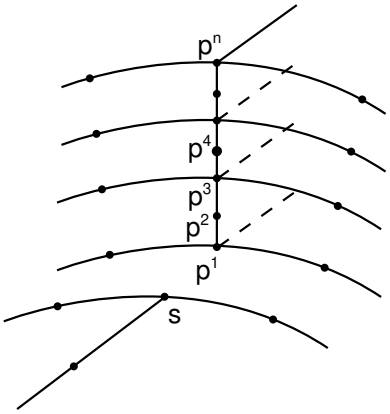


Figure 20: SS BILINEAR type MPC. Corner of 8-node shell to edge of 20-node bricks.

Input Data

Give the shell node, *S*, then the list of nodes along the corresponding line through the thickness in the solid element mesh. Referring to *Figure 20*, give *S*, p^1, p^2,…, p^n. The shell node number must be different from the solid mesh node numbers.

Input File Usage: **MPC*
SS BILINEAR, *S*, p^1, p^2, …, p^n

Abaqus/CAE Usage: Multi-point constraints for transitions are not supported in Abaqus/CAE.

Using MPC Type SSF BILINEAR

MPC type SSF BILINEAR constrains a midside node on a quadratic shell element (S8R, S8R5) to a line of midface nodes on solid 20-node bricks. This MPC type is available only in Abaqus/Standard.

The constrained node need not lie exactly on the line, but it is suggested that it be in close proximity to the line for meaningful results.

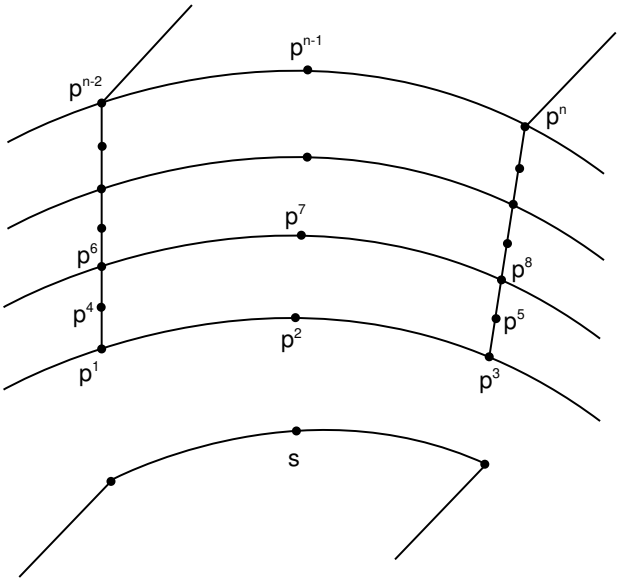


Figure 21: SSF BILINEAR type MPC. Midside of 8-node shell to surface of 20-node bricks.

Input Data

Give the shell node, *S*, then the list of nodes on the solid face, in the order $\boldsymbol{p^1}$, $\boldsymbol{p^2}$,…, $\boldsymbol{p^n}$ as shown in *Figure 21*.

Input File Usage: **MPC*

SSF BILINEAR, *S*, $\boldsymbol{p^1}$, $\boldsymbol{p^2}$, …, $\boldsymbol{p^n}$

Abaqus/CAE Usage: Multi-point constraints for transitions are not supported in Abaqus/CAE.

Kinematic Coupling Constraints

Products: Abaqus/Standard

References:

- *About Kinematic Constraints*
- **KINEMATIC COUPLING*

Overview

Kinematic coupling constraints limit the motion of a group of nodes to rigid body motion defined by a reference node.

In addition, kinematic coupling constraints:

- can be applied only to specific user-specified degrees of freedom at the constrained nodes;
- can be specified with respect to local coordinate systems at the constrained nodes; and
- can be used in geometrically linear or nonlinear analysis.

The preferred method of providing a kinematic constraint of this type is described in *Coupling Constraints*.

Typical Applications

The kinematic coupling constraints are useful in cases where a large number of nodes (the “coupling” nodes) are constrained to the rigid body motion of a single node and the degrees of freedom that participate in the constraint are selected individually in a local coordinate system. In many such cases MPCs either are not available or must be prescribed individually for each constrained node. A typical example is shown in *Figure 1*, where a kinematic coupling constraint is used to prescribe a twisting motion to a model without constraining radial motions. In other applications the kinematic coupling constraint can be used to provide coupling between continuum and structural elements.

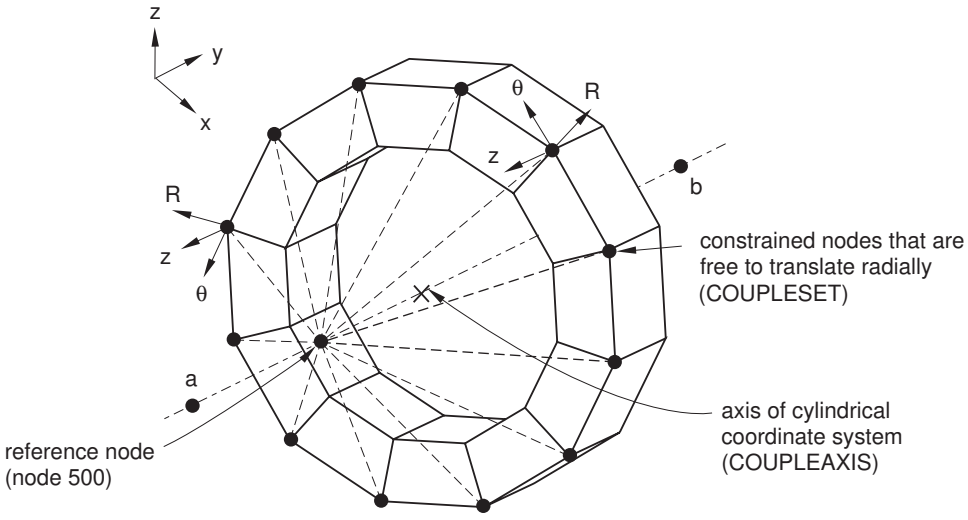


Figure 1: A kinematic coupling constraint used to transmit rotation to a structure while permitting radial motion.

Defining the Constraint

A kinematic coupling constraint requires the specification of a reference node, coupling nodes, and the constrained degrees of freedom at these nodes. The reference node has both translational and rotational degrees of freedom.

Kinematic constraints are imposed by eliminating degrees of freedom at the coupling nodes. Once any combination of displacement degrees of freedom at a coupling node is constrained, additional displacement constraints—such as MPCs, boundary conditions, or other kinematic coupling definitions—cannot be applied to any coupling node involved in a kinematic coupling constraint. The same limitation applies for rotational degrees of freedom.

Input File Usage: To constrain all available degrees of freedom:

**KINEMATIC COUPLING*, REF NODE=*node*
coupling node number or node set

To constrain a single degree of freedom:

**KINEMATIC COUPLING*, REF NODE=*node*
coupling node number or node set, dof

To constrain a range of degrees of freedom:

**KINEMATIC COUPLING*, REF NODE=*node*
coupling node number or node set, first dof, last dof

To specify noncontiguous lists of constrained degrees of freedom, repeat the node numbers or node sets on subsequent data lines. For example, the following input is used to constrain degrees of freedom 1, 2, 3, and 6 at node 10 to the motion of reference node 5:

**KINEMATIC COUPLING*, REF NODE=5
10, 1, 3
10, 6

Translational Degrees of Freedom

Translational degrees of freedom are constrained by eliminating the specified degrees of freedom at the coupling nodes. When all translational degrees of freedom are specified, the coupling nodes follow the rigid body motion of the reference node.

Rotational Degrees of Freedom

All combinations of selected rotational degrees of freedom result in rotational behavior that is identical to existing MPC types. Specifically:

- Selection of three rotational degrees of freedom along with three displacement degrees of freedom is equivalent to MPC type BEAM.
- Selection of two rotational degrees of freedom is equivalent to MPC type REVOLUTE.
- Selection of one rotational degree of freedom is equivalent to MPC type UNIVERSAL.

Internal nodes are created by the kinematic coupling to enforce the constraints that are equivalent to MPC types REVOLUTE and UNIVERSAL. These nodes have the same degrees of freedom as the additional nodes used in these MPC types and are included in the residual check for nonlinear analysis.

Specifying a Local Coordinate System

The constrained degrees of freedom at the coupling nodes can be specified in a local coordinate system instead of the (default) global coordinate system (see *Orientations*). *Figure 1* illustrates the use of a local coordinate system definition with a kinematic coupling constraint to constrain all but the radial translation of a group of nodes to a reference node. In this example a local cylindrical coordinate system is defined that has its axis coincident with the structure's axis. The coupling node constraints are then specified in this local coordinate system. In this example the constrained nodes are attached to continuum elements; thus, only translational degrees of freedom need to be specified.

Input File Usage: **KINEMATIC COUPLING*, REF NODE=*node*, ORIENTATION=*name*

For example, the following input is used to specify the kinematic coupling constraint shown in *Figure 1*:

```
*ORIENTATION, SYSTEM=CYLINDRICAL, NAME=COUPLEAXIS
0.0, -1.0, 0.0, 0.0, 1.0, 0.0
*KINEMATIC COUPLING, REF NODE=500,
ORIENTATION=COUPLEAXIS
COUPLESET, 2, 3
```

Constraint Directions and Finite Rotations

In geometrically nonlinear analysis steps, the coordinate system in which the constrained degrees of freedom are specified rotate with the reference node regardless of whether the constrained degrees of freedom are specified in the global coordinate system or in a local system. Thus, the constraint shown in *Figure 1* will enable free radial motion throughout arbitrary rotations of the structure. Radial motion in this case is defined as motion normal to the structure's axis (defined in the undeformed configuration by points *a* and *b* in the figure), with this axis rotating with the reference node. Therefore, the free radial expansion shown in *Figure 1* will not refer to an axis parallel to the global *y*-axis for general rotations of the reference node but will refer to an axis that rotates with the structure. Rotation of the constraint directions is not affected by the selection of the constrained degrees of freedom.

Thermal Expansion of a Kinematic Coupling

In Abaqus/Standard a kinematic coupling can experience expansion due to a temperature increase. Each coupled node expands along the line that joins the node to the reference node. The magnitude of the expansion depends on the distance of the coupled 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.

Input File Usage: **KINEMATIC COUPLING*, ALPHA=*thermal_expansion_coeff*

Surface-Based Constraints

You can model kinematic relationships between surfaces to tie two surfaces together, couple a group of nodes to a reference node, couple shell elements to solid elements, and create point-to-point connections between surfaces.

In this section:

- *Mesh Tie Constraints*
- *Coupling Constraints*
- *Shell-to-Solid Coupling*
- *Mesh-Independent Fasteners*

Mesh Tie Constraints

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- **TIE*
- *Defining tie constraints*
- *Using contact and constraint detection*

Overview

A surface-based tie constraint permanently bonds two surfaces.

A surface-based tie constraint:

- ties two surfaces together for the duration of a simulation;
- can be used only with surface-based constraint definitions;
- can be used in mechanical, coupled temperature-displacement, coupled thermal-electrical-structural, coupled thermal-electrochemical, acoustic pressure, coupled acoustic pressure-displacement, coupled pore pressure–displacement, coupled thermal-electrical, or heat transfer simulations;
- can also be used to create a constraint on a surface so that it follows the motion of a three-dimensional beam;
- is useful for mesh refinement purposes, especially for three-dimensional problems;
- allows for rapid transitions in mesh density within the model;
- constrains each of the nodes on the secondary surface to have the same motion and the same value of temperature, pore pressure, acoustic pressure, or electrical potential as the point on the main surface to which it is closest;
- takes the initial thickness and offset of shell elements underlying the surface into account by default; and
- eliminates the degrees of freedom of the secondary surface nodes that are constrained, where possible.

Defining a Tie Constraint for a Pair of Surfaces

A surface-based tie constraint can be used to make the translational and rotational motion as well as all other active degrees of freedom equal for a pair of surfaces. By default, as discussed below, nodes are tied only where the surfaces are close to one another. One surface in the constraint is designated to be the secondary surface; the other surface is the main surface. A name must be assigned to this constraint and might be used in postprocessing with Abaqus/CAE.

Input File Usage: **TIE*, NAME=*name*
secondary_surface_name, *main_surface_name*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**

Defining the Surfaces to Be Constrained

Either element-based or node-based surfaces can be used as the secondary surface. Any surface type (element-based, node-based, or analytical) can be used as the main surface. You can define an element-based surface to contain either the element facets or the element edges (also referred to as an edge-based surface). You should not assume that a constraint over element faces also constrains their edges. If you want to constrain the element edges, a separate constraint must be defined over these edges.

You might need to take some surface restrictions into consideration depending on which tie formulation is used and whether the analysis is conducted in Abaqus/Standard or Abaqus/Explicit. Two tie formulations are available: the surface-to-surface formulation, which is used by default in Abaqus/Standard, and the more traditional node-to-surface formulation, which is used by default in Abaqus/Explicit; these formulations are discussed in more detail later in this section. *Table 1* and *Table 2* provide comparisons of surface restrictions for the different formulations and analysis codes.

Table 1: Comparison of characteristics for surface-based tie formulations.

Tie formulation	Optimized stress accuracy	Node-based surfaces allowed	Mixture of rigid and deformable subregions allowed	Treatment of nodes/facets shared between main and secondary surfaces
Surface-to-surface (Abaqus/Standard or Abaqus/Explicit)	Yes	Reverts to node-to-surface formulation	No	Eliminated from secondary surface
Node-to-surface in Abaqus/Standard	No	Yes	No	Eliminated from secondary surface
Node-to-surface in Abaqus/Explicit	No	Yes	Yes	Eliminated from main surface

Table 2: Comparison of element-based surface characteristics allowed for surface-based tie formulations.

Tie formulation	Surface Characteristics (Yes=allowed, No=not allowed)			
	Double-sided	Discontinuous	T-intersection	Edge-based
Surface-to-surface (Abaqus/Standard or Abaqus/Explicit)	Main: Yes Secondary: Yes	Main: Yes Secondary: Yes	Main: No Secondary: Yes	Main: Yes Secondary: Yes
Node-to-surface in Abaqus/Standard	Main: Yes Secondary: Yes	Main: Yes Secondary: Yes	Main: No Secondary: Yes	Main: Yes Secondary: Yes
Node-to-surface in Abaqus/Explicit	Main: Yes Secondary: Yes	Main: Yes Secondary: Yes	Main: Yes Secondary: Yes	Main: Yes Secondary: Yes

The surface-to-surface formulation generally avoids stress noise at tied interfaces. As indicated in *Table 1* and *Table 2*, only a few surface restrictions apply to the surface-to-surface formulation: this formulation reverts to the node-to-surface formulation if a node-based surface is used. The surface-to-surface formulation does not allow for a mixture of rigid and deformable portions of a surface, and the main surface must not contain T-intersections. Any nodes shared between the secondary and main surfaces are not tied with the surface-to-surface formulation. The same comments apply to both Abaqus/Standard and Abaqus/Explicit in these tables for the surface-to-surface formulation.

With the more traditional node-to-surface formulation additional surface restrictions apply in Abaqus/Standard but fewer restrictions apply in Abaqus/Explicit in comparison to the surface-to-surface formulation. Relatively stringent restrictions on main surface connectivity for the node-to-surface tie formulation in Abaqus/Standard are indicated in *Table 2*: the main surface must be simply connected and must not contain complex intersections

such as T-intersections (see *About Contact Pairs in Abaqus/Standard* for examples of surfaces with various connectivity characteristics).

Differences with the node-to-surface formulation in Abaqus/Explicit are apparent in *Table 1*: partially rigid surfaces can be used and the treatment of shared portions of main and secondary surfaces is unique to this case. Nodes and faces that are shared between the main and secondary surfaces are eliminated automatically from the main surface in this case if the paired surfaces are either both element-based or both node-based, enabling the possibility of tying multiple secondary surfaces (defined over various regions of the model) to a common main surface defined over the entire model. This is a convenient way to define tie constraints in large models, as it eliminates the need for defining specialized main surfaces for each surface pairing; however, you must still take care that secondary surfaces do not include portions of the opposing surface to which they should be tied (for example, no tie constraints are generated if the main and secondary surfaces are identical). In the node-to-surface formulation in Abaqus/Explicit all facets attached to nodes that are common between main and secondary surfaces are excluded from being tied to secondary nodes. Sometimes when meshes are transitioned from one type of element to another type or from one element size to another element size, common nodes might exist at the interface of the two regions. Typically, a tie constraint is defined at the interface of the two zones to stitch the two meshes together. In a situation like this, common nodes might get tied to a neighboring facet on the interface and might cause undesirable mesh distortion because of the tie adjustment. One possible way to avoid the undesirable mesh distortion is to specify a very small position tolerance for the tie pair. Another situation that might arise when common nodes occur between the main and secondary surfaces at the interface of mesh transition zones is that secondary nodes in the vicinity of the common node might not get tied. This happens because of the exclusion of main facets attached to the common nodes. Therefore, care must be taken to ensure that elements in different mesh zones do not share common nodes at the interface. For all such common nodes, duplicate nodes occupying the same physical location should be defined.

Input File Usage: Use the **SURFACE* option to define the secondary and main surfaces used in the constraint (see *About Surfaces*):

**SURFACE*, NAME=*secondary_surface_name*
**SURFACE*, NAME=*main_surface_name*

Abaqus/CAE Usage: In Abaqus/CAE you can select one or more faces directly in the viewport when you are prompted to select a surface. In addition, you can define surfaces as collections of faces and edges using the Surface toolset.

Specifying the Subset of Secondary Nodes to Be Constrained

By default, Abaqus uses a position tolerance criterion to determine the constrained nodes based on the distance between the secondary nodes and the main surface. Alternatively, you can specify a node set containing the secondary nodes to be constrained regardless of their distance to the main surface.

Using the Position Tolerance Criterion

The default position tolerance criterion ensures that nodes are tied only where the secondary and main surfaces are close to one another in the initial configuration. For example, consider the case shown in *Figure 1*. Surfaces `Comp1_surf` and `Comp2_surf` are defined to cover all exposed faces of Component 1 and Component 2, respectively. These two surfaces can be used as the secondary and main surfaces in a tie constraint to tie the two components in the desired region, because only the nodes at the initial interface between the two surfaces are tied.

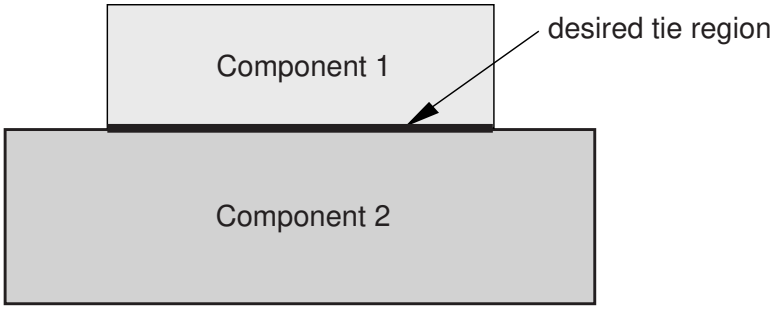


Figure 1: Example of two components to be tied together.

The default value of the position tolerance, d_{tol}, typically results in desired tie constraints with little effort. Details regarding the calculation of distances between surfaces and default values of the position tolerances are provided below. You can modify the position tolerance if desired.

Input File Usage: Use the following option to use the default position tolerance:

**TIE*

Use the following option to specify a position tolerance:

**TIE*, POSITION TOLERANCE=*distance*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: **Position Tolerance**: **Specify distance**

Calculating the Distance between Surfaces

The following factors influence the calculation of the distance between surfaces for a particular secondary node:

- Shell thickness. By default, calculations of distances between surfaces account for shell thickness and offset effects for element-based secondary or main surfaces: the distance is measured from the actual top or bottom side of the surface, whichever is closer to the other surface. Alternatively, you can specify that surface thicknesses and offsets should be ignored, which also has implications for nodal position adjustments for resolving initial gaps (discussed later).

  Input File Usage: Use the following option to ignore surface thicknesses and offsets in the distance calculations:

  **TIE*, NO THICKNESS

  Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: **Exclude shell element thickness**

- Whether the surface-to-surface or node-to-surface constraint formulation (discussed below) is used. If a position tolerance is in effect, a constraint is generated at a secondary node for either formulation if the distance between the surfaces, as calculated at the secondary node, does not exceed d_{tol}. The distance between surfaces at a secondary node is based on a closest-point projection to the main surface for the node-to-surface constraint formulation and is computed along the normal direction to the secondary surface for the surface-to-surface constraint formulation. Additional secondary nodes might be tied if the surface-to-surface constraint formulation is used along with an element-based secondary surface and a main surface that is not node-based. The following addendum to the position tolerance criterion applies in such cases: if the distance

between the surfaces is within d_{tol} over a significant portion of a secondary face (or segment in two dimensions) that forms an angle of less than 30° with the main surface, all secondary nodes attached to such a face (or segment) are considered to satisfy the position tolerance.

- The types of surfaces involved (element-based, node-based, or analytical).

Position Tolerance for an Element-Based Main Surface

The default position tolerance for element-based main surfaces is 5% or 10% of the typical main facet diagonal length for the node-to-surface and surface-to-surface tie formulations, respectively. When using an element-based main surface, the distance between surfaces for a particular point on a secondary surface is based on the closest point on the main surface (which might be on the edge of the main surface or within a facet). *Figure 2*

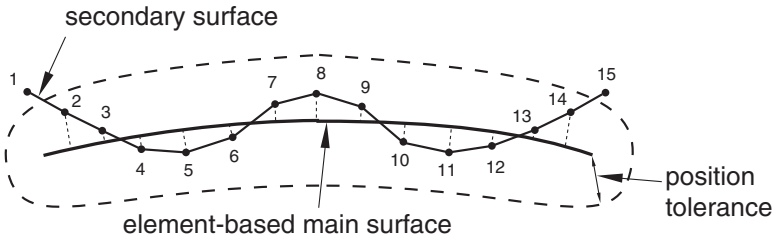


Figure 2: Tolerance region around an element-based main surface with no thickness.

shows an example with no thickness: nodes 2–14 satisfy the position tolerance criterion for the node-to-surface and surface-to-surface constraint formulations. Significant portions of the end secondary segments (that is, the segment connecting nodes 1 and 2 and the segment connecting nodes 14 and 15) are within the position tolerance shown, so nodes 1 and 15 would also satisfy the position tolerance criterion for the surface-to-surface constraint formulation except for the fact that the angle between the secondary and main surfaces is slightly greater than 30° at those locations.

Position Tolerance for a Node-Based Main Surface

The default position tolerance for a node-based main surface is based on the average distance between nodes in the main surface. The distance between the surfaces for a particular secondary node is based on the closest main node. If this distance is less than the position tolerance, Abaqus will create a tie constraint between the secondary node, the closest main node, and other main nodes in similar proximity to the secondary node. For mismatched meshes across a tied interface, the distance between secondary and main nodes can be much larger than the "normal" distance between the surfaces, which can lead to confusion when using a position tolerance criterion with a node-based main surface. *Figure 3* shows how the tolerance region is defined around a node-based main surface. The surface-to-surface constraint formulation reverts to the node-to-surface constraint formulation for a node-based main surface.

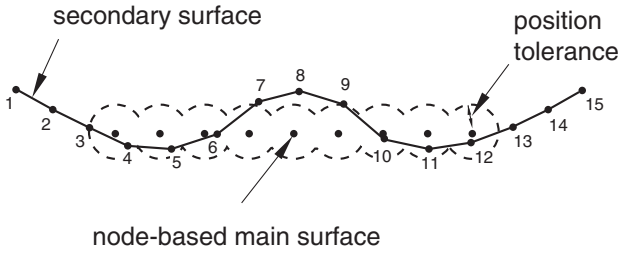


Figure 3: Tolerance region around a node-based main surface with no thickness.

Position Tolerance for an Analytical Rigid Main Surface

The default position tolerance for tie constraints between an element-based secondary surface and an analytical rigid main surface is 5% or 10% of the typical secondary facet diagonal length for the node-to-surface and surface-to-surface tied formulations, respectively. The default position tolerance for tie constraints between a node-based secondary surface and an analytical rigid main surface is 5% of the typical distance between secondary nodes. When using an analytical rigid main surface, the distance between surfaces for a particular point on the secondary surface is based on the closest point on the main surface.

Specifying the Constrained Nodes Directly

This method allows you direct control over which secondary nodes are tied.

Input File Usage: **TIE*, TIED NSET=*node_set_label*

Abaqus/CAE Usage: Specifying the constrained nodes directly is not supported in Abaqus/CAE.

Unconstrained Nodes in Tie Constraint Pairs

Abaqus does not constrain secondary nodes to the main surface unless they are included in the tied node set or within the tolerance distance from the main surface at the start of the analysis, as discussed above. Any secondary nodes not satisfying these criteria will remain unconstrained for the duration of the simulation; they will never interact with the main surface as part of the tie constraint. In mechanical simulations an unconstrained secondary node can penetrate the main surface freely unless contact is defined between the secondary node and main surface. The general contact algorithms in Abaqus/Standard and Abaqus/Explicit will generate contact exclusions automatically for secondary node–main surface combinations corresponding to constrained nodes of tie constraint pairs, but no such contact exclusions are generated for nodes outside the position tolerance of the constraints. In a thermal, acoustic, electrical, or pore pressure simulation an unconstrained secondary node will not exchange heat, fluid pressure, electrical current, or pore fluid pressure with the main surface.

Determining Which Secondary Nodes Have Been Tied and Which Secondary Nodes Have Not Been Tied

For each tie constraint pair, Abaqus creates a node set comprising secondary nodes that will be tied and a node set comprising secondary nodes that will be left unconstrained. These node sets are available for display during postprocessing in Abaqus/CAE, where they are listed as internal node sets.

In addition, Abaqus prints a table in the data (`.dat`) file listing each secondary node and the main surface nodes to which it will be tied if model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). If a constraint cannot be formed for a given secondary node, Abaqus/Standard issues a warning message in the data file.

In Abaqus/Explicit you can also request two nodal field output variables: TIEDSTATUS will help you identify the constrained and unconstrained secondary nodes, and TIEADJUST will help you visualize the adjustment performed at the nodes (see *Abaqus/Explicit Output Variable Identifiers*). A tied node that participates in more than one tie definition as a secondary node as well as a main node is shown as “tied” regardless of whether it got tied as a secondary node or as a main node.

When creating a model with surface-based tie constraints, it is important to use the information provided by Abaqus to identify any unconstrained nodes and to make any necessary modifications to the model to constrain them.

Constraining the Rotational Degrees of Freedom

By default, Abaqus will constrain the rotational degrees of freedom when they exist on both secondary and main surfaces (see *Figure 4*).

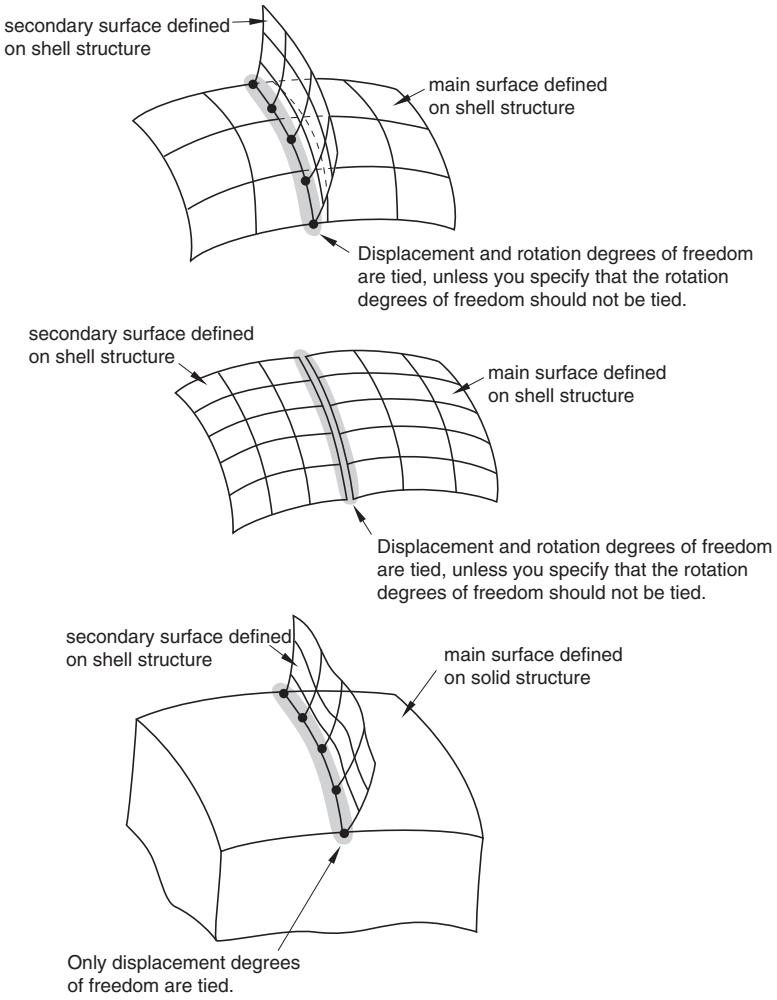


Figure 4: Surface-based tie algorithm.

You can specify that the rotational degrees of freedom should not be tied.

Input File Usage: **TIE*, NO ROTATION

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: toggle off **Tie rotational DOFs if applicable**

Constraining the Faces of a Cyclic Symmetric Structure in Abaqus/Standard

You can enforce proper constraints on the faces bounding a repetitive sector of a cyclic symmetric structure (see *Analysis of Models that Exhibit Cyclic Symmetry*). This makes it possible to define a single sector of the cyclic symmetry model together with its axis of cyclic symmetry to define the behavior of the 360° model. Cyclic symmetry models can be used within the following procedures: static; quasi-static; eigenfrequency extraction, based on the Lanczos solver technique; steady-state dynamics, based on modal superposition; and heat transfer.

If an eigenfrequency extraction is performed on a cyclic symmetric model, the nodes involved in the cyclic symmetry constraint cannot be used in any other constraint (e.g., multi-point constraints, equations, rigid bodies, couplings, or kinematic couplings).

Input File Usage: **TIE*, CYCLIC SYMMETRY

This parameter can be used only with the **CYCLIC SYMMETRY MODEL* option.

Abaqus/CAE Usage: Interaction module: **Interaction->Create**: **Cyclic symmetry**

The Surface-Based Tie Constraint Formulation

Abaqus uses the criteria discussed above to determine which secondary nodes will be tied to the main surface. Abaqus then forms constraints between these secondary nodes and the nodes on the main surface. A key aspect in forming the constraint for each secondary node is determining the tie coefficients. These coefficients are used to interpolate quantities from the main nodes to the tie point. Abaqus can use one of two approaches to generate the coefficients: the "surface-to-surface" approach or the "node-to-surface" approach.

If an analysis carried out with Abaqus/Standard is imported into Abaqus/Explicit or vice versa, the tie constraints are not imported and must be redefined. If the imported analysis is essentially a continuation of the original analysis, it is important that the tie constraints are as similar as possible. Hence, you should make sure that the same constraint type is used. If the default approach was used in the original Abaqus/Standard analysis, the surface-to-surface approach should be specified in the Abaqus/Explicit analysis. Similarly, if the default approach was used in the original Abaqus/Explicit analysis, the node-to-surface approach should be specified in the Abaqus/Standard analysis.

The "Surface-to-Surface" Approach

The "surface-to-surface" approach minimizes numerical noise for tied interfaces involving mismatched meshes. The surface-to-surface approach enforces constraints in an average sense over a finite region, rather at discrete points as in the traditional node-to-surface approach. The surface-to-surface formulation for surface-based tie constraints is similar to the surface-to-surface contact formulation (see *Contact Formulations in Abaqus/Standard*); however, a fundamental difference is that each surface-based tie constraint involves only one secondary node (and multiple main nodes), whereas each surface-to-surface contact constraint involves multiple secondary nodes.

The surface-to-surface approach is used by default in Abaqus/Standard with exceptions noted below, and it is optional in Abaqus/Explicit. For the case of infinite acoustic elements tied to shell elements in Abaqus/Standard the added cost of the surface-to-surface approach can be quite significant; therefore, the node-to-surface approach is used by default in this case. If the surface-to-surface approach is "on by default" or explicitly specified, Abaqus automatically reverts to the node-to-surface approach for individual tie constraints in the following circumstances:

- if either of the surfaces being tied is node-based;
- if the projection along the secondary surface normal direction does not intersect the main surface; or
- if single-sided secondary and main surfaces have surface normals in approximately the same direction.

Abaqus/Explicit might automatically add a small amount of artificial mass to the model to maintain numerical stability if the surface-to-surface approach is specified.

The surface-to-surface approach generally involves more main nodes per constraint than the node-to-surface approach, which tends to increase the solver bandwidth in Abaqus/Standard and, therefore, can increase solution cost. In most applications the extra cost is fairly small, but the cost can become significant in some cases. The following factors (especially in combination) can lead to the surface-to-surface approach being quite costly:

- A large fraction of tied nodes (degrees of freedom) in the model

- The main surface being more refined than the secondary surface
- Multiple layers of tied shells, such that the main surface of one tie constraint acts as the secondary surface of another tie constraint

Input File Usage: **TIE*, TYPE=SURFACE TO SURFACE

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: **Discretization method: Surface to surface**

The "Node-to-Surface" Approach

The traditional "node-to-surface" approach (which is used by default in Abaqus/Explicit and is optional in Abaqus/Standard) sets the coefficients equal to the interpolation functions at the point where the secondary node projects onto the main surface. This approach is somewhat more efficient and robust for complex surfaces.

For the node-to-surface method of establishing the tie coefficients with an element-based main surface, the point on the surface closest to each secondary node is calculated and used to determine the main nodes that are going to form the constraint (see *Figure 5*). For example, nodes 202, 203, 302, and 303 are used to constrain node *a*; nodes 204 and 304 are used to constrain node *b*; and node 402 is used to constrain node *c*.

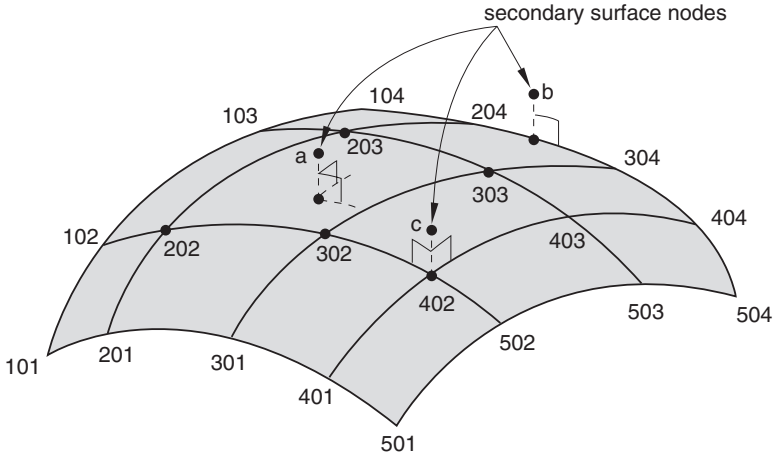


Figure 5: Searching for the points on an element-based main surface that are closest to nodes *a*, *b*, and *c*.

Input File Usage: **TIE*, TYPE=NODE TO SURFACE

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: **Discretization method: Node to surface**

Choosing the Secondary and Main Surfaces of a Surface-Based Tie Constraint

The choice of secondary and main surfaces can have a significant effect on the accuracy of the solution, in particular if the "node-to-surface" approach is used. The effect is much less (and the accuracy generally better) for the "surface-to-surface" approach. In either case, if both surfaces in a constraint pair are deformable surfaces, the main surface should be chosen as the surface with the coarser mesh for best accuracy.

In Abaqus/Standard a rigid surface cannot act as a secondary surface in a tie constraint. To comply with this rule, the capability to automatically resolve overconstraints in Abaqus/Standard (see *Overconstraint Checks*) will modify tie constraint definitions in the following cases:

- Tie constraints between two surfaces of the same rigid body are removed.
- Tie constraints between two surfaces of two rigid bodies are replaced by a BEAM-type connector between the respective rigid body reference nodes.
- Tie constraints specified with a purely rigid secondary surface and a purely deformable main surface are modified to reverse the main and secondary assignments unless this is not possible because of other modeling restrictions (in which case an error message is issued).

These methods are not applied if the secondary surface that you specified is partially rigid and partially deformable; Abaqus/Standard issues an error message in such cases.

In acoustic, structural-acoustic, and elastic wave propagation problems care should be exercised when tying meshes of highly dissimilar refinement. If two media have different wave speeds, the optimal meshes for each of the media will have different characteristic element lengths: the faster medium will have larger elements. If surfaces of these meshes are used in a tie constraint, the surface of the finer mesh (of the slower medium) should be designated as the secondary. Nevertheless, in the region near the tied surfaces, the physical wave phenomena in *both* fast and slow media will typically have length scales characteristic of the slower medium; that is, of the shortest length scale in the physical problem. Therefore, if these phenomena are important, the mesh of the faster medium should be refined to the scale of the slower medium in the vicinity of the contact region.

Adjusting the Surfaces and Considering Offsets

By default, with the exceptions mentioned below, Abaqus will automatically reposition the secondary nodes to be tied in the initial configuration without causing strain to resolve gaps such that the surfaces are just touching, accounting for any shell thickness (unless you have specified that thickness should not be accounted for, as discussed above in the context of the position tolerance criterion) but not accounting for beam or membrane thickness. One exception is that no adjustments are made where tied surfaces are closer together than the combined half-shell thickness. All adjustments are performed such that the secondary and main surfaces are never pushed apart; that is, the reference surfaces will only become closer as a result of the adjustments.

It is recommended that you allow the automatic adjustments to occur, especially if neither surface has rotations; in this case a constant offset vector is used, so incorrect behavior of the constraint under rigid body rotation can occur when secondary nodes are not lying exactly on the main surface. Adjustments are not made if the secondary surface belongs to a substructure or when either the secondary or main surface is a beam element-based surface; in the latter cases you should locate the beam element nodes with the desired offset from the other surface.

Input File Usage: **TIE*, ADJUST=YES *or* NO

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: toggle **Adjust secondary surface initial position**

Criteria for Adjustment

A secondary node is considered for adjustment if both of the following conditions are met:

- The secondary node satisfies whatever criterion is in effect for generating a constraint (either because it satisfies the position tolerance criterion or belongs to the specified node set of constrained secondary nodes, as previously discussed).

- The secondary node is more than the combined thickness of the secondary and main surfaces away from its projection point on the main reference surface, accounting for any offset of the element reference surfaces from the respective element midsurfaces.

For an element-based main surface a secondary node will be moved toward the closest point on the main surface; for a node-based main surface a secondary node will be moved toward the closest main node. The corrected position of an adjusted secondary node is determined from the combined effects of shell element thickness and any specified reference surface offset relative to the shell midsurface of either secondary or main surfaces. *Figure 6* shows the adjusted secondary node position in an example with two shell element-based surfaces tied together (in this example one of the element reference surfaces is offset from the element midsurface). It is assumed that the surfaces were farther apart than shown in *Figure 6* prior to the adjustment; otherwise, the secondary nodes would not have been adjusted.

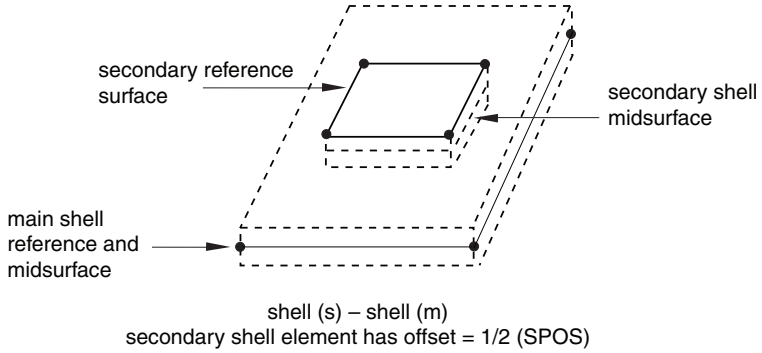


Figure 6: Adjusted secondary node position for two shell element-based surfaces tied together. The secondary shell element has an offset of 0.5.

Adjustments are made only for secondary nodes that are included in the user-specified tied node set or that meet the tolerance criteria described above.

Adjustments for Overlapping Constraints

Nodal adjustments for tie constraints are processed sequentially in the order of the constraint definitions at the start of an analysis. If different constraint or contact definitions involve the same nodes, some adjustments might cause lack of compliance for contact or constraint definitions that were previously processed. These conflicts are less likely to occur in Abaqus/Explicit because the adjustments in Abaqus/Explicit are automatically processed in the chaining order discussed in *Overlapping Constraints*. These conflicts can be avoided in Abaqus/Standard in some cases by changing the processing order of constraint and contact definitions: nodes in common between different contact or constraint definitions should be processed first as secondary nodes and later as main nodes.

Input File Usage: To change the processing order of constraint and contact definitions, change the order of the definitions in the input file. Constraint and contact definitions are processed in the order in which they appear.

Abaqus/CAE Usage: To change the processing order of constraint and contact definitions, change the names of the constraints and interactions in the model. Constraints and interactions are processed alphabetically according to their name.

Accounting for an Offset between Tied Surfaces

Abaqus allows a gap to exist between tied surfaces. Such gaps might exist if you prevent nodal adjustments for tied surfaces. A gap between the reference surfaces might remain because of the presence of shell thickness even

if nodal adjustments are performed. *Figure 7* shows some cases where an offset between the reference surfaces might be desirable for tied surface pairs to account for shell or beam thickness.

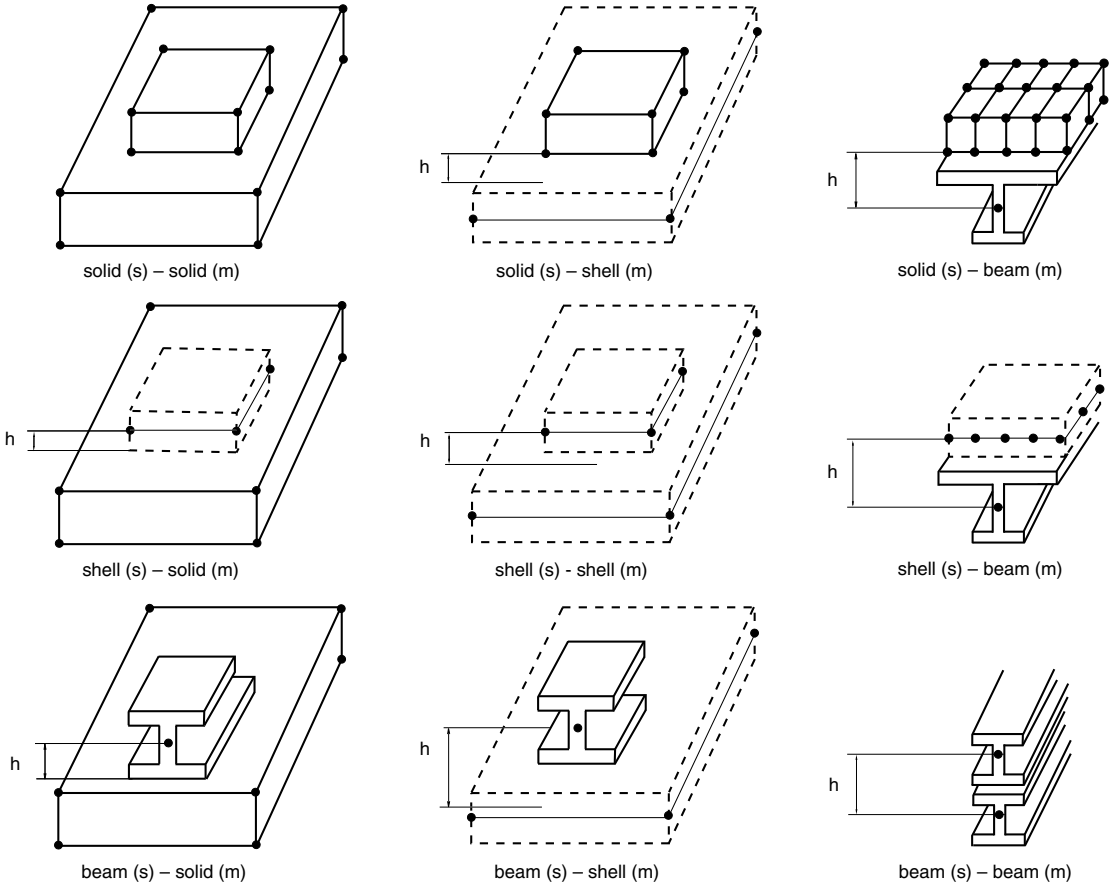


Figure 7: Tie constraints being applied between surfaces based on various element types (h = offset between secondary and main surfaces).

Rigid body motion is properly accounted for when the nodes are separated by a finite distance when at least one of the surfaces is based on shell or beam elements; when the main surface is an analytical rigid surface; or, in the case of node-based surfaces, when the nodes on at least one surface have active rotational degrees of freedom.

The nature of the constraint on translational motion between surfaces in Abaqus depends on whether there is an offset between the surfaces and on which surfaces have rotational degrees of freedom, as discussed below.

Neither Surface Has Rotational Degrees of Freedom

If neither surface has rotational degrees of freedom, the global translational degrees of freedom of the secondary node and the closest point on the main surface are constrained to be the same. When an offset exists, the behavior of Abaqus/Standard differs from that of Abaqus/Explicit.

Abaqus/Explicit enforces the constraint through the fixed offset like a PIN-type MPC when the nodes in the MPC are not coincident. Because the fixed offset does not rotate, the surface-based constraint will not represent rigid body rotation correctly. The constraint represents rigid body motion correctly when the offset is zero. This behavior can be ensured by specifying that all tied secondary nodes should be moved onto the main surface. If an offset needs to be maintained, general contact with surface-based cohesive behavior (as explained in *Contact Cohesive Behavior*) that correctly accounts for rigid body rotation of the offset should be used.

In general, Abaqus/Standard enforces the constraint such that the surface-based constraint represents rigid body rotation correctly; the enforcement of this constraint will introduce nonlinearity in the model. There are, however,

two exceptions in which rigid body rotation between the tied surfaces cannot be enforced: (1) when node-based main surfaces are used and (2) when using tie constraints for cyclic symmetry.

Only One Surface Has Rotational Degrees of Freedom

If the secondary surface has rotational degrees of freedom and the main surface does not, the translational motion is constrained at the closest point on the main reference surface. When the reference surfaces are offset, a moment will be applied to each secondary node based on the constraint force times the offset distance. Similarly, if the main surface has rotational degrees of freedom and the secondary surface does not, the translational motion is constrained at each secondary node and a moment will be applied to the relevant nodes on the main surface if an offset exists. In either case the surface-based constraint will behave correctly under rigid body rotation regardless of the amount of offset.

Both Surfaces Have Rotational Degrees of Freedom

If both surfaces have rotational degrees of freedom, are not offset, and the rotations are tied, each secondary node is constrained to the main surface like a TIE-type MPC. If an offset exists between the surfaces, the constraint acts like a BEAM-type MPC between the secondary node and the closest point on the main reference surface.

If the rotations are not tied, Abaqus allows you to choose the location of the translational constraint. It can be enforced at the main reference surface, the secondary reference surface, or anywhere in between. The location of the translational constraint enforcement for surfaces where the rotations are not tied will affect the distribution of moment to each of the surfaces. The most physically reasonable choice is to locate the constraint at the point where the actual top or bottom sides of each surface meet. The constraint then models a perfect adhesive between the surfaces, which transfers shear stress to each surface. Abaqus will choose the location of the translational constraint as follows:

- If the main surface is shell element-based, the translational constraint is enforced on the top or bottom side of the main surface.
- If the secondary surface is shell element-based and the main surface is not, the translational constraint is enforced at the top or bottom side of the secondary surface.
- Otherwise, the translational constraint is enforced at the main reference surface.

To override these default locations, you can specify a constraint ratio for the tie constraint equal to the fractional distance between the main reference surface and the secondary node at which the translational constraint should act. *Figure 8* shows an example of the use of a constraint ratio to prescribe the location of the translational constraint between two shell surfaces that are tied together with no rotational constraints. The distance between the main reference surface and the secondary reference surface is b. The prescribed constraint ratio, r, is then used to locate the translational constraint at a distance a from the main reference surface. All distances are measured along the vector between the secondary node and its projection point onto the main reference surface. The constraint behavior is then similar to that of two rigid beams pinned together, as shown.

Input File Usage: **TIE*, CONSTRAINT RATIO=*value*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**: **Constraint ratio**

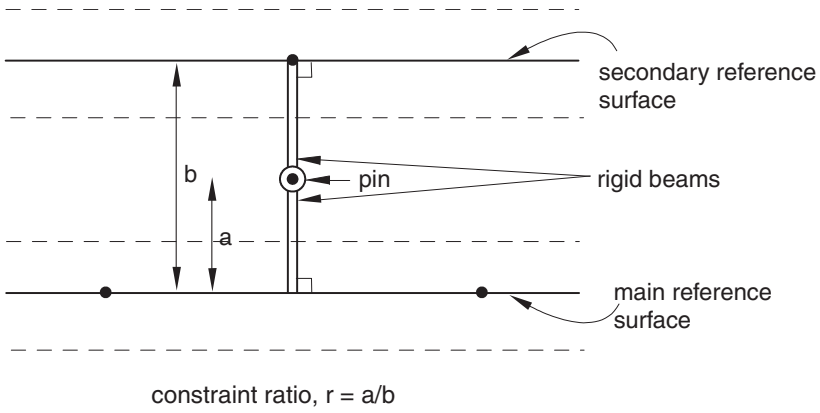


Figure 8: Use of a constraint ratio to prescribe the location of the translational constraint.

Constraining a Surface to a Three-Dimensional Beam

The main surface for a tie constraint can be based on three-dimensional beam elements. For this case each secondary node is projected onto the line formed by the nodes of the beam elements in the undeformed configuration to find the projection point. During the subsequent analysis the motion of each secondary node is rigidly constrained to the motion (translation and rotation) of its projection point; that is, each secondary node and its projection point are connected by a rigid beam. Constraining other elements to a beam element-based main surface allows modeling of interactions between the surface of a (complex) beam section and its surroundings, without having to model the beam with continuum and/or shell elements. This feature can be particularly useful for modeling acoustic-structural interactions.

Use of Tie Constraints in Nonmechanical Simulations

The surface-based tie constraint capability can be used in models where the nodal degrees of freedom on both the secondary and main surfaces include electrical potential, pore pressure, acoustic pressure, and/or temperature. Except for the type of nodal degree of freedom being constrained, Abaqus uses exactly the same formulation for the tie constraint in nonmechanical simulations as it does for mechanical simulations. In general, degrees of freedom common to both surfaces are tied, and any other degrees of freedom are unconstrained. For example, a thermal tie between a solid element and a shell element constrains only degree of freedom 11.

The case of structural-acoustic constraints is the exception to this rule. Here, appropriate relations between the acoustic pressure on the fluid surface and displacements on the solid surface are formed internally (see *Coupled Acoustic-Structural Analysis*). The displacements and/or pressure degrees of freedom on the surfaces are the only ones affected; rotations are ignored by the tie constraint in this case.

The internally computed structural-acoustic coupling conditions use surface areas and normal directions associated with the secondary surface elements. The secondary surface for structural-acoustic tie constraints cannot be a node-based surface. In two-dimensional analyses the out-of-plane thickness of the secondary elements is required. Generally, this thickness is the thickness specified on the section definition for the secondary surface elements. However, when beam elements form the secondary surface in a tie constraint pair with acoustic elements, a unit thickness in the out-of-plane direction is assumed for the beams.

In Abaqus/Standard you can define coupling between solid medium and acoustic medium infinite elements along the surfaces that extend to infinity. These surfaces are defined using the edges of the acoustic elements and sides numbered “2” and higher of the solid medium infinite elements. The infinite surfaces of solid medium and acoustic infinite elements can be coupled only through the use of a surface-based tie constraint. As shown in *Figure 9*, the acoustic infinite elements must be the secondary elements and the edges of the acoustic infinite elements should lie within the specified position tolerance to the solid medium infinite element base facets.

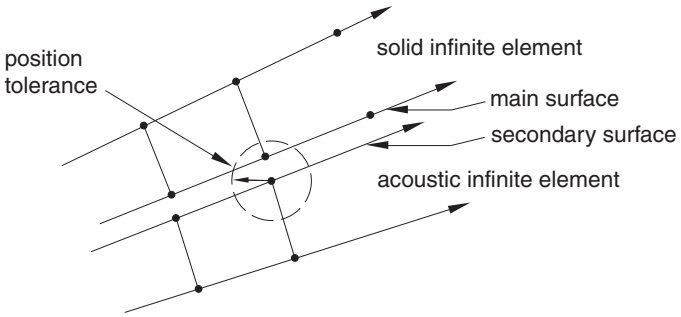


Figure 9: Use of a surface-based tie constraint to prescribe the coupling between solid medium and acoustic medium infinite elements.

If the base facets of acoustic infinite elements are to be coupled to solid medium finite elements, to solid medium infinite elements, or to structural elements, either a surface-based tie constraint or acoustic-structural interaction elements can be used. Surfaces defined on solid medium infinite elements cannot be used in a surface-based tie constraint in Abaqus/Explicit.

Table 3 enumerates all possible cases. For other secondary-main pairings not listed in this table, an error message is issued.

Table 3: Possible secondary-main surface pairings.

Secondary Surface	Main Surface	Degrees of Freedom Tied
Acoustic	Acoustic	Acoustic pressure
Acoustic	Stress	Translations
Stress	Acoustic	Acoustic pressure
Stress	Stress	Translations and/or rotations
Heat-Stress	Stress	Translations and/or rotations
Stress	Heat-Stress	Translations and/or rotations
Heat-Stress	Heat-Stress	Temperature, translations and/or rotations
The following surface pairings are available only in Abaqus/Standard:		
Heat transfer	Heat transfer	Temperature
Electrical-Heat	Heat transfer	Temperature
Heat transfer	Electrical-Heat	Temperature
Electrical-Heat	Electrical-Heat	Temperature and solid electric potential
Pore-Stress	Pore-Stress	Pore pressure and translations
Pore-Stress	Stress	Translations
Stress	Pore-Stress	Translations
Pore-Stress	Pore	Pore pressure
Electrochemical-Heat	Electrochemical-Heat	Temperature, electric potential in the solid, electric potential in the fluid, and ion concentration

You can specify that certain degrees of freedom should not be tied between the surfaces involved in a tie constraint. These degrees of freedom are pore pressure, solid electric potential, temperature, fluid electric potential, and ion concentration.

Input File Usage: **TIE*, NO PORE
**TIE*, NO ELECTRIC POTENTIAL
**TIE*, NO TEMPERATURE
**TIE*, NO FLUID ELECTRIC POTENTIAL
**TIE*, NO ION CONCENTRATION

Abaqus/CAE Usage: In Abaqus/CAE pore pressure, solid electric potential, and temperature degrees of freedom common to both surfaces in a tie constraint are always tied if applicable.

Defining fluid electric potential and ion concentration are not supported in Abaqus/CAE.

Tie Constraints Versus Tied Contact in Abaqus/Standard

There are the following advantages to using a surface-based tie constraint in Abaqus/Standard instead of defining tied contact as discussed in *Defining Tied Contact in Abaqus/Standard*:

- Degrees of freedom of the secondary surface nodes will be eliminated.
- The tie constraint is more efficient in terms of the size of the fronts of the operator matrix because fewer main surface nodes are associated with each secondary node.
- Rotational degrees of freedom as well as translational degrees of freedom can be tied.
- Tie constraints are much more general since they allow the use of general surfaces.
- Surface offsets and shell thickness are taken into account.

Overlapping Constraints

In a model with multiple tie constraint definitions it is possible that the secondary and main surfaces of different tie constraint definitions might intersect. If two tie constraint definitions have part or all of their main surfaces in common or if the surfaces tied are layered (that is, the main surface of one tie constraint definition acts as the secondary surface of a subsequent tie constraint definition), Abaqus will attempt to chain the constraint definitions together. This will reduce the number of degrees of freedom and lower the computational expense, resulting in faster run times. However, in a model with multiple tie constraint definitions and nodes on the secondary surface of one tie constraint definition are part of the secondary surface of other tie constraint definitions, an overconstraint occurs. In most cases the overconstraint is because of the existence of redundant constraints, and it is safe to eliminate this redundancy. However, the overconstraint might also be due to conflicting constraints, in which case the problem is because of a modeling error that you should correct. Simulation results will vary depending on which constraint is removed to avoid an overconstraint if the overlapping constraints are not identical. It is recommended that, wherever possible, you order the secondary and main surfaces of the constraint definitions to avoid intersecting secondary surfaces. See *Adjustments for Overlapping Constraints* for a discussion of initial strain-free adjustments for overlapping constraints. When secondary nodes of a tie constraint also participate in other types of constraints, Abaqus will replace these nodes in the other constraints with corresponding main nodes from the tie constraint.

Overconstrained Secondary Nodes in Abaqus/Standard

If an overconstraint occurs, Abaqus/Standard issues an error message unless the constraints are redundant or nearly redundant, as discussed below. As discussed previously, each tie constraint involves a single secondary node and a set of main nodes with nonzero tie coefficients. Abaqus/Standard considers tie constraints involving the same secondary node to be nearly redundant if at least one node is common among the respective sets of

main nodes with nonzero tie coefficients. In such cases, rather than issuing an error message, Abaqus/Standard issues a warning message and only enforces one of the constraints.

The surface-based tie constraint is imposed in Abaqus/Standard by eliminating the degrees of freedom on the secondary surface; therefore, nodes on the secondary surface should not be used to apply boundary conditions, nor should they be used in any subsequent tie, multi-point, equation, or kinematic coupling constraint (see *Overconstraint Checks* for a more complete discussion of overconstraints in Abaqus/Standard).

Overconstrained Secondary Nodes in Abaqus/Explicit

In contrast, Abaqus/Explicit treats overconstraints with a penalty method, thus enforcing the constraints in an average sense; the computational cost of the analysis might increase in these cases.

In addition, if the secondary surface for a tie constraint definition in Abaqus/Explicit is part of a rigid body while the main surface comprises a deformable element- or node-based surface and the main surface acts as the secondary surface in a subsequent tie constraint definition, the resolution of the resulting constraints can prove to be computationally intensive. It is recommended that, wherever possible, you order the secondary and main surfaces of the constraint definitions to avoid such a situation.

Nullifying the Tie Constraint on Secondary Nodes because of Element Deletion in Abaqus/Explicit

In Abaqus/Explicit tie constraints are nullified as underlying elements of tied surfaces are deleted because of material point failure. The tie constraint between a secondary node and its corresponding main nodes is deleted when either all the elements attached to the secondary node are deleted or the main element to which the secondary node is tied is deleted.

Limitations

The following limitations exist for tie constraints:

- Surface-based tie constraints cannot be used to connect gasket elements that model thickness-direction behavior only.
- A rigid surface cannot act as a secondary surface in a constraint pair in Abaqus/Standard.
- A secondary node of a tie constraint cannot act as a secondary node of another constraint in Abaqus/Standard.
- Tie constraints cannot be used to tie infinite elements to finite elements in Abaqus/Explicit. To couple infinite and finite elements in Abaqus/Explicit, the elements must share nodes.
- The axisymmetric solid Fourier elements with nonlinear, asymmetric deformation cannot form element-based surfaces; therefore, such surfaces cannot be used in tie constraints.
- In Abaqus/Standard, tie constraints cannot be used to connect nodes included in a node-based surface or nodes included in an element-based surface defined using an element edge identifier if such nodes have more than one temperature degree of freedom.

Coupling Constraints

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- **COUPLING*
- **KINEMATIC*
- **DISTRIBUTING*
- **UNIFORM*
- *Defining coupling constraints*

Overview

A surface-based coupling constraint couples the motion of a collection of nodes on a surface with the motion of a reference node.

A surface-based coupling constraint:

- is of type kinematic when the coupling enforces rigid body motion among the system of nodes;
- is of type distributing when the reference node is constrained in an average sense to the group of surface nodes;
- is of type uniform when the coupling enforces uniformity of nonmechanical degrees of freedom among the system of nodes;
- automatically selects the coupling nodes located on a surface lying within a region of influence;
- can be used with two- or three-dimensional stress/displacement elements; and
- can be used in geometrically linear and nonlinear analysis.

Surface-Based Coupling Definitions

The surface-based coupling constraint in Abaqus provides coupling between a reference node and a group of surface nodes. Surface-based coupling constraints are useful for transmitting loads or imposing conditions on a surface region of a part. The loads or imposed conditions can be specified directly, or they can arise from discrete connections, such as spotwelds, between surfaces. Use of a surface-based coupling rather than applying a load or imposing a condition at a single surface node reduces sensitivity of simulation results to mesh details.

Key differences between the available surface-based coupling types include whether rigidity is imposed among nodes involved in the coupling and which degrees of freedom are associated with the coupling.

- Kinematic coupling acts on mechanical degrees of freedom and imposes rigidity on the system of nodes involved.
- Distributing coupling acts on mechanical degrees of freedom and does not impose rigidity; instead, the reference node displacement and rotation correspond to a weighted-average motion among surface nodes involved in the coupling (taking into consideration any offset of the reference node from the center of the surface nodes).

The constraint equations associated with distributing coupling have the effect of distributing a force or moment acting on the reference node among the surface nodes of the coupling. Uniform coupling imposes uniformity of nonmechanical degrees of freedom among the coupled nodes. A uniform coupling constraint can act on the same

reference node and surface as a kinematic or distributing constraint because different (nonmechanical versus mechanical) degrees of freedom are involved.

Surface-based coupling provides functionality similar to the kinematic coupling constraint element and the distributing coupling element types (DCOUP2D, DCOUP3D) in Abaqus/Standard; however, it has a surface-based user interface. Abaqus selects the group of surface nodes automatically by specifying a surface and an optional influence region. The procedure used to define the coupling nodes is discussed below.

Abaqus automatically calculates the weight factors associated with averaging of a distributing coupling constraint if the surface is an element-based surface. In such a case, the weight factors are based on the tributary area at each coupling node, except for a surface along a shell edge, where the weight factors are based on the tributary edge length. In addition, you can modify the distributing weight factors using one of several weighting methods, which allow the forces transferred to the coupling nodes to vary inversely with the radial distance from the reference node.

Typical Applications

You can use surface-based coupling constraints effectively in the following situations:

- To distribute loads on a model, where the load distribution can be described with a moment-of-inertia expression. Examples of such cases include the classic bolt-pattern and weld-pattern distribution expressions.
- To apply dimensionality transitions between continuum and structural elements. For example, a distributing coupling allows flexible coupling between structural and solid elements.
- To model end conditions. For example, modeling a rigid end plate or modeling plane sections of a solid to remain planar can be done easily with a kinematic coupling definition.
- To simplify modeling of complex constraints. In a kinematic coupling definition the degrees of freedom that participate in the constraint can be selected individually in a local coordinate system.
- To model interactions with other constraints, such as connector elements. For example, a hinged part can be modeled more realistically by two distributing coupling definitions, whose reference nodes are connected by a hinge connector element. The load transfer then occurs between two “clouds” of nodes, rather than between two single nodes. *Substructure analysis of a one-piston engine model* illustrates this use of connector elements with coupling constraints to model a one-piston engine.
- To constrain nonmechanical degrees of freedom of the nodes on a surface to a uniform value with a uniform coupling. The kinematic and distributing couplings act on mechanical degrees of freedom (displacements and rotations) only. A uniform coupling constraint can be used together with a kinematic or distributing coupling constraints (such as the ones shown in *Figure 1* or *Figure 2*), or independently.

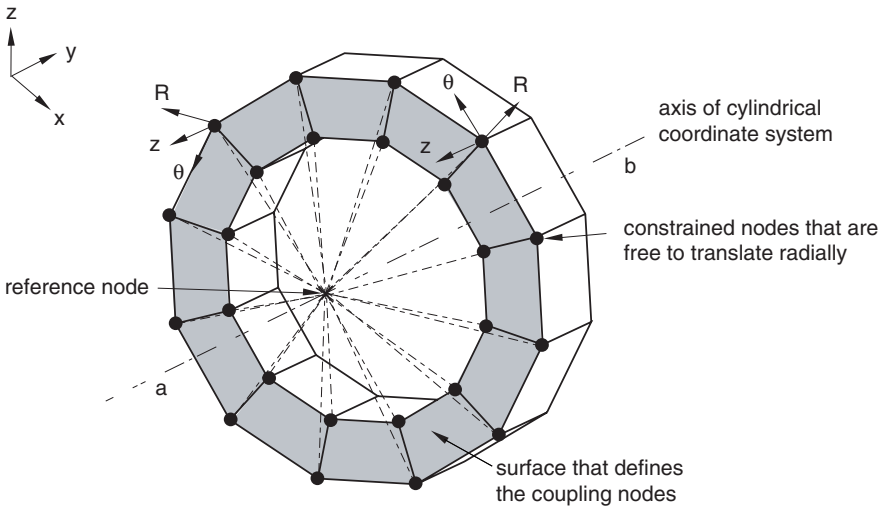


Figure 1: Kinematic coupling constraint.

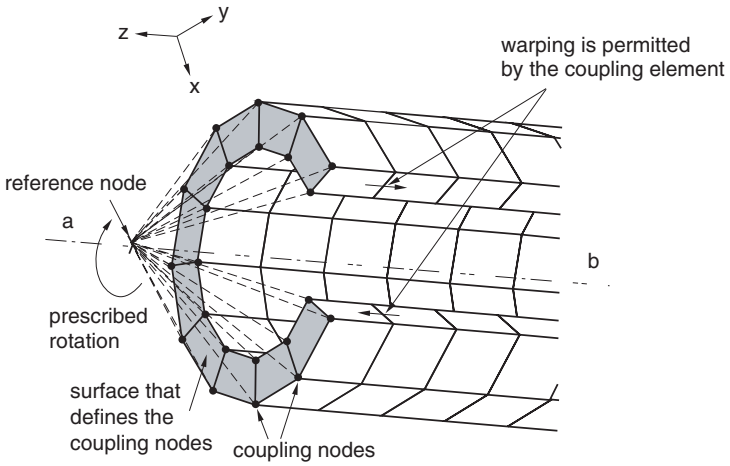


Figure 2: Distributing coupling constraint.

Defining a Surface-Based Coupling Constraint

Defining a coupling constraint requires the specification of the reference node (also called the constraint control point), the coupling surface, and the constraint type. The coupling constraint associates the reference node with the nodes of the coupling surface. A name must be assigned to the constraint and can be used in postprocessing with Abaqus/CAE. A node number or node set name can be specified for the reference node. When the reference node set contains multiple nodes, you should specify a radius of influence to avoid overconstraint issues. The reference node for a kinematic coupling constraint has both translational and rotational degrees of freedom. The surface on which the coupling nodes are located can be node-based; element-based; or, in Abaqus/Explicit, a combination of both surface types. You can specify an optional radius of influence to limit the coupling nodes to a specific region on the surface. Details on how coupling nodes are defined by specifying an influence region are discussed below.

The constraint type can be either kinematic or distributing, as discussed below.

Input File Usage: Use the following options:

**COUPLING*, CONSTRAINT NAME=*name*, REF NODE=*n*, SURFACE=*surface*
**KINEMATIC*, **DISTRIBUTING*, *or *UNIFORM*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Kinematic**, **Continuum distributing**, or **Structural distributing**

Specifying a Region of Influence

By default, coupling nodes belonging to the entire surface are selected for the coupling definition. You can limit the coupling nodes to lie within a spherical region centered about the reference node by defining a radius of influence.

The procedure by which coupling nodes are selected for the constraint definition depends on the surface type:

- For a node-based surface, all the nodes defined by the surface definition that fall within the influence region are selected for the coupling definitions.

- For an element-based surface, the surface facets that are either fully or partially inscribed by the influence region are determined. All nodes belonging to these facets, whether or not these nodes fall within the influence region, are selected for the coupling nodes. When the influence radius is less than the distance to the closest coupling node, Abaqus selects all nodes belonging to the closest facet. If the projection of the reference node on the surface falls on an edge or a vertex of multiple facets, all nodes belonging to these facets adjoining the edge or vertex are included in the coupling definition. In the case where the influence radius is less than the distance to the closest coupling node, adjacent surface faces must have consistent normal directions; otherwise, Abaqus issues an error message.
- A distributing coupling constraint must include at least two coupling nodes. If fewer than two coupling nodes are found, Abaqus issues an error message during input file preprocessing.

Input File Usage: **COUPLING*, CONSTRAINT NAME=*name*, REF NODE=*n*, SURFACE=*surface*, INFLUENCE RADIUS=*r*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Influence radius**: **Specify**

Kinematic Coupling Constraints

Kinematic coupling constrains the motion of the coupling nodes to the rigid body motion of the reference node. The constraint can be applied to user-specified degrees of freedom at the coupling nodes with respect to the global or a local coordinate system. Only required degrees of freedoms are activated at the reference node. If no displacement degree of freedom is constrained, the displacement degrees of freedom at the reference node might not be activated.

Kinematic constraints are imposed by eliminating degrees of freedom at the coupling nodes. In Abaqus/Standard once any combination of displacement degrees of freedom at a coupling node is constrained, additional displacement constraints—such as MPCs, boundary conditions, or other kinematic coupling definitions—cannot be applied to any coupling node involved in a kinematic coupling constraint. The same limitation applies for rotational degrees of freedom. This restriction does not apply in Abaqus/Explicit. See *About Kinematic Constraints* for more information.

Input File Usage: Use both of the following options to define a kinematic coupling constraint:

**COUPLING*
**KINEMATIC*
first dof, *last dof*

For example, the following coupling constraint is used to constrain degrees of freedom 1, 2, and 6 on surface `surfA` to reference node 1000:

**COUPLING*, CONSTRAINT NAME=C1, REF NODE=1000, SURFACE=surfA
**KINEMATIC*
1, 2
6,

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Kinematic**: toggle on the degrees of freedom

Translational Degrees of Freedom

Translational degrees of freedom are constrained by eliminating the specified degrees of freedom at the coupling nodes. When all translational degrees of freedom are specified, the coupling nodes follow the rigid body motion of the reference node.

Rotational Degrees of Freedom

Rotational degrees of freedom are constrained by eliminating the specified degrees of freedom at the coupling nodes.

All combinations of selected rotational degrees of freedom result in rotational behavior identical to existing MPC types:

- Selection of three rotational degrees of freedom along with three displacement degrees of freedom is equivalent to MPC type BEAM.
- Selection of two rotational degrees of freedom is equivalent to MPC type REVOLUTE in Abaqus/Standard.
- Selection of one rotational degree of freedom is equivalent to MPC type UNIVERSAL in Abaqus/Standard.

In Abaqus/Standard, internal nodes are created by the kinematic coupling to enforce the constraints that are equivalent to MPC types REVOLUTE and UNIVERSAL. These nodes have the same degrees of freedom as the additional nodes used in these MPC types and are included in the residual check for nonlinear analysis.

Specifying a Local Coordinate System

The kinematic coupling constraint can be specified with respect to a local coordinate system instead of the global coordinate system (see *Orientations*). *Figure 1* illustrates the use of a local coordinate system to constrain all but the radial translation degrees of freedom of the coupling nodes to the reference node. In this example, a local cylindrical coordinate system is defined that has its axis coincident with the structure's axis. The coupling node constraints are then specified in this local coordinate system.

Input File Usage: **COUPLING*, ORIENTATION=*local*

For example, the following input is used to specify the kinematic coupling constraint shown in *Figure 1*:

```
*ORIENTATION, SYSTEM=CYLINDRICAL, NAME=COUPLEAXIS
0.0, -1.0, 0.0, 0.0, 1.0, 0.0
*COUPLING, REF NODE=500, SURFACE=Endcap,
ORIENTATION=COUPLEAXIS
*KINEMATIC
2, 3
```

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Edit**: select local coordinate system

Constraint Direction and Finite Rotation

In geometrically nonlinear analysis steps the coordinate system in which the constrained degrees of freedom are specified will rotate with the reference node regardless of whether the constrained degrees of freedom are specified in the global coordinate system or in a local coordinate system.

Thermal Expansion of a Kinematic Coupling

In Abaqus/Standard a kinematic coupling can experience expansion because of a temperature increase. Each coupled node expands along the line that joins the node to the reference node. The magnitude of the expansion depends on the distance of the coupled 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. Thermal expansion can be used only when the temperature is a field variable.

Input File Usage:
**KINEMATIC*, ALPHA=*thermal_expansion_coeff*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Kinematic**: **Thermal expansion coefficient**

Distributing Coupling Constraints

Distributing coupling constraints couple translation and rotation of the reference node to the average motion of the coupling cloud nodes. The reference node follows the motion of the cloud nodes in an average sense, without restricting relative motion among the cloud nodes.

A distributing coupling constraint enforces that the rotation of the reference node equals an average rotation measure of the cloud nodes. By default, this averaging typically considers only translations of cloud nodes, such that reference node rotation corresponds to the average "swirling" of cloud nodes about the center of the cloud system. However, if all cloud nodes are approximately collinear, cloud node rotational degrees of freedom (if they exist) influence reference node rotation by default.

A distributing coupling constraint enforces that translation of the reference node equals the average translation of the cloud nodes plus the effect of rotation of a rigid arm from the center of the cloud node system to the reference node. This rigid-arm rotation corresponds to the reference node rotation. Cloud node rotational degrees of freedom can also participate in an additional offset term for the translational coupling constraint such that the reference node remains close to a shell surface during bending.

Forces and moments acting on the reference node are typically distributed as a nodal force distribution among cloud nodes, plus (often small) moments acting at any cloud nodes whose rotations influence translational or rotational constraints. The coupling constraints distribute loads such that the resultants of forces and moments at the cloud nodes are equivalent to the forces and moments at the reference node. For cases of more than a few cloud nodes, the distribution of forces/moments is not determined by equilibrium alone; distributing weight factors are used to define the force distribution.

Input File Usage: **DISTRIBUTING*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Distributing**: (**Translational Continuum distributing or Translational Structural distributing and Rotational Coupling Type: Rotational Continuum distributing or Rotational Structural distributing**)

Controlling Whether Cloud Node Rotations Influence Reference Node Rotation

By default, Abaqus couples reference node rotations to cloud node translations alone, unless all cloud nodes are nearly collinear and have rotational degrees of freedom. You can optionally control whether or not the cloud node rotations influence the reference node rotation. However, these options are subject to the following conditions:

- If the cloud nodes are collinear, Abaqus cannot couple the reference node rotation about the axis of collinearity of the cloud system to the cloud node translations alone. When this situation arises, Abaqus issues a warning message that identifies the axis about which the reference node rotation is not coupled to the cloud system. A consequence of the above is that a component of moment about the axis of collinearity acting at the reference node will not be transmitted to the cloud system.
- If the cloud nodes are not nearly collinear and you force cloud node rotations to influence reference node rotation, the actual influence of cloud node rotations on the reference node rotation is typically insignificant. Some performance degradation might occur because of the appearance of additional terms in the coupling equations.

Input File Usage: Use the following option (default) to allow Abaqus to choose if the cloud nodes rotations should be included in the rotational constraint:

**DISTRIBUTING*, ROTATIONAL COUPLING=AUTOMATIC(default)

Use the following option to neglect the cloud node rotations from the rotational constraint:

**DISTRIBUTING*, ROTATIONAL COUPLING=CONTINUUM

Use the following option to include the cloud node rotations in the rotational constraint:

**DISTRIBUTING*, ROTATIONAL COUPLING=STRUCTURAL

Abaqus/CAE Usage: A nondefault option for Abaqus/Standard to neglect cloud rotations in a rotational constraint for distributing coupling is not supported in Abaqus/CAE.

Optional Offset Associated with Bending

Cloud node rotations can optionally be considered in translational coupling constraints such that the reference node remains near a shell surface as it bends. For this coupling method to be active, all rotation degrees of freedom at all coupling nodes must be active (as is the case when the constraint is applied to a shell surface) and the constraints must be specified in all degrees of freedom (the default). In addition, for the constraint to be meaningful, the local (or global) z-axis used in the constraint should be such that it is parallel to the average normal direction of the constrained surface.

The translational constraint enforces a rigid beam connection between the reference node and a moving point that remains in the vicinity of the constrained surface at all times. The location of this moving point is determined by the approximate current curvature of the surface, the current location of the weighted center of the coupling nodes, and the z-axis used in the constraint.

Use of this option is independent of whether or not cloud node rotations influence rotational coupling constraints.

Input File Usage: **DISTRIBUTING*, COUPLING=STRUCTURAL

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Structural distributing**

Node-Based Surface

User-defined weight factors are used for node-based surfaces. The cross-sectional areas specified in the surface definition are used as the weight factors (see *Node-Based Surface Definition*). It is a best practice to define cloud surfaces as element-based surfaces, to ensure that representative surface areas are associated with each cloud node. If cloud surfaces are defined as node-based surface, care should be taken such that nodal area assignments for nodes of a node-based surface are of the correct order of magnitude. Using default nodal area assignment of 1.0 (regardless of length units for the mode) can lead to unit-dependence and undesired characteristics in the coupling of the reference node rotation to cloud system.

Releasing Components of the Rotational Coupling Constraint

You can optionally specify which degrees of freedom of the coupling constraint to constrain. Only rotational components can be released. All available translational degrees of freedom at the reference node are always coupled to the average translation of the coupling nodes and must be included in the degrees of freedom to constrain. One rotational constraint component can be released in a two-dimensional analysis. One, two, or three rotational constraint components can be released in a three-dimensional analysis. You can specify the rotational constraint directions in the global coordinate system or in a local coordinate system.

In a three-dimensional Abaqus/Standard analysis, if all three rotational constraints are released by constraining only degrees of freedom 1 through 3, only translation degrees of freedom are activated on the reference node. If only one or two rotation degrees of freedom are released, all three rotation degrees of freedom are activated at the reference node. In this case you must ensure that proper constraints have been placed on the unconstrained rotation degrees of freedom to avoid numerical singularities. Most often this is accomplished by using boundary conditions or by attaching the reference node to an element such as a beam or shell that provides rotational stiffness to the unconstrained rotational degrees of freedom.

In Abaqus/Explicit releasing one or more of the rotational constraints can lead to significant computational performance degradation. This is also the case when other constraints intersect the cloud of coupling nodes. In these cases, the degradation in performance is particularly noticeable when a large number of such distributed couplings are present in the model or when the size of the constrained “cloud” is large. Therefore, when the modeling conditions mentioned above are encountered, the size of the coupling nodes cloud is limited to 1000.

The following modeling technique can be used to alleviate rotational constraint issues: constrain all rotations in the distributed coupling and use an appropriate connector element at the reference node (such as REVOLUTE, HINGE, CARDAN, or BUSHING) to model released moments at the coupling's reference node. This technique also has the advantage of being able to specify finite compliance such as elasticity, plasticity, or damage in the “released” rotational component.

In geometrically nonlinear analysis steps the coordinate system of the degrees of freedom that define the rotational constraint release rotates with the reference node regardless of whether the global coordinate system or a local coordinate system is used.

Input File Usage: **DISTRIBUTING*
first dof, *last dof*

If no degrees of freedom are specified, all available degrees of freedom are coupled. If you specify one or more rotation degrees of freedom but not all available translation degrees of freedom, Abaqus issues a warning message and adds all available translation degrees of freedom to the constraint.

For example, the following coupling constraint is used to constrain degrees of freedom 1–5 on the reference node 1000 to the average translation and rotation of surface `surfA`:

*COUPLING, CONSTRAINT NAME=C1, REF NODE=1000, SURFACE=surfA
*DISTRIBUTING
1, 5

In this example the moment constraint between the reference node and the coupling nodes will be released in the 6-direction but will be enforced in the 4- and 5-directions. This provides a "revolute-like" rotation connection between the reference node and the coupling nodes (see *General Multi-Point Constraints*).

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Continuum distributing** or **Structural distributing**: toggle on the rotational degrees of freedom

Abaqus/CAE automatically constrains the translational degrees of freedom.

Node-Based Surface

User-defined weight factors are used for node-based surfaces. The cross-sectional areas specified in the surface definition are used as the weight factors (see *Node-Based Surface Definition*).

Element-Based Surface

For element-based surfaces the weight factors are calculated by Abaqus. The default weight distribution is based on the tributary surface area at each coupling node, except for a surface along a shell edge where the weight distribution is based on the tributary edge length. The procedure used to calculate the default weight factors is designed to ensure that if a radius of influence is prescribed, the default weight distribution varies smoothly with the influence radius.

Calculating the Default Distributing Weight Factors

The procedure to calculate the distributing weight factors depends on whether or not an influence radius is specified.

- If no influence radius is specified, the entire surface is used in the coupling definition. In this case all nodes located on the surface are included in the coupling definition and the distributing weight factor at each coupling node is equal to the tributary surface area.
- If an influence radius is specified, the default distributing weight factors at the coupling nodes are calculated as follows:

  1. A "participation factor" is calculated for each surface facet. The participation factor is defined below.
  2. The tributary nodal area (or tributary edge length along a shell edge) at each facet node is computed and is multiplied by the facet participation factor.
  3. The coupling node distributing weight factor is computed as the sum of the corresponding facet nodal areas (calculated above) for all joining facets.

Calculating the Facet Participation Factor

The participation factor defines the proportion of the facet's area that contributes to the distributing weight factors when an influence radius is specified. The participation factor varies between zero and one.

To define the participation factor, the distance of the facet node closest to the reference node, r_{min}, and the distance of the facet node farthest from the reference node, r_{max}, are calculated.

- If $r_{max} \leq r_{infl}$, where r_{infl} is the influence radius, all facet nodes lie within the influence region; and a participation factor of one is used.
- If $r_{min} \geq r_{infl}$, none of the facet nodes lie within the influence region; and the participation factor is set to zero.
- If $r_{max} > r_{infl}$, the facet is partially inscribed in the influence region; and the facet is assigned a participation factor equal to $(r_{infl} - r_{min}) / (r_{max} - r_{min})$.

If all coupling nodes fall outside the influence radius (that is, $r_{min} > r_{infl}$ for all facets), Abaqus selects all nodes belonging to the closest facets (as outlined under "Specifying a region of influence") and uses a participation factor equal to one.

Weighting Methods

You can modify the default weight distribution defined above. Various weighting methods are provided that monotonically decrease with radial distance from the reference node. For each case the default weight distribution that is based on the tributary surface area (or tributary edge length along a shell edge) is scaled by the weight factor w_i. If the weighting method is not specified, a uniform weighting method is used in which all weight factors are equal to 1.0.

Linearly Decreasing Weight Distribution

A linearly decreasing weighting scheme

$$w_i = 1 - \frac{r_i}{r_0},$$

where w_i is the weight factor at coupling node *i*, r_i is the coupling node radial distance from the reference node, and r_0 is the distance to the furthest coupling node.

Input File Usage: **i* *DISTRIBUTING*, WEIGHTING METHOD=LINEAR

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Continuum distributing** or **Structural distributing**: **Weighting method**: **Linear**

Quadratic Polynomial Weight Distribution

A quadratic polynomial weight distribution defined by

$$w_i = 1 - \left(\frac{r_i}{r_0}\right)^2.$$

Input File Usage: **i* *DISTRIBUTING*, WEIGHTING METHOD=QUADRATIC

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Continuum distributing** or **Structural distributing**: **Weighting method**: **Quadratic**

Monotonically Decreasing Weight Distribution

A monotonically decreasing weight distribution according to the cubic polynomial

$$w_i = 1 - 3\left(\frac{r_i}{r_0}\right)^2 + 2\left(\frac{r_i}{r_0}\right)^3.$$

Input File Usage: **DISTRIBUTING*, WEIGHTING METHOD=CUBIC

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Coupling type**: **Continuum distributing** or **Structural distributing**: **Weighting method**: **Cubic**

Processing of Unattached Nodes

Cloud nodes that have no stiffness cause numerical singularities in Abaqus/Standard analyses. You can guide Abaqus/Standard to provide proper management of such nodes. By default, Abaqus/Standard allows nodes that are not attached to any user elements. You can direct Abaqus/Standard to remove nodes that are not attached to any user elements or to issue an error message. You should keep unattached nodes if they derive their stiffness by being main nodes to other nodes that have stiffness.

Input File Usage: Use the following option to direct Abaqus/Standard to allow nodes that are not attached to any user elements (default):

**DISTRIBUTING*, FREE NODES=ALLOW

Use the following option to direct Abaqus/Standard to remove nodes that are not attached to any user elements:

**DISTRIBUTING*, FREE NODES=REMOVE

Use the following option to direct Abaqus/Standard to issue an error message:

**DISTRIBUTING*, FREE NODES=ERROR

Abaqus/CAE Usage: Directing Abaqus/Standard to remove or allow nodes that are not attached to any user elements is not supported in Abaqus/CAE.

Specifying a Local Coordinate System

The distributing coupling constraint can be specified with respect to a local coordinate system instead of the global coordinate system (see *Orientations*). *Figure 2* illustrates the use of a local coordinate system to release the moment constraints between the reference node and the coupling nodes in the local 4- and 6-directions, providing a "universal-like" rotation connection. In this example, a local rectangular coordinate system is defined that has its local *y*-axis coincident with the global *z*-axis. The moment constraint is specified in this local coordinate system.

Input File Usage: **COUPLING*, ORIENTATION=*local*

For example, the following input is used to specify the distributing coupling constraint shown in *Figure 2*:

```
*ORIENTATION, SYSTEM=RECTANGULAR, NAME=COUPLEAXIS
0.0, 1.0, 0.0, 0.0, 0.0, 1.0
*COUPLING, REF NODE=500, SURFACE=Endcap,
ORIENTATION=COUPLEAXIS
*DISTRIBUTING
```

1, 3
5, 5

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Coupling**: **Edit**: select local coordinate system

Converting a Distributing Coupling Attached to a Single Element Facet to a Tie Constraint

Some workflows involve fastening two surfaces together, with the stiffness of the connection represented by a hexahedral element (see *Auto convert single facet distributing coupling to tie* and *Figure 3*), where each node of the hexahedral element acts as a reference node of a distributing coupling whose cloud nodes are associated with a single shell element. This approach results in some shell nodes participating in two or four distributing couplings. Abaqus/Explicit generally introduces artificial mass to nodes acting as clouds nodes in multiple distributing couplings.

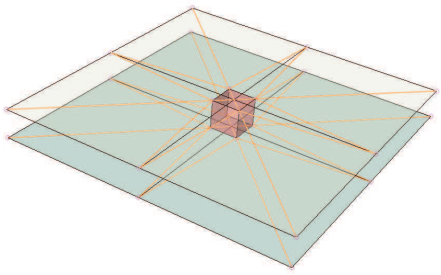


Figure 3: A fastener construct involving a hexahedral element representing the faster stiffness.

To reduce the likelihood of adding artificial mass to a model, Abaqus/Explicit converts a distributing coupling that spans only one facet of an element to a surface-based tie constraint by default unless the reference node is either:

- connected to a connector element
- offset from the element facet and nodes of the facet do not have rotational degrees of freedom

Input File Usage: Use the following option to direct Abaqus/Explicit to convert a distributing coupling attached to only one facet of the element to convert to a tie constraint:

**DISTRIBUTING*, SINGLE FACET=TIE CONVERT

To prevent Abaqus/Explicit from converting to a tie constraint, use the following option:

**DISTRIBUTING*, SINGLE FACET=DISTRIBUTING

Uniform Coupling Constraints

Uniform coupling constraints impose uniformity in temperature or other nonmechanical degrees of freedom among the coupled nodes. You can prescribe the value of the uniform field with a boundary condition at the reference node, or the evolving value of the uniform field can be determined by the simulation.

For a uniform coupling constraint, the reference node acts as the independent (or main) node and the surface nodes act as dependent (or secondary) nodes. A dependent degree of freedom of the uniform coupling constraint cannot also be constrained with a boundary condition or act as a secondary degree of freedom of another constraint in Abaqus/Standard.

If a reference node of a uniform coupling constraint is not involved in other couplings or elements, that node will have only the degree of freedom type associated with the uniform coupling constraint (such as temperature), even if surface nodes of the uniform coupling constraint have displacement degrees of freedom. For example, a force load cannot be applied to such a reference node.

It is common for the reference node of a uniform coupling constraint to act as a reference node of a kinematic or distributing coupling constraint as well. In such cases, the reference node has displacement and rotational degrees of freedom, along with whatever degrees of freedom are associated with the uniform coupling constraint.

Input File Usage: Use both of the following options to define a uniform coupling constraint (you can constrain only one degree of freedom for each use of the **UNIFORM* option, and you can constrain only nonmechanical degrees of freedom using this method):

```
*COUPLING
*UNIFORM
degree of freedom
```

Abaqus/CAE Usage: Uniform coupling constraint is not supported in Abaqus/CAE.

Limitations

- You cannot use a distributing coupling or kinematic coupling constraint with axisymmetric elements with asymmetric deformation. This element type is not compatible with coupling constraints.
- If you use a distributing coupling or kinematic coupling constraint with axisymmetric elements with twist, the constraint does not include the twist degree of freedom 5 in those elements. It involves the displacement degrees of freedom 1 and 2 only.
- If you use a distributing coupling or kinematic coupling with axisymmetric elements and the reference node lies on the axis of symmetry, you must constrain the displacement in the radial direction.

Shell-to-Solid Coupling

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Coupling Constraints*
- *About Surfaces*
- **SHELL TO SOLID COUPLING*
- *Defining shell-to-solid coupling constraints*

Overview

Surface-based shell-to-solid coupling connects a shell edge to a solid surface.

Surface-based shell-to-solid coupling:

- is most useful when local modeling should use a full three-dimensional analysis but other parts of the structure can be modeled as shells;
- uses a set of internally defined distributing coupling constraints to couple the motion of a “line” of nodes along the edge of a shell model to the motion of a set of nodes on a solid surface;
- automatically selects the coupling nodes located on a solid surface lying within a region of influence;
- can be used with three-dimensional stress/displacement shell and solid (continuum) elements;
- does not require any alignment between the solid and shell element meshes; and
- can be used in geometrically linear and nonlinear analysis.

Shell-to-Solid Coupling

Shell-to-solid coupling in Abaqus is a surface-based technique for coupling shell elements to solid elements. *Figure 1* illustrates two examples taken from *Shell-to-solid submodeling and shell-to-solid coupling of a pipe joint* and *The pinched cylinder problem*. Shell-to-solid coupling is intended to be used for mesh refinement studies where local modeling requires a relatively fine through-the-thickness solid mesh coupled to the edge of a shell mesh, as shown in *Figure 2*. In such a case Abaqus assembles constraints that couple the displacement and rotation of each shell node to the average displacement and rotation of the solid surface in the vicinity of the shell node.

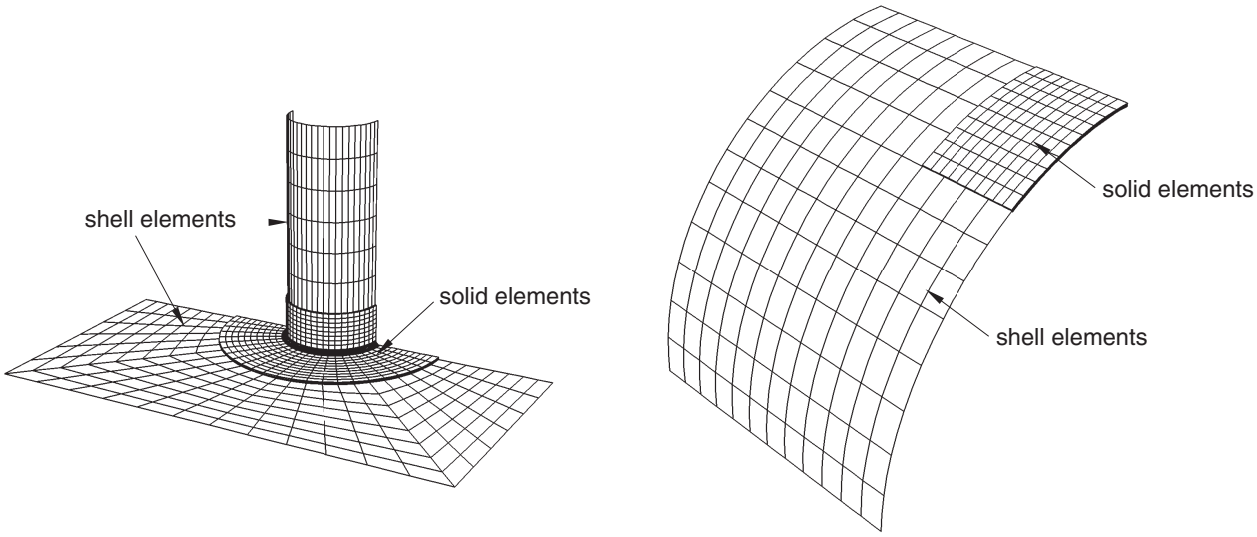


Figure 1: Typical examples of shell-to-solid coupling.

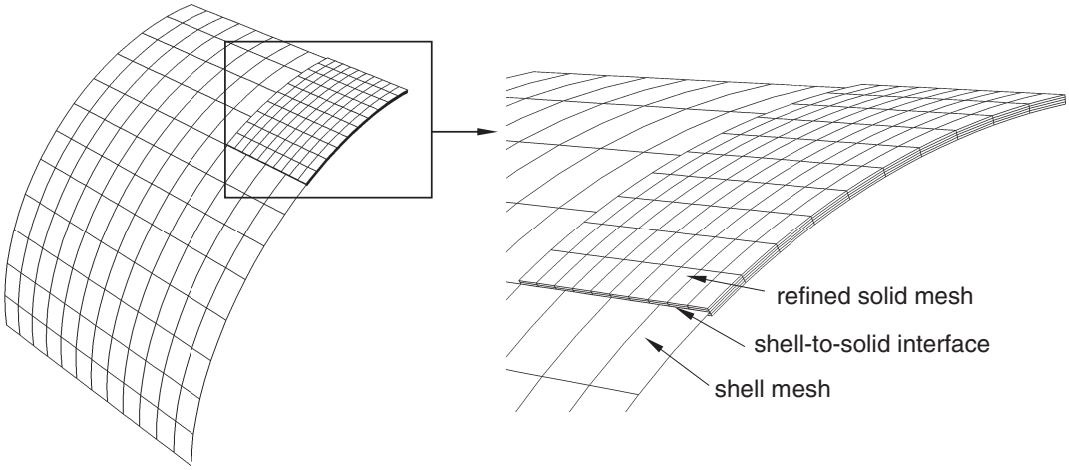


Figure 2: Shell-to-solid interface.

As shown in *Figure 2*, the coupling occurs along a shell-to-solid interface defined by two user-specified surfaces: an edge-based shell surface and an element- or node-based solid surface (see *About Surfaces*). The shell surface (*Figure 3*) is referred to as the “shell edge.”

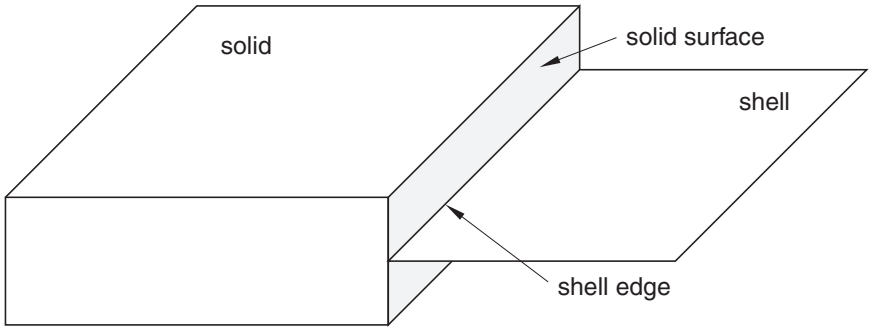


Figure 3: Shell and solid surfaces.

The shell element edges that define the edge-based shell surface are referred to as “edge facets.” The edge facets are either linear or parabolic segments depending if the underlying shell elements are linear or quadratic.

The shell-to-solid coupling is enforced by the automatic creation of an internal set of distributing coupling constraints (see *Coupling Constraints*) between nodes on the shell edge and nodes on the solid surface. Abaqus uses default or user-defined distance and tolerance parameters (discussed below) to determine which nodes on the shell edge will be coupled to which nodes on the solid surface. For each shell node involved in the coupling, a distinct internal distributing coupling constraint is created with the shell node acting as the reference node and the associated solid nodes acting as the coupling nodes. Each internal constraint distributes the forces and moments acting at its shell node as forces acting on the related set of coupling surface nodes in a self-equilibrating manner. The resulting line of constraints enforces the shell-to-solid coupling.

Defining Shell-to-Solid Coupling

Defining a shell-to-solid coupling constraint requires the specification of a constraint name, an edge-based shell surface, and an element- or node-based solid surface.

Abaqus automatically determines which nodes on the two surfaces participate in the coupling and creates appropriate internal distributed coupling constraints. You can also control which nodes on the two surfaces participate in the coupling by specifying a position tolerance and influence distance as described below.

The resulting coupling constraint definitions are printed to the data file when model definition data are requested (see *Controlling the Amount of analysis input file processor Information Written to the Data File*). Abaqus also creates an internal node set that contains all the solid nodes included in the coupling; the node set can be visualized using the Visualization module of Abaqus/CAE. The name of the internal node set is the name assigned to the coupling constraint.

Input File Usage: **SHELL TO SOLID COUPLING*, CONSTRAINT NAME=*name*
shell_surface, *solid_surface*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Shell-to-solid coupling**

Controlling the Shell Nodes Included in the Coupling

A *position tolerance* determines the absolute distance from the solid surface within which all shell nodes to be included in the coupling must lie. Shell nodes that lie outside this tolerance are not coupled to the solid surface. To determine the candidate nodes on the solid surface, the shell edge-based surface is first projected on the solid surface. If the shell edge-based surface cannot be projected on the solid surface, no couplings are formed.

When using an element-based solid surface, the defined distance between a shell node and the solid surface is the projected distance measured along a line extending from the shell node to the closest point on the solid surface

(which might be on the edge of the solid surface). The default position tolerance when using an element-based solid surface is 5% of the length of a typical facet on the shell edge.

For a node-based solid surface the defined distance of a shell node to the surface is the distance to the closest node on the solid surface. The default position tolerance when using a node-based solid surface is based on the average distance between nodes on the solid surface.

You can specify a nondefault position tolerance for element- or node-based solid surfaces.

Input File Usage: **SHELL TO SOLID COUPLING*, POSITION TOLERANCE=*distance*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Shell-to-solid coupling**: select the surfaces: choose **Specify distance** for the **Position Tolerance**

Controlling the Solid Nodes Included in the Coupling

A geometric tolerance, which is referred to as the *influence distance*, is defined for each edge facet. For a given node or element facet on the solid surface to be included in the coupling constraint, its perpendicular distance from at least one edge facet must be less than or equal to the influence distance defined for that edge facet. The default influence distance for an edge facet is half the thickness of the underlying shell element. The default automatically accounts for any offset or nodal thickness included with the shell element's cross-section definition. You can specify a nondefault influence distance.

A user-defined influence distance is optional in all cases except when an edge facet involved in the coupling is associated with a general arbitrary elastic shell section definition in which you specified the general stiffness. In this case since the shell thickness is not defined directly, you must supply an influence distance.

Input File Usage: **SHELL TO SOLID COUPLING*, INFLUENCE DISTANCE=*distance*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Shell-to-solid coupling**: select the surfaces: choose **Specify value** for the **Influence Distance**

Computation of the Internal Coupling Constraints

This section outlines the basic procedure used by Abaqus to compute the internal shell-to-solid coupling constraints.

A single distinct internal distributing coupling constraint is created for each shell node that lies within the position tolerance from the solid surface. Internal coupling constraints are not created for shell nodes that lie outside this tolerance. The shell node acts as the reference node, and a set of nodes on the solid surface act as the coupling nodes. Abaqus finds the coupling nodes on the solid surface and computes the weight factors for the internal constraints by considering each shell edge facet separately. The following procedure is carried out for each edge facet:

1. Abaqus finds all nodes on the solid element surface that lie within the region of influence (discussed below) of the current edge facet. These nodes are included in the coupling constraint.
2. Abaqus then computes a set of weight factors for the solid nodes. A weight factor is a measure of both the tributary area of the solid node contained within the region of influence and the relative position of the solid node with respect to each shell node. The tributary areas for node-based surfaces are the cross-sectional areas that you specified when you defined the surface. For element-based surfaces, the tributary areas are calculated by Abaqus. The sum of all the weight factors in each coupling constraint is a measure of the total tributary area of the solid surface that is contained within the region of influence.

3. The above procedure is carried out for all the shell edge facets contained within the shell surface. If a shell node belongs to more than one edge facet, all the coupling nodes and weight factors are combined into a single distributing constraint definition. The resulting line of constraints along the shell edge enforces the shell-to-solid coupling.

There are two situations in which a shell node might satisfy the position tolerance but no coupling constraint is defined. If a shell node lies within the position tolerance but is not connected by an edge facet to at least one other shell node that also satisfies the tolerance, a coupling constraint is not created for this shell node. In this case it might be necessary to increase the position tolerance. Alternatively, if nonzero weight factors are not computed for at least two solid nodes associated with the shell node, a coupling constraint is not created for this shell node. The most likely cause for zero weight factors is that the influence distance is too small. In the case of a node-based surface, zero weights might also arise if the default cross-sectional area is used. For shell-to-solid coupling the default area is zero.

The Region of Influence for an Edge Facet

The region of influence of an edge facet is defined by a cylindrical volume whose centerline is the edge facet and whose radius is the edge facet's influence distance. The ends of the cylindrical volume are defined by two bounding planes whose normals are the shell tangents at the two ends of the edge facet (see *Figure 4*).

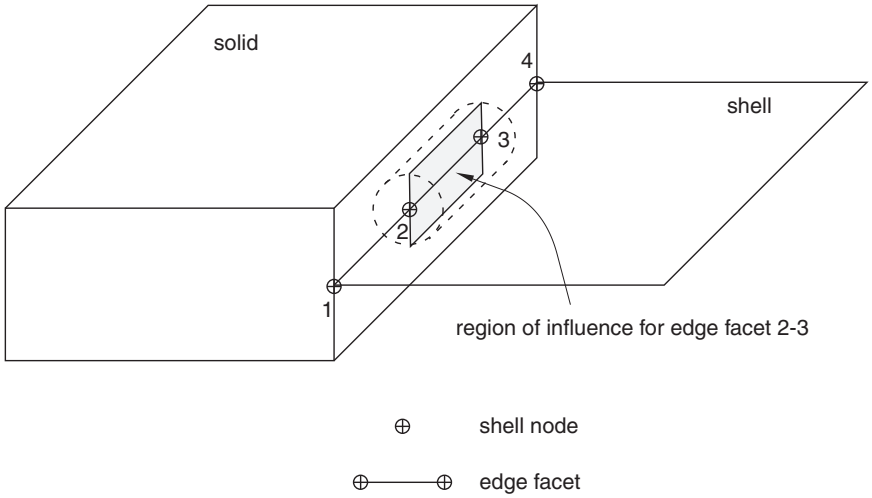


Figure 4: Regions of influence for an edge facet.

In this example a region of influence is constructed for shell edge 2–3. For a node-based solid surface, only the nodes that lie within or on the boundary of the region of influence are assigned to the current edge facet and included in the coupling definition. For an element-based solid surface, each solid facet node is associated with part of the facet surface. If the part of the facet assigned to a given solid node falls within the region of influence, that node is included in the coupling definition.

Using the Normal on an Element-Based Solid Surface to Restrict Solid Nodes That Are Used in the Coupling

In the case of an element-based solid surface, Abaqus compares the normal of each solid facet within the region of influence to the normal of the solid surface closest to the centerline of the cylindrical volume (see *Figure 4*). In general, if the normal of a surface facet is not within 20° of the normal at the centerline, the nodes on the solid surface facet are not included in the coupling definition. For the case illustrated in *Figure 4* this check would prevent nodes on the top and bottom surface of the solid mesh from being coupled to the shell nodes even if the influence distance was arbitrarily large and the solid surface definition included all sides of the solid geometry.

This check is not used if the centerline is on or near a feature edge of the solid mesh where the normal is not well defined (see the discussion about shell offsets below).

Comments, Restrictions, and Modeling Recommendations for Shell-to-Solid Coupling

- The shell-to-solid coupling formulation assumes that the interface surface between the shell and solid elements is normal to the shell. Therefore, while the solid surface can be curved in a direction tangent to the shell edge, it should be straight in the direction along the shell normals. This is an assumption on the geometry of the surfaces, not on the mesh. It is not necessary for the nodes on the solid surface to line up with each other or to line up with the shell nodes.
- The shell-to-solid coupling capability is designed for analyses where the solid mesh is fine with respect to the shell thickness. It is recommended that at least two solid elements be included through the thickness at a shell-to-solid interface. Along the shell-to-solid interface the length of a shell edge facet should in general be of the same order as the characteristic surface dimension of a solid element facet.
- An assumption used in the design of the shell-to-solid coupling algorithms is that the weight factors are based on accurate nodal tributary areas, such as those automatically computed by Abaqus when an element-based surface is used. Therefore, it is generally recommended that an element-based solid surface be used instead of a node-based solid surface. However, in cases where the shell and solid meshes align with each other, it is sometimes advantageous to use a node-based solid surface especially when a homogenous solution is expected.
- *Figure 5* illustrates some recommended modeling practices for shell-to-solid coupling. If the shell reference surface is not offset, the shell edge should be centrally located with respect to the thickness direction of the solid (*Figure 5*(a)). The solid surface should include only the portion needed for the coupling (the shaded region shown in *Figure 5*(a)).

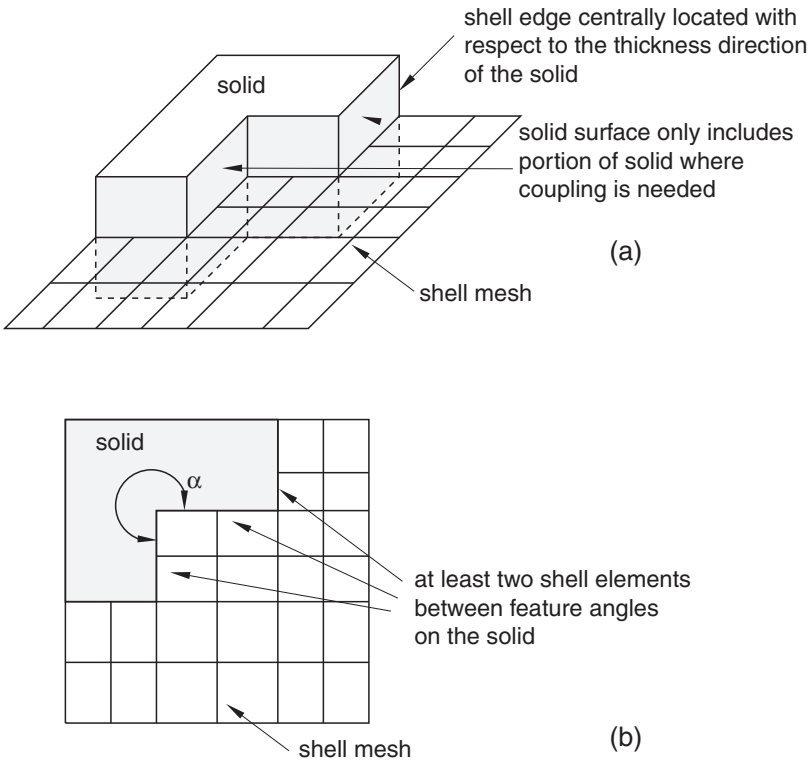


Figure 5: Modeling recommendations for the shell-to-solid interface.

- The shell-to-solid interface can be defined around geometric feature angles (corners), α (*Figure 5*(b)). However, it is recommended that the feature angles satisfy $60° < \alpha < 300°$. In addition, as illustrated in *Figure 5*(b), at least two shell element edges should be included between each feature angle.
- If an offset is defined for the shell section and the reference shell edge is placed at or near a feature edge on the solid surface (*Figure 6*), the solid surface should include only the side of the solid that you want to be included in the coupling definition.

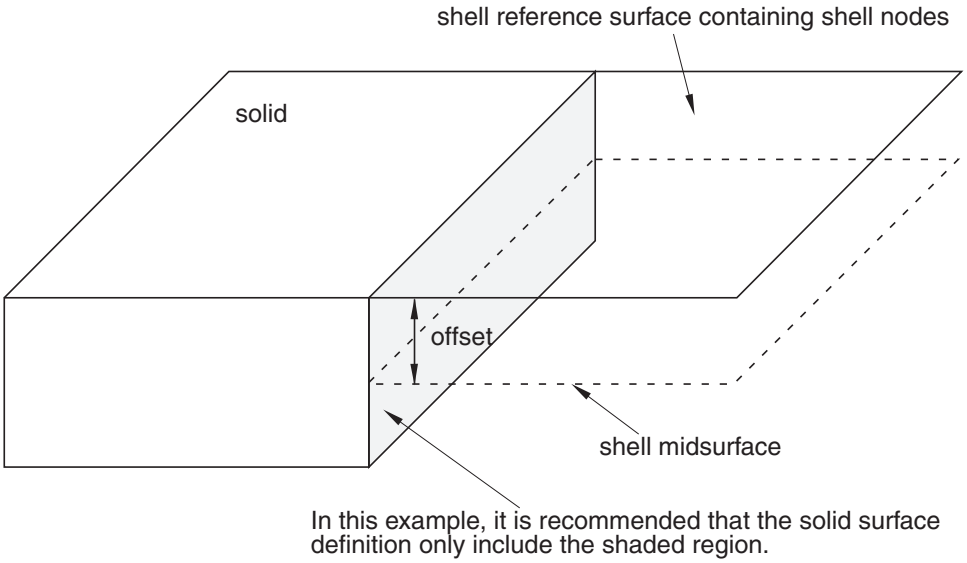


Figure 6: Modeling recommendations for the shell-to-solid interface with a shell offset.

  For example, if the top of the solid in *Figure 6* is included in the surface definition, Abaqus includes nodes on the top of the surface in the coupling constraint, which is not what you intended. You intended only that the shell be coupled to the shaded region of the solid in *Figure 6*. Therefore, the solid surface definition should include only this region.
- Care must be taken in interpreting the local stress and strain fields in the immediate vicinity of the shell-to-solid interface. This is especially true if the shell-to-solid interface includes corners or edges. The interface should be placed at least a distance more than the shell thickness away from the region in the solid mesh where the stress and strain fields are of interest.
- The shell-to-solid interface should be located in a region of the model where shell theory is a valid modeling approximation.
- Corners or kinks might exist in models made of shell elements. At such corners or kinks the shell elements only approximate the distribution of the material away from the midsurface of the shell. While the global moments and forces between the shell and solid models are transferred correctly, the local stress and displacement fields in the region of the shell-to-solid interface might be inaccurate.
- Only displacement degrees of freedom in the solid elements and displacement and rotation degrees of freedom in the shell elements are coupled in shell-to-solid coupling. Shell-to-solid coupling does not couple other degrees of freedom such as temperature and pressure.
- Shell-to-solid coupling can be used to couple three-dimensional shells to all three-dimensional continuum elements except cylindrical elements (*Cylindrical Solid Element Library*).

Mesh-Independent Fasteners

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surfaces*
- *Coupling Constraints*
- *Connector Elements*
- **FASTENER*
- **FASTENER PROPERTY*
- *About fasteners*

Overview

The mesh-independent fastener capability is a convenient method to define a localized connection between two or more surfaces such as a spot weld or rivet connection.

In addition, the mesh-independent fastener capability:

- uses spatial coordinates of fastener locations to define point-to-point connections independent of underlying meshes;
- combines either connector elements or BEAM MPCs with distributing coupling constraints to provide a connection that can be located anywhere between two or more surfaces regardless of the mesh refinement or location of nodes on each surface;
- can be used to connect both deformable and rigid element-based surfaces;
- can model either rigid, elastic, or inelastic connections with failure by using the generality of connector behavior definitions; and
- is available only in three dimensions.

Introduction

Many applications require modeling of point-to-point connections between parts. These connections might be in the form of spot welds, rivets, screws, bolts, or other types of fastening mechanisms. There might be hundreds or even thousands of these connections in a large system model such as an automobile or airframe.

The fastener can be located anywhere between the parts that are to be connected regardless of the mesh. In other words, the location of the fastener can be independent of the location of the nodes on the surfaces to be connected. Instead, the attachment to each of the parts being connected is distributed to several nodes in the surfaces to be connected in the neighborhood of the fastening points. *Figure 1* shows a typical one-layer and two-layer fastener configuration.

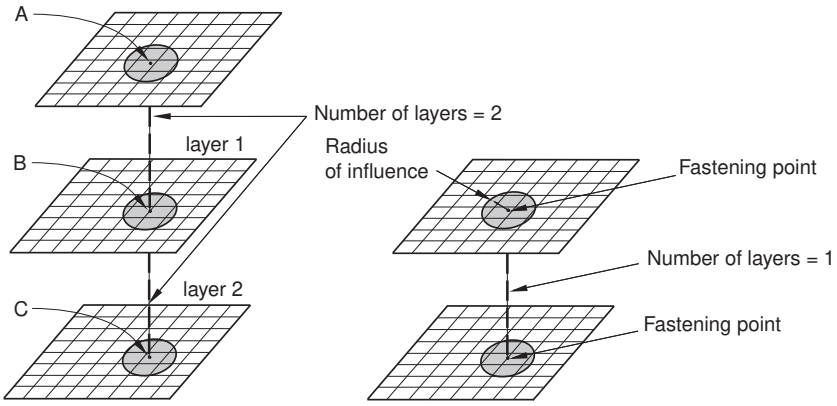


Figure 1: Typical one-layer and two-layer fastener configuration.

Each layer connects two fastening points using either a connector element or a BEAMMPC. Each fastening point is connected to the surface using a distributing coupling constraint that couples the displacement and rotation of each fastening point to the average displacement and rotation of the nearby nodes.

The mesh-independent fastener capability in Abaqus is designed to model these connections in a convenient manner. The fastener automatically:

- determines the locations of nodes and orientations of connector elements or BEAM MPCs between two or more surfaces;
- generates distributing coupling constraints to attach the connector elements or BEAM MPCs to each surface in a mesh-independent manner; and
- calculates weights for the distributing coupling constraints that complete the mesh-independent connection.

For an example of the use of mesh-independent fasteners, see *Buckling of a column with spot welds*. Mesh-independent fasteners are referred to as point-based fasteners by Abaqus/CAE. For more information, see *About fasteners*. It is also possible to assemble fasteners in Abaqus/CAE using connector elements, coupling constraints, etc. For further details, see *About assembled fasteners*.

Fastener Interactions

Fasteners are defined in groups called interactions, which are assigned names. Each interaction defines one or more fasteners. The number of individual fasteners is equal to the number of positioning points used to locate the fasteners. Fastening points on each surface are found by considering the position of the positioning point as discussed in subsequent sections.

Fasteners can be defined using connector elements or BEAMMPCs. BEAMMPCs allow modeling of perfectly rigid connectors between components; while connector elements allow you to model much more complex behavior, such as deformable connectors that include the effects of elasticity, damage, plasticity, and friction.

Input File Usage: **FASTENER*, INTERACTION NAME=*name*

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Name**: *name*,
Type: **Point-based**

Defining Fasteners Using BEAMMPCS

For modeling perfectly rigid connections you need not define fasteners using connector elements. Instead, Abaqus can internally generate BEAMMPCs connecting the fastening points of the fasteners. In this approach you assign a reference node set containing a list of user-defined nodes to the fastener interaction. The nodes in this reference

node set will be used as positioning points to locate the fasteners. If single-layer fasteners are to be modeled, Abaqus generates single BEAMMPCs with each node in the reference node set becoming the first node of the BEAMMPC. The second node of each BEAMMPC will be generated internally by Abaqus. If multi-layer fasteners are to be defined, Abaqus generates linked sets of BEAMMPCs with each node in the reference node set becoming the first node of the first BEAMMPC in each linked set. The subsequent nodes in each linked set will be generated internally by Abaqus. For multi-layer fasteners each linked set contains as many BEAMMPCs as the number of layers in the fastener.

Input File Usage: Use the following options:

**FASTENER*, INTERACTION NAME=*name*,
REFERENCE NODE SET=*node set label*
**NSET*, NSET=*node set label*

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Property**: **Section**: **Rigid MPC**

Defining Fasteners Using Connector Elements

Using connector elements as the basis for a point-to-point connection allows for very complex behavior to be modeled with fasteners. Like other uses of connector elements, the connection can be fully rigid or might allow for unconstrained relative motion in local connector components. In addition, deformable behavior can be specified using a connector behavior definition that can include the effects of elasticity, damping, plasticity, damage, and friction. There are two methods to define fasteners that use connector elements to model the behavior between fastening points. For both methods the fastener interaction refers to an element set containing the connector elements. You must specify a connector section definition that refers to this element set. You should be careful when specifying the connector orientation (if needed) as discussed below in *Defining the Fastener Orientation*.

Defining the Connector Elements Directly

The most controlled approach to specifying fasteners using connector elements is to define the connector elements explicitly and associate them with an element set. The fastener interaction refers to the element set. Each fastener in the fastener interaction corresponds to one or more connector elements depending on the number of layers of the fastener (see *Figure 2*).

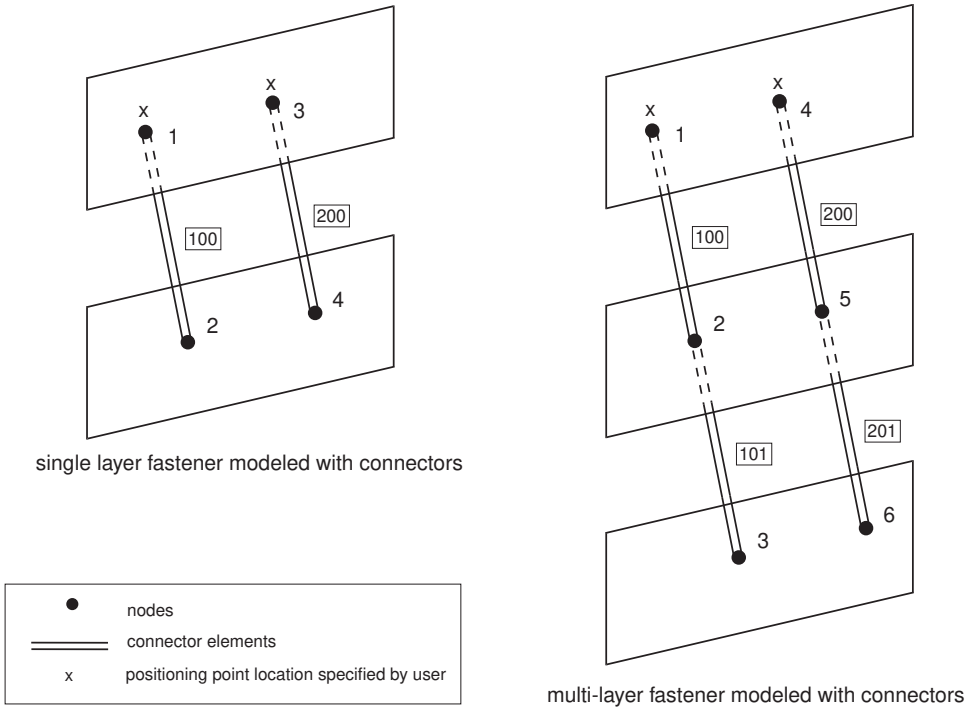


Figure 2: Single- and multi-layer fasteners modeled with connector elements.

A single connector element is associated with each layer, and the two nodes of the connector element correspond to the fastening points of the two adjacent surfaces. When specifying a multi-layer fastener, the connector elements for each layer should share nodes with the connector elements of adjacent layers.

For a single-layer fastener the positioning point used to locate the fastener and its fastening points is taken as the nodal coordinates of the first node of the connector element. For a multi-layer fastener the positioning point is taken as the first node of the first connector in a linked set of connectors with as many members as layers. Examples of defining a single-layer and multi-layer fastener are included at the end of this section.

Input File Usage: Use the following options:

**FASTENER*, INTERACTION NAME=*name*, ELSET=*element set label*
blank line
**ELEMENT*, TYPE=CONN3D2, ELSET=*element set label*
**CONNECTOR SECTION*, ELSET=*element set label*

Abaqus/CAE Usage: For point-based fasteners in Abaqus/CAE, you cannot define the connector elements directly; the connector elements are generated by Abaqus.

Connector Elements Generated by Abaqus

In this approach you do not need to explicitly define the connector elements that connect the fastening points of the fastener. The fastener interaction refers to an empty element set. You must specify a connector section definition that refers to this element set. In addition, you assign a reference node set containing a list of user-defined nodes to the fastener interaction. The nodes in this reference node set are used as positioning points to locate the fasteners.

If single-layer fasteners are to be modeled, Abaqus generates single connector elements with each node in the reference node set becoming the first node of a connector element. The second node of each connector element

will be generated internally by Abaqus. If multi-layer fasteners are to be defined, Abaqus generates linked sets of connector elements with each node in the reference node set becoming the first node of the first connector element in each linked set. The subsequent nodes in each linked set will be generated internally by Abaqus. For multi-layer fasteners each linked set contains as many connector elements as the number of layers in the fastener. The connector elements are given internally generated element numbers and assigned to the named user-specified element set. You can use this element set to request output for these connector elements. However, this element set should not be included in another element set definition.

Input File Usage: Use the following options:

**FASTENER*, INTERACTION NAME=*name*, ELSET=*element set label*,
REFERENCE NODE SET=*node set label*
blank line
**NSET*, NSET=*node set label*
**CONNECTOR SECTION*, ELSET=*element set label*

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Property**: **Section**: **Connector section**: select connector section

Example: Using Connector Elements to Define Single-Layer Fasteners Directly

To define a single-layer fastener directly using connector elements:

- Define two connector elements with user element numbers 100 and 200 and user-defined node numbers 1, 2 and 3, 4, respectively, and include them in an element set. Nodes 1 and 3 act as the positioning points for the two fasteners (see *Figure 2*).
- Refer to the element set in the fastener interaction and connector section definitions.
- Assign section properties to the fasteners. Suppose in this example that relative displacements between the fastening points are to be allowed. Therefore, the fasteners must be assigned a section that has available components of motion; for example, a CARTESIAN section can be used.
- The relative displacement between the fastening points gives rise to elastic deformations. Hence, the material between the fasteners is modeled as linear elastic with a spring stiffness of 10000 using connector elasticity.

The following input can be used:

```
*FASTENER, INTERACTION NAME=fastinter, ELSET=fastconn, PROPERTY=fastprop
blank line
surface1, surface2
*ELEMENT, TYPE=CONN3D2, ELSET=fastconn
100, 1, 2
200, 3, 4
*CONNECTOR SECTION, ELSET=fastconn, BEHAVIOR=behav
CARTESIAN,
*CONNECTOR BEHAVIOR, NAME=behav
*CONNECTOR ELASTICITY, COMPONENT=1
10000,
*CONNECTOR ELASTICITY, COMPONENT=2
10000,
*CONNECTOR ELASTICITY, COMPONENT=3
10000,
```

Example: Using Connector Elements to Define Multi-Layer Fasteners Directly

To define a multi-layer fastener directly using connector elements:

- Define two linked sets of connector elements with each linked set containing exactly two connectors. The first linked set comprises element numbers 100 and 101, with node numbers 1, 2 and 2, 3, respectively. The second linked set comprises element numbers 200 and 201, with node numbers 4, 5 and 5, 6, respectively. Include the connector elements in an element set. Nodes 1 and 4 act as the positioning points for the two fasteners (see *Figure 2*).
- Refer to the element set in the fastener interaction and connector section definitions
- Assign section properties to the fasteners. Suppose in this example that rigid beam-type behavior between the fastening points is to be modeled; in that case the fasteners must be assigned a BEAM section.

The following input can be used:

```
*FASTENER, INTERACTION NAME=fastinter, ELSET=fastconn, PROPERTY=fastprop
blank line
surface1, surface2, surface3
*ELEMENT, TYPE=CONN3D2, ELSET=fastconn
100, 1, 2
101, 2, 3
200, 4, 5
201, 5, 6
*CONNECTOR SECTION, ELSET=fastconn
BEAM,
```

Specifying the Positioning Points, Projection Method, and Fastening Points

Each interaction defines one or more fasteners. The number of individual fasteners is equal to the number of positioning points used to locate the fasteners. Positioning points are nodes defined at the fastener locations and assigned as a reference node set to the interaction.

In general, a positioning point should be located as close to the surfaces being connected as possible. The reference node specifying the positioning point can be one of the nodes on the connected surfaces or can be defined separately. Abaqus determines the actual points where the fastener layers attach to the surfaces that are being connected by first projecting the positioning point onto the closest surface. Abaqus offers the following projection methods to find fastening points on the specified surfaces to form fasteners:

- Face-to-face
- Face-to-edge
- Edge-to-face
- Edge-to-edge
- Connector direction

The choice of method depends on how the surfaces are oriented relative to each other.

Fastening Surfaces That Are Nearly Parallel to Each Other

Most commonly the surfaces to be fastened together are nearly parallel to each other; in which case the fastening points are located on element facets away from the periphery of the surfaces. The face-to-face projection method is most appropriate for such situations. It is also the default projection method.

In the face-to-face projection method, Abaqus projects each positioning point onto the closest surface along a directed line segment normal to the surface. Alternatively, you can specify the projection direction. Specifying the direction might be useful when two-dimensional drawings are used to identify the positioning point locations and those locations are known precisely in two dimensions but not in a third. For this case the direction specified is typically the normal to the plane of the drawing.

Once the fastening point on the closest surface has been identified, Abaqus determines the points on the other surface or surfaces to be connected by projecting the first fastening point onto the other surfaces along the fastener normal direction, which is typically normal to the closest surface. *Figure 3* shows the two ways of locating the projection points. When surfaces to be fastened are not exactly parallel, Abaqus sometimes sets attachment points to be at the closest facet edges or corner on the surface, rather than along the fastener normal direction.

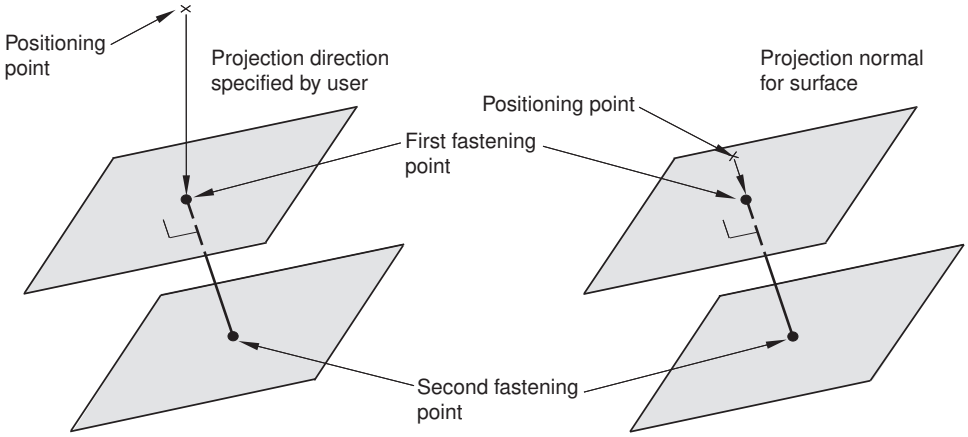


Figure 3: Directed and normal projection to locate the fastening points for the face-to-face projection method.

The location of the positioning point (a node in the reference node set) might not coincide with the locations of the fastening points found by Abaqus. Hence, the coordinates of the node at the positioning point might change from their user-prescribed values when the node is shifted to a fastening point. If the node at the positioning point is part of the connectivity of a user-defined element, this can cause the element whose connectivity includes that node to undergo unacceptable initial distortions. In such situations it is recommended that you define the node at the positioning point separately. In general, you should not specify this node to be one of the nodes of the connected surfaces.

Input File Usage: Use the following option to allow Abaqus to define the projection direction:

**FASTENER*, REFERENCE NODE SET=*node set label*, ATTACHMENT METHOD=FACETOFACE (default)
blank line

Use the following option to define the projection direction directly:

**FASTENER*, REFERENCE NODE SET=*node set label*, ATTACHMENT METHOD=FACETOFACE (default)
x-component, y-component, z-component

Abaqus/CAE Usage: Use the following input to allow Abaqus to define the projection direction:

Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Domain** tabbed page: **Direction vector**: **Default**, **Criteria** tabbed page: **Attachment method**: **Face-to-Face**

Use the following input to define the projection direction directly:

Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Domain** tabbed page: **Direction vector**: **Specify**, **Criteria** tabbed page: **Attachment method**: **Face-to-Face**

Fastening Nearly Perpendicular Surfaces

When you need to fasten surfaces that are perpendicular or nearly perpendicular to each other; that is, forming a T-intersection, the face-to-edge or the edge-to-face projection methods are appropriate choices. *Figure 4* shows attachments for the face-to-edge and edge-to-face projection methods.

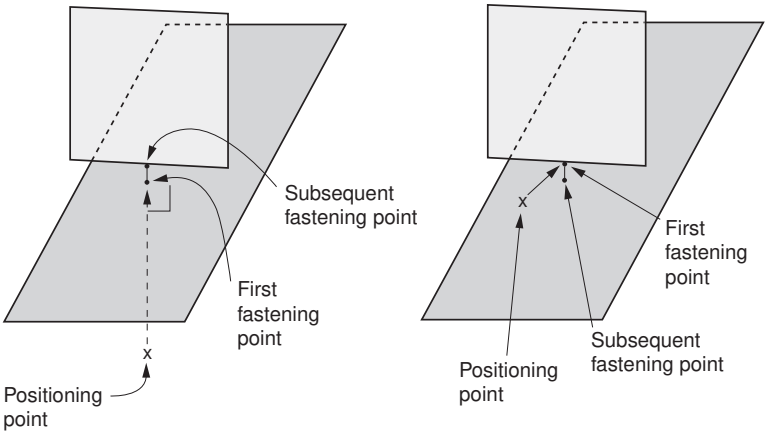


Figure 4: Face-to-edge and edge-to-face projection methods to locate fastening points for surfaces that form T-intersections.

Creating the First Fastening Point on a Face

In the face-to-edge projection method Abaqus projects the positioning point onto the closest surface along a directed line segment normal to the surface. The subsequent fastening points are found by searching for the closest points on the remaining specified surfaces. The closest fastening point might fall on the edge or a corner of a surface.

Input File Usage: **FASTENER*, REFERENCE NODE SET=*node set label*, ATTACHMENT METHOD=FACETOEDGE
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Criteria**: **Attachment method**: **Face-to-Edge**

Creating the First Fastening Point on an Edge

In the edge-to-face projection method, the first fastening point is found by searching for the closest point on the specified surface or surfaces. The closest point might be on the edge or corner of the surface. For subsequent fastening points Abaqus projects the previous fastening point along a directed line segment normal to the surface.

Input File Usage: **FASTENER*, REFERENCE NODE SET=*node set label*, ATTACHMENT METHOD=EDGETOFACE
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Criteria**: **Attachment method**: **Edge-to-Face**

Fastening Abutting Surfaces

When it is desired to form fasteners between surfaces that are butting against each other, the edge-to-edge projection method is appropriate. In this method the first as well as the subsequent fastening points are located by searching for the closest point on the specified surface or surfaces. The fastening points in this method might be located on the edge of a surface. *Figure 5* shows attachments for the edge-to-edge projection method.

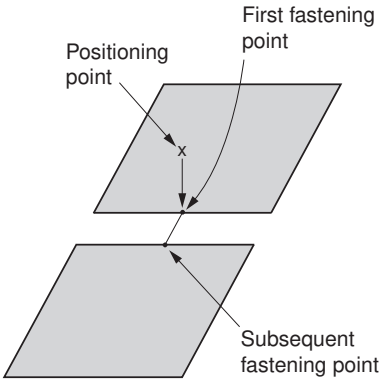


Figure 5: Edge-to-edge projection method to locate fastening points for abutting surfaces.

Input File Usage: **FASTENER*, REFERENCE NODE SET=*node set label*, ATTACHMENT METHOD=EDGETOEDGE
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: select positioning points: **Criteria**: **Attachment method**: **Edge-to-Edge**

Fastening Surfaces That Are Not Parallel

When fastening surfaces that are not parallel to one another, you can control the precise location and direction of the fastener. To define the location and direction, prescribe a connector element for each fastener with nodes at a specific location. Abaqus maintains the location and the direction of the connector element.

Input File Usage: **FASTENER*, ELSET=*element set label*, ATTACHMENT METHOD=CONNECTORDIRECTION
blank line

Abaqus/CAE Usage: Selecting a connector to control the location and direction of the fastener is not supported in Abaqus/CAE.

Specifying the Surfaces to Be Fastened

Once the positioning points have been specified, the surfaces to be fastened can be specified using two different approaches. In the first approach you directly specify the surfaces that are to be connected with a fastener. In the second approach you specify a search zone, and Abaqus automatically identifies the surfaces that are to be connected. However, in the second approach Abaqus does not distinguish between coincident facets. Hence, if coincident facets are to be fastened, you should specify distinct surfaces containing each of the coincident facets and use the first approach. Only element-based surfaces defined on faces can be fastened together (see *Element-Based Surface Definition* and *Operating on Surfaces*).

Forming Fasteners on User-Specified Surfaces

If you specify multiple surfaces as part of the interaction definition, the surfaces to be fastened are restricted to these surfaces. In general, specifying multiple surfaces is the preferred way of defining fasteners; this method leads to a more precise fastener construct definition. The number of layers of each fastener is one less than the number of surfaces specified. One fastening point is found on each surface.

Input File Usage: **FASTENER*
first data line
surface1, surface2, surface3, etc.

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Domain**: **Approach**: **Fasten specified surfaces by proximity**, select surfaces

When you select multiple surfaces for a single surface region, Abaqus/CAE combines the multiple surfaces using the single-surface search method, as described in *Forming Fasteners on Surfaces inside a User-Specified Search Zone* below.

Controlling Connectivity of Fasteners on User-Specified Surfaces

By default, the connectivity of the fastening points is determined by their relative position along the fastener projection direction. For example, the default connectivity for the two-layer example shown in *Figure 1* connects fastening point A to point B (layer 1) and point B to point C (layer 2).

You can control the connectivity of the fastening points when the fasteners are formed on user-specified surfaces. You can specify that the connectivity of the fastening points be defined by the order in which you specified their associated surfaces.

Input File Usage: **FASTENER*, UNSORTED
first data line
surface1, surface2, surface3, etc.

If user-specified surfaces are not included on the data lines, the UNSORTED parameter is ignored.

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Domain**: **Approach**: **Fasten in specified order**, select surfaces

Forming Fasteners on Surfaces inside a User-Specified Search Zone

If you do not specify any surfaces as part of the interaction definition, Abaqus searches for fastening points on all element facets that fall within a sphere of user-specified radius R with its center at the positioning point. If you do not specify the search radius, Abaqus computes a default search radius based on five times the facet thickness (for shell element facets) or the characteristic element length (for other element types) in the vicinity of each positioning point.

To refine the search, you can specify a single surface definition that will limit the facet search to element facets belonging to that surface. In this case you must define a collective surface that includes at least each connected surface. A combined surface can also be used (see *Operating on Surfaces* for a discussion on combining surfaces).

To refine the search further, you can specify a positive integer value, *N*, for the number of layers of each fastener.

Abaqus searches for the $N+1$ fastening points closest to the positioning point. If BEAMMPCs are used to model the fastener, a warning message is issued if the requisite number of fastening points is not found. However, if connector elements are used to model the fastener and the requisite number of fastening points is not found, Abaqus issues an error message. Thus, when specifying the number of layers, you should ensure that the search radius has been specified such that $N+1$ fastening points can be found.

If multiple surfaces are listed as part of the fastener definition, the number of layers for each fastener is ignored. If a user-specified search radius is used for the multiple surface case, Abaqus searches for fastening points on all facets belonging to each of the listed surfaces that fall within a sphere of user-specified radius *R* with its center at the positioning point. Facets of the listed multiple surfaces that lie outside this sphere are not included in the search. A maximum of 15 layers can be specified for a particular fastener definition.

You should always examine the fastener definitions that Abaqus created to make sure that they are appropriate for your model.

Input File Usage: **FASTENER*, SEARCH RADIUS=*R*, NUMBER OF LAYERS=*N*
first data line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Criteria**: **Search radius**: **Specify**: *R*, **Maximum layers for projection**: **Specify**: *N*

Defining the Radius of Influence

Each fastening point is associated with a group of nodes on the surface in the immediate neighborhood of the fastening point called a region of influence. The motion of the fastening point is then coupled in a weighted sense to the motion of the nodes in this region by a distributed coupling constraint. Several weighting options are available and are discussed in the next section.

To define the region of influence, Abaqus computes an internal radius of influence based on the geometric properties of the fastener, the characteristic length of the connected facets, and the type of weighting function used. The default radius of influence is always chosen to be the largest of the internally computed radius of influence, the physical fastener radius, and the distance of the projection point to the closest node. You can also specify the desired radius of influence. However, Abaqus overrides a user-specified radius of influence that is smaller than the computed default radius of influence. In any case each region of influence will contain a minimum of three nodes.

Input File Usage: **FASTENER*, RADIUS OF INFLUENCE=*distance*
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Adjust**: **Influence radius**: **Specify**: *distance*

Defining the Weighting Method

The weighting methods available for the distributed coupling constraints created for a fastener interaction are the same as those available for the surface-based coupling constraints in Abaqus (see *Coupling Constraints*). Besides an area-based uniform weighting scheme, various weighting methods are provided that monotonically decrease with radial distance from the fastening point: linear, quadratic, and cubic polynomial weight distributions. By default, Abaqus uses the uniform weighting method. You can modify the default weighting distribution.

The default radius of influence calculated by Abaqus is larger for higher-order weighting methods since the resulting weights for nodes away from the fastening point contribute comparatively little to the motion of the

fastening point. Hence, to ensure that there is a sufficient "smearing" effect, it becomes necessary to increase the number of nodes in the region of influence by increasing the size of the default radius of influence. In comparison, for a uniform weighting scheme, surface nodes away from the fastening point contribute significantly to the motion of the fastening point. For this case the default radius of influence chosen can be comparatively small, since even with a small number of nodes in the region of influence, the smearing effect is sufficiently strong. If fewer than three cloud nodes are found, increasing the radius of influence might help in forming the fastener by including more nodes in the cloud of coupling nodes.

Input File Usage: Use the following option to specify a uniform weight distribution:

**FASTENER*, WEIGHTING METHOD=UNIFORM
blank line

Use the following option to specify a linear weight distribution:

**FASTENER*, WEIGHTING METHOD=LINEAR
blank line

Use the following option to specify a quadratic polynomial weight distribution:

**FASTENER*, WEIGHTING METHOD=QUADRATIC
blank line

Use the following option to specify a cubic polynomial weight distribution:

**FASTENER*, WEIGHTING METHOD=CUBIC
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Formulation**: **Weighting method**: **Uniform**, **Linear**, **Quadratic**, or **Cubic**

Defining the Fastener Orientation

Each fastener is formulated in a local coordinate system that rotates with the motion of the fastener. By default, Abaqus defines the local system by projecting the global coordinate system onto the surfaces that are being fastened according to the usual convention for surfaces in space (see *Conventions*). Local directions specified in this manner are such that the local z-axis for each fastener is normal to the surface that is closest to the reference node for the fastener.

You can override the default local system by specifying a local coordinate system for the fastener interaction. Generally, the user-defined orientation should be such that the local z-axis of the orientation is approximately normal to the surfaces that are being connected and the local x- and y-axes are approximately tangent to the surfaces that are being connected. By default, Abaqus adjusts the user-defined orientation such that the local z-axis for each fastener is normal to the surface that is closest to the reference node for the fastener. In cases where you wish to define the local directions precisely, you can specify that Abaqus should not adjust them.

Fasteners support only rectangular, cylindrical, and spherical orientation definitions. Additional rotations defined as part of the orientation definition are ignored.

In geometrically nonlinear analysis steps the local directions rotate with the motion of the fastener reference node.

Local Coordinate System When Connector Elements Are Used

If a connector element is used to model a fastener, the local coordinate system defined on the connector section, $\mathbf{T}_{connector}$, operates on the local coordinate system for the fastener, $\mathbf{T}_{fastener}$, to determine the final local coordinate system of the connector element, $\mathbf{T}_{connectorfinal}$. In other words,

$$\mathbf{T}_{connectorfinal} = \mathbf{T}_{connector} * \mathbf{T}_{fastener}.$$

In the above equations $\mathbf{T}_{connector}$ and $\mathbf{T}_{fastener}$ are assumed to be orthogonal rotation matrices with the local 1-, 2-, and 3-directions being the first, second, and third rows, respectively. The local coordinate system for a connector element modeling a fastener should be specified with respect to the local coordinate system of the fastener. The orientation displayed in the Visualization module of Abaqus/CAE (Abaqus/Viewer) is $\mathbf{T}_{connectorfinal}$ at all fastener locations unless you specify not to write the orientations to the database; in this case, only $\mathbf{T}_{connector}$ is displayed. If connector or displacement field output is requested, field output for additional nodal rotation at the connector nodes is generated automatically to ensure that the appropriate connector orientation directions are displayed as the analysis progresses. Otherwise, the orientation $\mathbf{T}_{connectorfinal}$ computed at the beginning of the analysis is displayed at all times with the updated orientations used for computation purposes.

For example, suppose you use a HINGE connector and want the released rotational degree of freedom, which is in the connector's local 1-direction, to be normal to the surfaces that are being fastened. If the default local coordinate system is used for the fastener (local 3-direction normal to the surface), the local 1-direction for the connector should be set to (0., 0., 1.); that is, the local 3-direction of the fastener. When compounded with the local coordinate system for the fastener, the local 1-direction for the connector will be normal to the surface. See *Mesh-independent spot welds* for an example of a compounded orientation.

Input File Usage: **FASTENER*, ORIENTATION=*orientation name*,
ADJUST ORIENTATION=NO
blank line

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Adjust**: **Fastener CSYS**: **Edit**: select local coordinate system, toggle off **Adjust CSYS to make local Z-axis normal to closest surface**

Clarifications regarding the Computation of $\mathbf{T}_{fastener}$

A few clarifications regarding the default definition of $\mathbf{T}_{fastener}$ are necessary for a precise understanding of the behavior when connector elements are used to model fasteners. The positioning point is always projected on the closest surface to be fastened. Therefore, the choice of coordinates of the reference node relative to the stack of surfaces to be fastened determines which surface is used to compute the local directions. Typically this choice does not matter much in realistic applications because the surfaces to be fastened are more or less parallel to each other in the fastener area.

The projection of the reference node on the closest surface generates a fastening point for the connector element.

The local z-axis for each fastener ($\mathbf{T}_{fastener}$) is normal to the surface at this fastening point. The fastening point generated on the closest surface is by default the first fastening point and, therefore, the first connector node. The precise direction into which the local z-axis is pointing is chosen such that the dot product with the unit vector pointing from the first node of the connector to the second node of the connector is positive. As explained above, you can control the connectivity of the fastening points in the connectors by specifying unsorted surfaces.

Therefore, you can control the precise direction the local *z*-axis is pointing along the surface normal by either selecting appropriate coordinates for the reference node and/or by using unsorted surfaces.

The two tangential directions in $\mathbf{T}_{fastener}$ are computed by default according to the usual convention for surfaces in space (see *Conventions*). The global *X*-axis is projected onto the closest surface at the location of the fastening point to determine the local *x*-axis in $\mathbf{T}_{fastener}$. If the global *X*-axis is within 0.1 degrees of being normal to the surface, the local *x*-axis in $\mathbf{T}_{fastener}$ is the projection of the global *Z*-axis on the closest surface. The local *y*-axis in $\mathbf{T}_{fastener}$ is then at right angles to the local *x*-axis and *z*-axis so that the three local axes form a right-handed set.

In the rare cases when the default definition of $\mathbf{T}_{fastener}$ does not suit your application, you can always specify the orientation directly. In this case the following occurs:

1. Abaqus first recomputes the local *z*-axis to align with the facet normal, with the precise direction chosen such that its dot product with the unit vector pointing from the first node of the connector to the second node of the connector is positive.
2. Abaqus checks the local *x*- and *y*-axes you specified to determine which of these two is closest to the plane of the current facet.
3. If the local *x*-axis is closest, Abaqus recomputes the local *y*-axis as the normalized cross product of the recomputed *z*-axis and the specified *x*-axis. Then Abaqus computes the new local *x*-axis as the normalized cross product of the recomputed *y*-axis and the recomputed *z*-axis.
4. If the local *y*-axis is closest, Abaqus recomputes the local *x*-axis as the normalized cross product of the specified *y*-axis and the recomputed *z*-axis. Then Abaqus computes the new local *y*-axis as the normalized cross product of the recomputed *z*-axis and the recomputed *x*-axis.

Common Modeling Practices

In most applications the default choice for $\mathbf{T}_{fastener}$ combined with a choice of global system for $\mathbf{T}_{connector}$ at both connector nodes would result in a $\mathbf{T}_{connector\,final}$ that is most suitable. The connection type that you choose depends on several modeling considerations, but very often the BUSHING connection type offers the best choice. To simplify the discussion, consider that only two surfaces are being fastened, a very common situation as illustrated in the spot weld example in *Connector Functions for Coupled Behavior*. For this common choice, $\mathbf{T}_{connector\,final}$ has the local *z*-axis normal to the closest surface and pointing from the first fastening point (first connector node) toward the second fastening point (second connector node). This choice ensures that for a fastener subjected to a tension load (fastened plates pulled apart) a positive force always develops in the connector along the local *z*-axis (CTF3) regardless of the choice of coordinates for the positioning point and/or use of unsorted surfaces. Conversely, if a compression load is applied (fastened plates pressed against each other), a negative force develops in the connector.

In most cases, the behavior in the tangential plane defined by the local *x*- and local *y*-axes is isotropic; therefore, the precise orientation of these two axes is of less interest to you. The spot weld example in *Connector Functions for Coupled Behavior* illustrates such a typical case where the (isotropic) magnitude of two in-plane forces (f_1, f_2) and of the two moments (m_1, m_2) are used in the kinetic behavior of the connector element.

If you need to specify anisotropic behavior in the tangential plane, you need to understand precisely how the directions in $\mathbf{T}_{fastener}$ are defined. As explained above, the choice of coordinates for the positioning point relative to the stack of surfaces to be fastened and/or use of unsorted surfaces determines the precise direction of the default local axes. In most cases you have two common modeling choices. In the first case you can specify the coordinates of the positioning points to be exactly on or very close to the surface onto which the first fastening points (connector nodes) are to be placed and use the default sorted surfaces. In this case you do not need to

specify the surfaces to be fastened individually. However, in many practical situations imprecise geometry for the surfaces to be fastened and/or inexact coordinates of the fastener reference nodes make the consistent placement of the reference nodes in the vicinity of one particular surface very hard to accomplish. The second modeling technique consists of using sorted surfaces. The exact location of the reference node with respect to the surface stack to be fastened is not that important because the first fastening point is always on the first specified surface. In this case you do have to specify two or more individual surfaces to be fastened. In the rare cases when neither of these modeling techniques suits your application, you can specify the fastener orientation directly to match your needs exactly.

Coupling Fasteners to Surfaces

Mesh-independent fasteners constrain the translation and rotation of each fastening point to the average motion of the associated coupling cloud nodes of a fastened surface. The cloud nodes follow the motion of the reference node in an average sense, with deformation allowed among cloud nodes.

Translational and rotational constraints for the fastening points are formed similarly to distributing coupling constraints, as described in more detail in *Distributing Coupling Constraints*. As for distributed coupling constraints, the fastener couplings constraints are formed such that:

- Cloud node translations always influence both translation and rotation of the fastening point.
- Cloud node rotations typically have a minor influence on the rotation of the fastening point. This influence becomes more significant if cloud nodes are collinear or nearly collinear (see *Distributing Coupling Constraints*).
- The translational coupling constraint is such that translation of the weighted center of the cloud corresponds to the average translation of the cloud nodes. By default, the fastening point translation follows the weighted center translation plus the effect of rotation of a rigid arm from the weighted center to the fastening point. This rigid-arm rotation corresponds to the fastening point rotation. Cloud node rotational degrees of freedom can participate in an additional offset term for the translational coupling constraint, such that the fastening point remains close to a shell surface during bending.

Input File Usage: **FASTENER*

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Formulation**: **Coupling type**: **Continuum distributing**

Optional Offset Associated with Bending

In many cases when the pair of fastened surfaces are close to each other, unrealistic contact interactions might occur between the two surfaces unless an evolving offset associated with bending is introduced in the translational coupling constraints. This offset introduces the influence of cloud node rotational degrees of freedom into the translational coupling constraints. For this coupling method to be active, all rotational degrees of freedom at all coupling nodes must be active (as is the case when shells are fastened together) and all degrees of freedom must be constrained (which is the default; see *Defining Fastener Properties* below).

Use of this option is independent of whether or not cloud node rotations influence rotational coupling constraints.

Input File Usage: **FASTENER*, COUPLING=STRUCTURAL

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Formulation**: **Coupling type**: **Structural distributing**

Controlling Whether Cloud Node rotations Influence Fastening Point Rotation

By default, Abaqus couples fastening point rotations to cloud node translations alone, unless all cloud nodes are nearly collinear and have rotational degree of freedom. You can optionally control whether or not the cloud node rotations influence the fastening point rotation. However, these options are subject to the following conditions:

- If the cloud nodes are collinear, Abaqus cannot couple the fastening point rotation about the axis of collinearity of the cloud system to the cloud node translations alone. When this situation arises, Abaqus issues a warning message that identifies the axis about which the fastening point rotation is not coupled to the cloud system. A consequence of the above is that a component of moment about the axis of collinearity acting at the reference node will not be transmitted to the cloud system.
- If a cloud nodes are not nearly collinear and you force cloud node rotations to influence fastening point rotation, the actual influence of cloud node rotations on the fastening point rotation is typically insignificant. Some performance degradation can occur due to appearance of additional terms in the coupling equations.

Input File Usage: Use the following option (default) to allow Abaqus to choose if the cloud nodes rotations should be included in the rotational constraint:

**FASTENER*, ROTATIONAL COUPLING=AUTOMATIC(default)

Use the following option to neglect the cloud node rotations from the rotational constraint:

**FASTENER*, ROTATIONAL COUPLING=CONTINUUM

Use the following option to include the cloud node rotations from the rotational constraint:

**FASTENER*, ROTATIONAL COUPLING=STRUCTURAL

Abaqus/CAE Usage: A nondefault option for Abaqus/Standard to neglect cloud rotations in a rotational constraint for fastening points of mesh-independent fasteners is not supported in Abaqus/CAE.

Defining Fastener Properties

Each fastener interaction definition must refer to a property, which defines the geometric section properties of the fastener.

Input File Usage: Use both of the following options:

**FASTENER*, PROPERTY=*fastener property name*
**FASTENER PROPERTY*, NAME=*fastener property name*

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Property**

Geometric Section Quantities

Fasteners are assumed to have a circular projection onto the connected surfaces. You are required to specify the radius of the fastener.

Input File Usage: **FASTENER PROPERTY*
r

Abaqus/CAE Usage: Interaction module: **Special->Fasteners->Create**: **Point-based**: **Property**: **Physical radius**: *r*

Releasing Degrees of Freedom on Fasteners Using Connector Elements

For fasteners modeled with connector elements, translational as well as rotational degrees of freedom can be released by prescribing connector section types that have unconstrained (available) degrees of freedom. For example, a HINGE connector can be used to release the rotational degree of freedom in the connector's local 1-direction.

Releasing Degrees of Freedom on Fasteners Using BEAMMPCS

For fasteners modeled with BEAMMPCs, the moment constraint between the rotation degrees of freedom at the fastening points and the average rotation of the coupling nodes can be released in one, two, or three directions. You can specify the moment constraint directions in the default local coordinate system or a user-defined local coordinate system. The three translational degrees of freedom at the fastening points are always coupled to the average translation of the coupling nodes. You specify the degrees of freedom of the fastening point to be coupled to the average motion of the coupling nodes as part of the fastener property definition.

If no degrees of freedom are specified as part of the fastener property definition, all six degrees of freedom are coupled. If you specify one or more degrees of freedom but not all available translation degrees of freedom, Abaqus issues a warning message and adds all the available translation degrees of freedom to the constraint. If a user-specified local orientation is specified for the fastener interaction, the local degrees of freedom are with respect to the user-defined coordinate system.

Input File Usage: **FASTENER PROPERTY*
section properties
first dof, *last dof*

For example, if the default local coordinate system is used, the following property definition would release the relative rotation constraint of the connected parts about the surface normal:

**FASTENER PROPERTY*
section properties
1, 5

The above property definition might be used to approximate a riveted connection.

Abaqus/CAE Usage: Abaqus/CAE always constrains all translational degrees of freedom in a fastener. Use the following input to remove constraints on the rotational degrees of freedom:

Interaction module: **Special->Fasteners->Create**: **Point-based**: **Formulation**: toggle off **UR1**, **UR2**, or **UR3**

Overconstraints in Fasteners Modeled with BEAMMPCS

There are several instances in which a model with fasteners modeled with BEAMMPCs might be overconstrained. Described below are two potential overconstraints that Abaqus automatically attempts to detect and resolve during solver input file processing.

Fasteners and Rigid Bodies

Fasteners can be used to connect both deformable and rigid element-based surfaces. However, if the fasteners are modeled with BEAMMPCs, potential overconstraints might arise if more than one rigid surface is involved in a given fastener definition. Abaqus automatically attempts to remove these types of overconstraints by allowing at most one rigid surface in any individual fastener definition. A warning message is generated if an overconstraint of this type is detected.

For example, suppose surfaces A and C in *Figure 1* are part of the same rigid body, and surface B is deformable. Abaqus automatically removes either surface A or surface C from the fastener definition and only forms the fastener between the deformable surface and the remaining rigid surface. If surface A and surface C belong to two separate rigid bodies, their respective rigid body reference nodes will be joined by an internally generated BEAMMPC.

In another example, suppose all three surfaces in *Figure 1* are rigid. In this case no fastener will be formed, and the unique rigid body reference nodes for surfaces A, B, and C will be joined by beam MPCs. Unresolvable overconstraints might arise if inconsistent kinematic constraints (such as displacement boundary conditions) are placed on rigid body reference nodes that have been joined by BEAMMPCs. In this case you must modify the model to resolve the overconstraints. Possible courses of action include removing some of the rigid surfaces from the fastener definitions or removing inconsistent kinematic conditions on the rigid body reference nodes.

The above-described procedure to resolve overconstraints with fasteners and rigid bodies will preserve the kinematics of the original model. In Abaqus/Standard you can bypass the overconstraint checks and prevent automatic model modifications in the model preprocessor (see *Overconstraint Checks*).

Overlapping Fasteners

Potential overconstraints exist with rigid fasteners if all the coupling nodes of any associated distributing coupling element are wholly contained within one or more other fastener definitions. This can happen if the spacing between positioning points is small compared to the typical element size in a mesh (which is often the case in automotive models). To avoid overconstraints in this situation, Abaqus uses a penalty formulation for all fastener distributing coupling elements that satisfy the above criteria. The penalty distributing coupling formulation relaxes, to a small degree, the constraint between the motion of the distributing coupling element reference node and its coupling nodes.

Output

If fasteners are modeled using connector elements, connector element output variables can be used to request output for fasteners (see *Connector Elements*). No fastener output is available if the fasteners are modeled using BEAMMPCs.

Embedded Elements

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Kinematic Constraints*
- **EMBEDDED ELEMENT*
- *Defining embedded region constraints*

Overview

The embedded element technique constrains "embedded" elements to move with "host" elements.

The embedded element technique:

- is used to specify an element or a group of elements that lie embedded in a group of host elements whose response is used to constrain the degrees of freedom of the embedded nodes (that is, nodes of embedded elements) that are in common with the degrees of freedom of host elements ;
- can be used in geometrically linear or nonlinear analysis;
- is not available for host elements with rotational degrees of freedom;
- can be used to model a set of rebar-reinforced membrane, shell, or surface elements that lie embedded in a set of three-dimensional solid (continuum) elements; a set of truss or beam elements that lie embedded in a set of solid elements; or a set of solid elements that lie embedded in another set of solid elements;
- will not constrain rotational degrees of freedom of the embedded nodes when shell or beam elements are embedded in solid elements; and
- can be imported from Abaqus/Standard into Abaqus/Explicit and vice versa.

Introduction

The embedded element technique is used to specify that an element or group of elements is embedded in "host" elements. For example, the embedded element technique can be used to model rebar reinforcement. Abaqus searches for the geometric relationships between nodes of the embedded elements and the host elements. If a node of an embedded element lies within a host element, degrees of freedom that are in common with the host element, except rotational degrees of freedom, are eliminated and the node becomes an "embedded node." The common degrees of freedom of the embedded node are constrained to the interpolated values of the corresponding degrees of freedom of the host element. Embedded elements are allowed to have rotational degrees of freedom, but these rotations are not constrained by the embedding. Multiple embedded element definitions are allowed.

Available Embedded Element Types

Different element types can be used in the element set containing embedded elements and the element set containing the host elements. The number of translational degrees of freedom at a node on the embedded element must be identical to the number of translational degrees of freedom at a node on the host element. If elements of type FP2D2, FP3D2, FPC2D2, and FPC3D2 with only pore pressure degree of freedom are embedded in a host element that has both translational and pore pressure degrees of freedom, only the common pore pressure degree of freedom is constrained at the embedded node.

The following general types of “embedded elements-in-host elements” are provided:

- Two-dimensional models:
  - Beam-in-solid
  - Solid-in-solid
  - Truss-in-solid
  - Fluid pipe-in-solid
- Axisymmetric models:
  - Membrane-in-solid (Abaqus/Standard only)
  - Shell-in-solid
  - Solid-in-solid
  - Surface-in-solid (Abaqus/Standard only)
- Three-dimensional models:
  - Beam-in-solid
  - Membrane-in-solid
  - Shell-in-solid
  - Solid-in-solid
  - Surface-in-solid
  - Truss-in-solid
  - Fluid pipe-in-solid

Specifying the Host Elements

By default, the elements in the vicinity of the embedded elements are searched for elements that contain embedded nodes; the embedded nodes are then constrained by the response of these host elements. To preclude certain elements from constraining the embedded nodes, you can define a host element set; the search will be limited to this subset of the host elements in the model. This feature is strongly recommended if the embedded nodes are close to discontinuities in the model (such as cracks and contact pairs).

Input File Usage: **EMBEDDED ELEMENT*, HOST ELSET=*name*

The **EMBEDDED ELEMENT* option must be included in the model definition portion of the input file. Multiple **EMBEDDED ELEMENT* options are allowed.

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Embedded region**: choose **Select Region** from the prompt area when selecting the host region

Specifying the Embedded Elements

You must specify the embedded elements. Individual elements or element sets can be specified. By default, Abaqus issues an error message if it is unsuccessful in fully embedding all of the specified embedded elements into host elements. Optionally, you can allow partial embedding in which only those nodes of embedded elements within host elements will be constrained.

An embedded element might share some nodes with host elements. These nodes, however, will not be considered to be embedded nodes.

Input File Usage: Use the following option to fully embed the elements (default):

**EMBEDDED ELEMENT*, PARTIAL EMBED=NO
embedded elements

Use the following option to partially embed the elements:

**EMBEDDED ELEMENT*, PARTIAL EMBED=YES
embedded elements

Abaqus/CAE Usage: You can only fully embed elements in Abaqus/CAE.

Interaction module: **Create Constraint**: **Embedded region**: select the embedded region

Specifying the Embedded Nodes

Optionally, you can specify the embedded nodes. Individual nodes or node sets can be specified. By default, Abaqus issues an error message if it is unsuccessful in fully embedding all of the specified embedded nodes into host elements.

Input File Usage: **EMBEDDED ELEMENT*, EMBED NODES
embedded nodes

Abaqus/CAE Usage: Embedding nodes is not supported in Abaqus/CAE.

Defining Geometric Tolerances

A geometric tolerance is used to define how far an embedded node can lie outside the regions of the host elements in the model. By default, embedded nodes must lie within a distance calculated by multiplying the average size of all nonembedded elements in the model by 0.05; however, you can change this tolerance.

You can define the geometric tolerance as a fraction of the average size of all nonembedded elements in the model. Alternatively, you can define the geometric tolerance as an absolute distance in the length units chosen for the model. If you specify both exterior tolerances, Abaqus uses the tighter tolerance of the two. The average size of all the nonembedded elements is calculated and multiplied by the fractional exterior, which is then compared to the absolute exterior tolerance to determine the tighter tolerance of the two. The exterior tolerance for embedded elements in host elements is indicated by the shaded region in *Figure 1*.

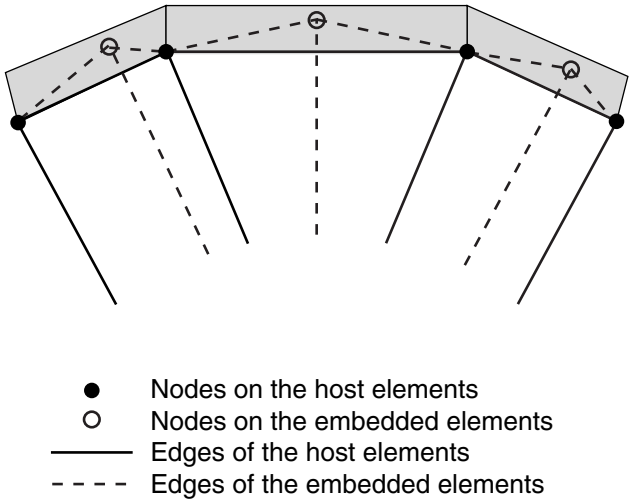


Figure 1: The exterior tolerance for embedded elements.

If an embedded node is located inside the specified tolerance zone, the node is constrained to the host elements. The position of this node will be adjusted to move the node precisely onto the host elements. If an embedded node is located outside the specified tolerance zone, an error message will be issued.

Input File Usage: Use the following option to define the tolerance as a fraction:

**EMBEDDED ELEMENT*, EXTERIOR TOLERANCE=*tolerance*

Use the following option to define the tolerance as an absolute distance:

**EMBEDDED ELEMENT*,
ABSOLUTE EXTERIOR TOLERANCE=*tolerance*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Embedded region**: **Fractional exterior tolerance** or **Absolute exterior tolerance**

Adjusting the Positions of Embedded Nodes

If an embedded node lies close to an element edge or an element face within a host element, it is computationally efficient to make a small adjustment to the position of the embedded node so that the node will lie precisely on the edge or face of the host element. A small tolerance, below which the weight factors of the nodes on a host element associated with an embedded node will be zeroed out, is defined. The small weight factors will be redistributed to the other nodes on the host element in proportion to their initial weights, and the position of the embedded node will be adjusted based on the new weight factors. This adjustment is performed only at the start of the analysis and does not create any strain in the model. It is most useful for making small adjustments to make the embedded nodes lie on the edge or face of a host element. If a large nondefault value of the roundoff tolerance is used to make significant adjustments to the positions of the embedded nodes, you should carefully review the mesh obtained after adjusting.

Input File Usage: **EMBEDDED ELEMENT*, ROUNDOFF TOLERANCE=*tolerance*

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Embedded region**: **Weight factor roundoff tolerance**

Use with Other Multiple Kinematic Constraints

If an embedded node is also tied by multi-point, equation, kinematic coupling, surface-based tie, or rigid body constraints, an overconstraint is introduced and an error message will be issued. If a boundary condition is applied to an embedded node, the embedded element definition always takes precedence. The boundary condition will be neglected, and a warning message will be issued.

Defining Surfaces on Embedded Elements

The faces of the embedded elements are not considered part of the all-inclusive surface defined automatically for interactions modeled with general contact, regardless of whether the elements are specified as fully or partially embedded. In addition, any surface definitions based on these elements must have the face identifier specified explicitly (see *Element-Based Surface Definition*).

Limitations

The following limitations exist for the embedded element technique:

- Elements with rotational degrees of freedom (except axisymmetric elements with twist) cannot be used as host elements.
- Rotational degrees of freedom at an embedded node are not constrained.
- Host elements cannot be embedded themselves.

- The material defined for the host element is not replaced by the material defined for the embedded element at the same location of the integration point.
- Additional mass and stiffness due to the embedded elements are added to the model.
- If modified tetrahedron elements are used as host elements, only the corner nodes are used to constrain the appropriate embedded nodes.

Example

Consider the example in *Figure 2*. Elements 3 (truss) and 4 (membrane) lie embedded in elements 1 and 2. Element 1 is formed by nodes a, b, c, d, e, f, g, and h; element 2 is formed by nodes e, f, g, h, i, j, k, and l; element 3 is formed by nodes A and B; and element 4 is formed by nodes C, D, E, and F. If the host element set includes elements 1 and 2 and the embedded element sets contain elements 3 and 4, respectively, Abaqus will attempt to find if there are any embedded nodes (A, B, C, D, E, and F) lying within host elements 1 or 2. If node A is found to be lying close to the a-b-f-e face of element 1, all the degrees of freedom at node A are constrained to nodes a, b, f, and e, with appropriate weight factors being determined based on the geometric location of node A in element 1. Similarly, if node B is found to be lying inside element 1 and node E is found to be lying close to the g–k edge of element 2, respectively, all the degrees of freedom at node B are constrained to nodes a, b, c, d, e, f, g, and h, and all the degrees of freedom at node E are constrained to nodes g and k, with appropriate weight factors being determined based on the geometric location of node B in element 1 and the geometric location of node E on the g–k edge of element 2, respectively.

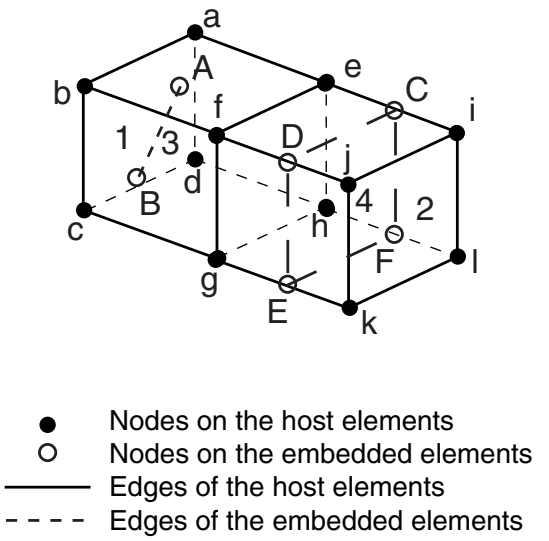


Figure 2: Elements lie embedded in host elements.

You should make sure that all the nodes on the desired embedded elements are properly constrained to nodes on the host elements. This can be verified by performing a data check analysis (see *Abaqus/Standard and Abaqus/Explicit Execution*). For each embedded node a list of nodes that are used to constrain this node and the associated weight factors are output to the data file during the data check analysis. An error message is issued if an embedded node is not constrained and full embedment is used.

Template

```
*HEADING
…
*NODE
Data line to define the nodal coordinates
*ELEMENT, TYPE=C3D8, ELSET=SOLID3D
Data line to define the solid elements
*ELEMENT, TYPE=T3D2, ELSET=TRUSS
Data line to define the truss elements
*ELEMENT, TYPE=M3D4, ELSET=MEMB
Data line to define the membrane elements
*EMBEDDED ELEMENT, EXTERIOR TOLERANCE=tolerance, HOST ELSET=SOLID3D
TRUSS, MEMB
*STEP
*STATIC (or any other allowable procedure)
Data line to define step time and control incrementation
…
*END STEP
```

Element End Release

Products: Abaqus/Standard

References:

- *About Kinematic Constraints*
- **RELEASE*

Overview

Element end release introduces hinge behavior at an end of a beam or pipe element in Abaqus/Standard.

Element end release:

- allows a rotational degree of freedom or a combination of rotational degrees of freedom to be released at one or both ends of an element or element set;
- can be used in geometrically linear or nonlinear analysis; and
- is available only for beam and pipe elements in Abaqus/Standard.

Introduction

Element end release is used to model hinged connections (hinged in one, two, or three orthogonal directions) at one or both ends of the element. By releasing rotational degrees of freedom, an element end is allowed to rotate freely relative to the node about the chosen degrees of freedom. Any rotational degrees of freedom that are not released are shared with the node. You must be careful not to release a given degree of freedom at a node for all elements that share that node; otherwise, the node has no stiffness for that degree of freedom and Abaqus/Standard issues zero pivot warning messages.

Element end release operates on the element local degrees of freedom. See *Beam Element Cross-Section Orientation* for a definition of the local axes ($\boldsymbol{n}_1$, $\boldsymbol{n}_2$, t) for beam-type elements. The rotational degrees of freedom affected by the release are the rotation about the local $\boldsymbol{n}_1$-axis, the rotation about the local $\boldsymbol{n}_2$-axis, and the rotation about the local t-axis for beams in space. For beams in a plane, only the rotation about the local $\boldsymbol{n}_1$-axis is active (which coincides with rotations about the negative global z-axis).

Equivalent MPCS

If only one rotational degree of freedom is released, the kinematic constraint is equivalent to MPC type REVOLUTE plus MPC type PIN between two nodes. If two rotational degrees of freedom are released, the kinematic constraint is equivalent to MPC type UNIVERSAL plus MPC type PIN. If all rotational degrees of freedom are released, the kinematic constraint is equivalent to MPC type PIN. See *General Multi-Point Constraints* for details.

Identifying the Element End Involved in the Release

Either element sets or individual elements can be specified for a release definition. Degrees of freedom can be released at the first, second, or first and second ends of an element. The first end of the element, S1, is node 1 on the element as defined by the element connectivity; the second end, S2, is the last node (node 2 or 3, as appropriate) on the element. See *Beam Element Library* for a definition of the node ordering for beam elements.

Identifying the Local Rotational Degrees of Freedom Involved in the Release

Rotation combination codes rather than degrees of freedom are specified to identify the rotational degrees of freedom involved in the release.

M1

refers to the rotation about the n_1-axis,

M2

refers to the rotation about the n_2-axis,

M1-M2

refers to a combination of rotational degrees of freedom about the n_1-axis and the n_2-axis,

T

refers to the rotation about the *t*-axis,

M1-T

refers to a combination of rotational degrees of freedom about the n_1-axis and the *t*-axis,

M2-T

refers to a combination of rotational degrees of freedom about the n_2-axis and the *t*-axis, and

ALLM

represents a combination of all the rotational degrees of freedom (that is, M1, M2, and T).

Input File Usage: **RELEASE*
element number or element set, element end ID, release combination code

For example, to release the rotational degree of freedom about the n_1-axis at the first end of element 10 and all the rotational degrees of freedom at the second end of the element, use the following input:

```
*RELEASE
10, S1, M1
10, S2, ALLM
```

Use with Transformed Coordinate Systems

Transformations applied to released nodes (*Transformed Coordinate Systems*) have no influence on the release. The release operates on the local degrees of freedom for the element.

Reading the Data from an Alternate Input File

The data for a release definition can be contained in a separate input file.

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

If the INPUT parameter is omitted, it is assumed that the data lines follow the keyword line.

Overconstraint Checks

Products: Abaqus/Standard

References:

- *Rigid Body Definition*
- *About Connectors*
- *Boundary Conditions*
- *General Multi-Point Constraints*
- *Mesh Tie Constraints*
- *Coupling Constraints*
- *Mesh-Independent Fasteners*
- *About Contact Pairs in Abaqus/Standard*
- **BASE MOTION*
- **CONSTRAINT CONTROLS*

Overview

Abaqus/Standard checks for problematic overlaps among various constraints.

An overconstraint means applying multiple consistent or inconsistent kinematic constraints. Many models have nodal degrees of freedom that are overconstrained. Such overconstraints might lead to inaccurate solutions or nonconvergence. Common examples of situations that might lead to overconstraints include (but are not limited to):

- contact secondary nodes that are involved in boundary conditions or multi-point constraints;
- edges of surfaces involved in a surface-based tie constraint that are included in contact secondary surfaces or have symmetry boundary conditions; and
- boundary conditions applied to nodes already involved in coupling or rigid body constraints.

The overconstraint checks performed in Abaqus/Standard:

- check for overconstraints caused by combinations of the following: base motions, boundary conditions, contact pairs, coupling constraints, linear constraint equations, mesh-independent spot welds, multi-point constraints, rigid body constraints, and surface-based tie constraints;
- check for overconstraints resulting from kinematic constraints introduced through connector elements, coupling elements, special-purpose contact elements, and elements with incompressible material behavior;
- identify through detailed messages the constraints that cause overconstraints;
- automatically resolve a limited set of consistent overconstraints detected during model preprocessing and during an Abaqus/Standard analysis;
- use the equation solver to detect overconstraints that cannot be resolved automatically; and
- can have the default behavior modified.

Overconstraints: General Remarks

In general, the term overconstraint refers to multiple constraints acting on the same degree of freedom. Overconstraints are then categorized as *consistent* (if all the constraints are compatible with each other) or *inconsistent* (if the constraints are incompatible with each other). Consistent overconstraints are also called *redundant* constraints, and inconsistent overconstraints are also called *conflicting* constraints.

In Abaqus/Standard the following types of constraints, in combination, might lead to overconstraints:

- boundary conditions or base motions,
- contact pairs,
- coupling constraints,
- mesh-independent spot welds,
- multi-point constraints or linear constraint equations,
- surface-based tie constraints, and
- rigid body constraints.

In addition to these constraints the following elements impose kinematic constraints and, when used in combination with each other or with the above constraints, might lead to overconstraints:

- connector elements,
- special-purpose contact elements, and
- hybrid elements for incompressible material response.

An illustration of several consistent overconstraints is given in *Figure 1*.

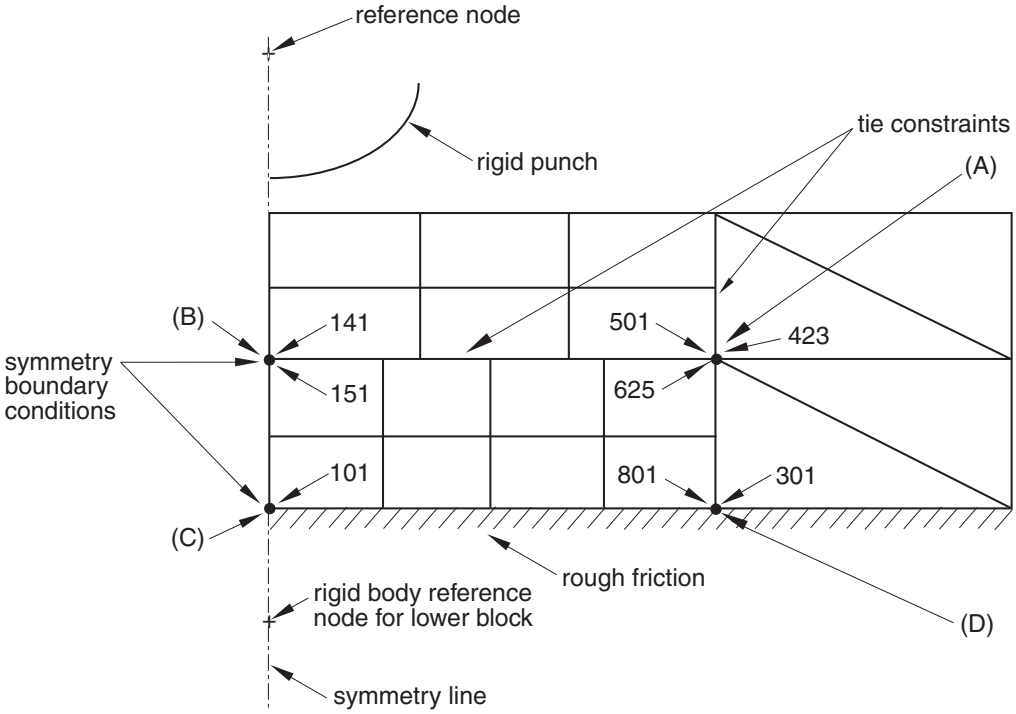


Figure 1: Model with redundant constraints.

The upper block is built from three separately meshed regions, which are connected together using a surface-based tie constraint. This block is in contact with the lower rigid block, which is made rigid by specifying a rigid body constraint. The rigid block's reference node is fixed. Symmetry boundary conditions are used at the left edge of the upper block, and rough friction is defined for the surface interaction between the upper and lower blocks. The following redundant constraints can be identified:

- Intersecting tie constraints: At (A) three nodes share the same location, and their relative motions are constrained by two surface-based tie constraints (one vertical and one horizontal). Only two constraints (two dependent nodes and one independent node) are needed to fully constrain the motion of the three nodes, but three constraints are generated internally (one for the horizontal tie constraint and two for the vertical one). Therefore, one redundant constraint exists.
- Tie constraint and symmetry boundary condition: At (B) nodes 141 and 151 have their motion constrained horizontally by the symmetry boundary condition, but their relative motion is also constrained by the surface-based tie constraint. Therefore, one redundant constraint exists.
- Rough friction and symmetry boundary condition: At (C) node 101 is constrained horizontally by the symmetry boundary condition. The rough friction contact acts in the same direction as the boundary condition. Therefore, one redundant constraint exists.
- Tie constraint and contact interactions: At (D) nodes 801 and 301 are involved in the surface-based tie constraint, but two contact constraints (one at each node) act in the vertical direction. Therefore, one redundant constraint exists.

Even in this simple model the number of redundant constraints is surprisingly large. If not appropriately accounted for, the redundant constraints can lead to convergence difficulties, even nonconvergence. Moreover, in the cases when a solution is obtained (despite the convergence difficulties), the reported reaction forces and contact pressures might be inaccurate.

Abaqus/Standard checks for the inappropriate use of combinations of constraints for the majority of constraint and element types listed in this section. Depending on the complexity of the constraints involved, Abaqus/Standard identifies three classes of consistent and inconsistent overconstraints.

Overconstraints detected in the model preprocessor

Many relatively simple overconstraints can be identified by inspecting the constraints defined at a node. If a consistent overconstraint is detected, the unnecessary constraints are eliminated automatically and a warning message is generated. If the overconstraints are inconsistent, the analysis is stopped and an error message is generated.

Overconstraints detected and resolved in an Abaqus/Standard analysis

Some overconstraints involving contact interactions might become overconstrained only during an analysis because of changes in contact status. Certain of these cases are detectable and eliminated automatically by Abaqus/Standard. Appropriate messages are issued.

Overconstraints detected by the equation solver

Many overconstraints involve complex interactions between various constraint definitions and element types. Automatic resolution of these situations might not be possible. In such cases the equation solver will detect the overconstraint, and a detailed message listing potential causes of the problem will be issued.

Overconstraints Detected in the Model Preprocessor

In this section we consider overconstraints that involve two or more of the following:

- surface-based tie constraints,
- rigid body constraints,
- boundary conditions, and
- connector elements.

While the number of cases handled automatically in the model preprocessor is limited, many often-encountered situations are corrected. The list of overconstraints to be resolved automatically in the preprocessor is organized based on the constraint types involved. Each case is illustrated by examples.

Intersecting Tie Constraints

Examples of intersecting tie constraint definitions are shown in *Figure 2*.

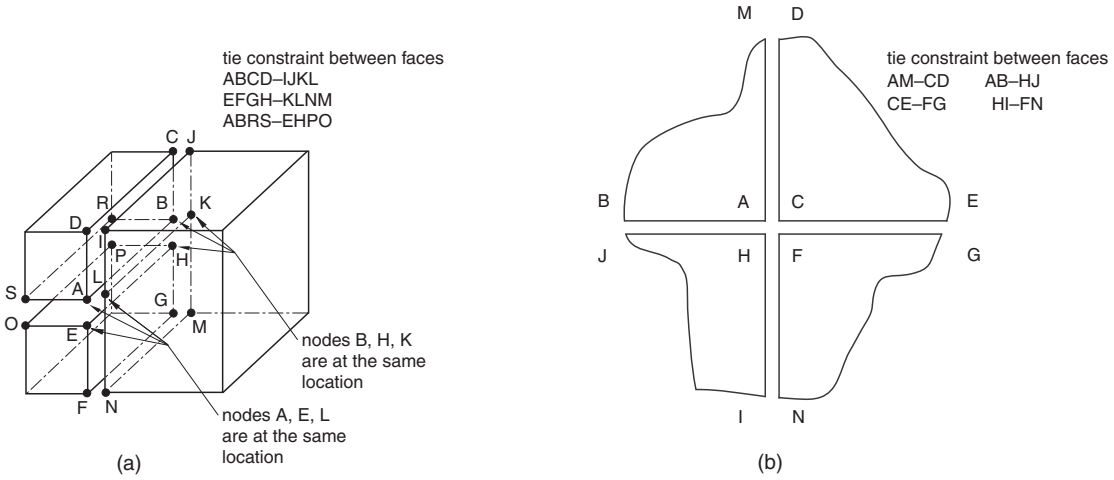


Figure 2: Consistent overconstraints because of intersecting tie constraints.

In both cases there is at least one node that, if not properly treated, will be redundantly constrained. In the case on the left, the three edges belonging to the three surfaces overlap (shown here in an exploded view for clarity). Each of the three end nodes on either end occupy the same location. Therefore, one redundant tie constraint exists. In the case shown on the right, four adjacent meshes are “glued” together using four tie constraints. Only three constraints are needed to “glue” the center nodes together, but four are generated (one from each tie constraint). Therefore, one constraint is not needed and in both cases one constraint is removed.

Tie Constraint inside a Rigid Body Constraint

An example of a tie constraint inside a rigid body constraint is shown in *Figure 3*(a). Two surfaces are connected by a tie constraint, and the two element sets are included in the same rigid body. Since the motion of all the nodes is constrained to the motion of the rigid body's reference node, the tie constraint is redundant. The tie constraint definition is removed from the model.

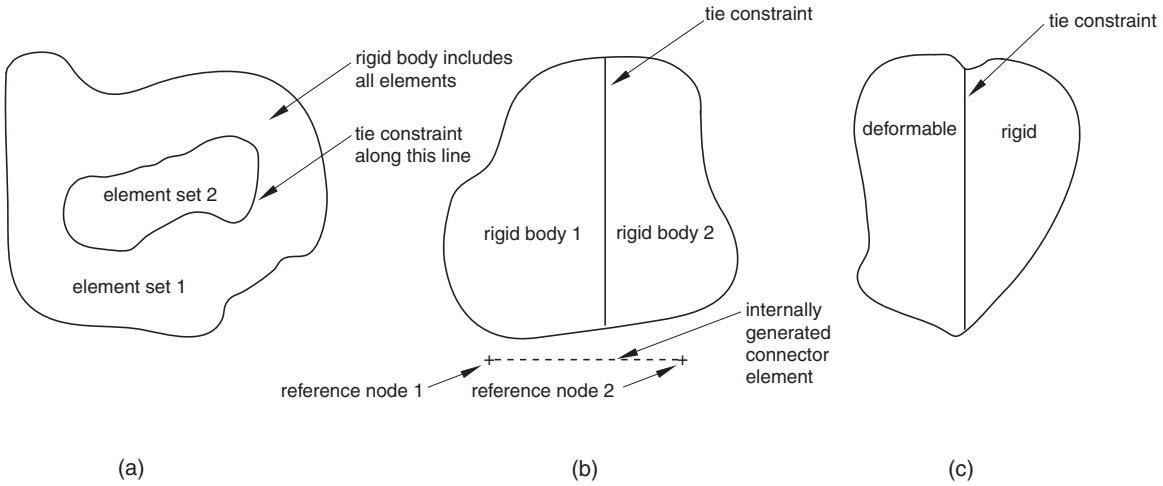


Figure 3: Consistent overconstraints because of combinations of tie and rigid body constraints.

Tie Constraint between Two Rigid Bodies

An example of a tie constraint between two rigid bodies is shown in *Figure 3*(b). If the two surfaces are connected by a tie constraint at more than two or three points (in two- or three-dimensional analyses, respectively), the tie constraint definition is redundant. A connector type BEAM is placed between the two reference nodes, and the tie constraint is removed.

Tie Constraint between a Deformable and a Rigid Body

An example of connecting a deformable body to a rigid body with a surface-based tie constraint is shown in *Figure 3*(c). If the secondary surface in the tie constraint definition belongs to the rigid body, the tie and the rigid body constraints are redundant for the secondary nodes. If possible, Abaqus/Standard will switch the main and the secondary surfaces in the tie constraint definition. If switching the main and the secondary surfaces is not possible because of other modeling restrictions, an error message is issued and the analysis is stopped.

Intersecting Rigid Bodies

Figure 4(a) illustrates the case when two rigid bodies partially overlap and, thus, the union of the two bodies behaves as one rigid body. However, the motion of the nodes in this region is governed by the motion of the two rigid body reference nodes; hence, the model is overconstrained. In *Figure 4*(b) several rigid bodies are included in a larger rigid body definition. The nodes belonging to the included bodies will be overconstrained.

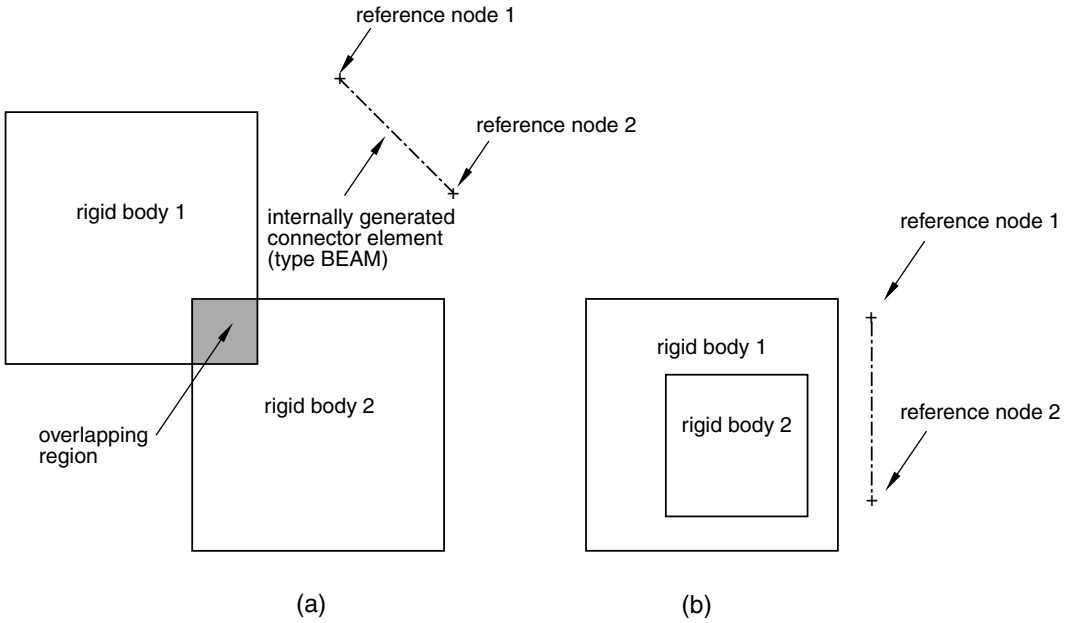


Figure 4: Rigid body including other rigid bodies.

In both cases the rigid body constraint will be enforced only once for the nodes that belong to several rigid bodies. To enforce the rigid behavior of the ensemble, connector elements of type BEAM are generated between the rigid body reference nodes to ensure a rigid connection between the intersecting rigid body definitions.

Tie Constraints and Boundary Conditions

There are numerous cases of overconstraints when a surface-based tie constraint and a boundary condition are used together, as illustrated in *Figure 5*.

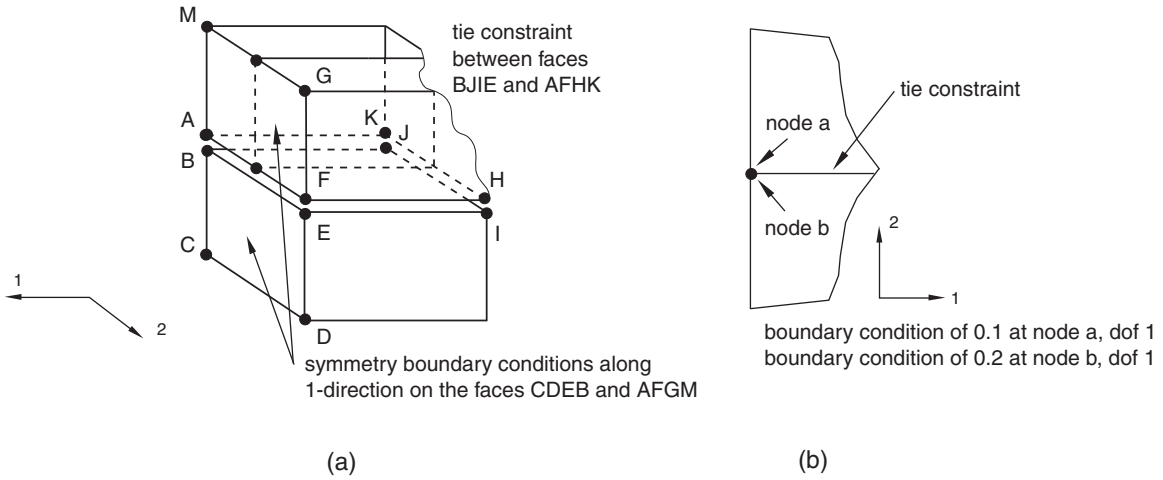


Figure 5: Overconstraints involving tie constraints and boundary conditions.

In the first case nodes A and B are constrained to move together by the tie constraint. The vertical symmetry boundary conditions will constrain the motion of both nodes in the horizontal direction, generating one redundant constraint. In the second case the two specified boundary conditions conflict, thus generating a conflicting constraint.

For every tie-dependent node with a boundary condition, Abaqus/Standard first determines which independent nodes are involved in the tie constraint (see *Mesh Tie Constraints*). If only one independent node is involved, Abaqus/Standard will transfer the boundary conditions from the dependent node to the independent node. If conflicting boundary conditions are detected at the independent node during the transferring process, the analysis is stopped and an error message is issued. If several independent nodes are involved, Abaqus/Standard checks if the specified boundary conditions at all the nodes involved in the constraint are identical. If no conflicts are identified, the boundary conditions at the independent node are redundant and, therefore, ignored. Otherwise, an error message is issued, and the analysis is stopped.

Rigid Body Constraints and Boundary Conditions

Combinations of rigid body constraints and boundary conditions can lead to overconstrained models when boundary conditions are specified at nodes other than the reference node (*Figure 6*). In *Figure 6*(a) boundary conditions are specified at several nodes belonging to the rigid body. In *Figure 6*(b) symmetry boundary conditions are specified on the flat surface of the rigid body, and the body is spun around an axis perpendicular to the symmetry plane at the reference node.

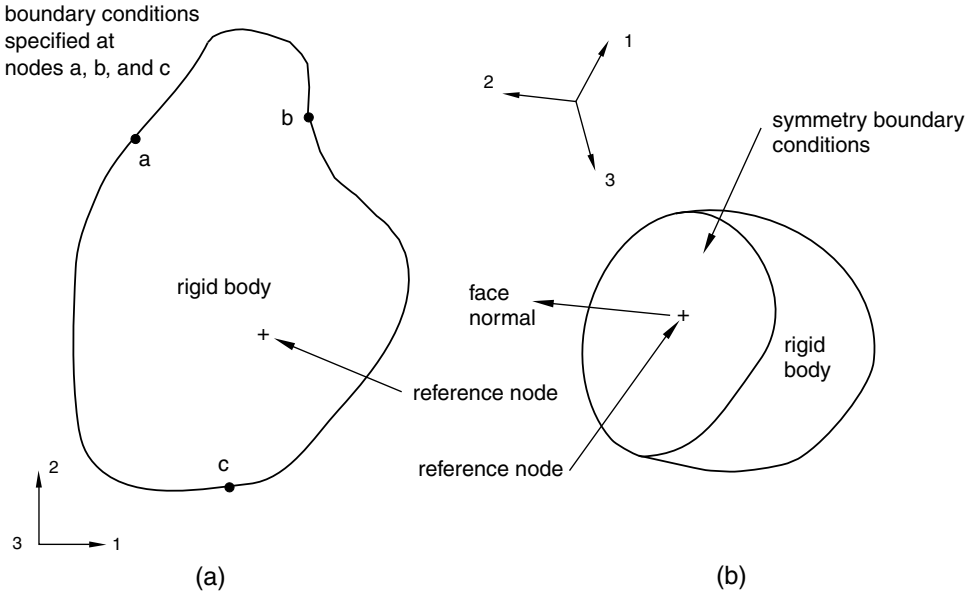


Figure 6: Overconstraints because of boundary conditions applied at rigid body nodes.

In case (a) if the specified boundary conditions are not consistent with the rigid constraint, the model will be inconsistently overconstrained. In case (b) if the reference node has the symmetry boundary conditions, there is no need to have symmetry boundary conditions at the nodes of the flat surface. Abaqus/Standard will attempt to remove all boundary conditions specified at the dependent nodes and redefine them at the reference node. To do so, the consistency of the boundary conditions specified at the dependent nodes is checked. If the boundary conditions are not identical, an error message is issued and the analysis is stopped (since otherwise the solution of a nonlinear system of equations would be required in the general case to assess whether the boundary conditions are consistent or not). Otherwise, Abaqus/Standard will try to merge the boundary conditions at the dependent nodes with those at the reference node by:

- checking the consistency of the overlapping boundary conditions;
- moving to the reference node any boundary conditions specified at the dependent nodes but not specified at the reference node; and
- applying additional zero rotational boundary conditions at the reference node to compensate for the removed displacement constraints from the dependent nodes.

To illustrate, refer to *Figure 6*(b): as the symmetry boundary conditions specified at the dependent nodes are consistent with each other, they are removed from the dependent nodes and applied to the reference node (boundary condition in the 2-direction). In addition, the symmetry constraints preclude rotations about the 1- and 3-directions; therefore, zero rotational boundary conditions are applied to the reference node about these axes.

Connector Elements and Rigid Bodies

In most cases detection and automatic resolution of redundant constraints involving connector elements cannot be done by simple inspection of the constraints involved. However, the examples shown in *Figure 7* are simple enough to be resolved automatically. It is assumed that the connector elements are connected to nodes on the rigid body whose rotational degrees of freedom are dependent on the rotation of the reference node. In *Figure 7*(a) the connector elements are assumed to enforce some kinematic constraints. They are redundant since the rigid body definition constrains the motion of all nodes to the motion of the rigid body's reference node. Abaqus/Standard automatically removes the connector elements from the model.

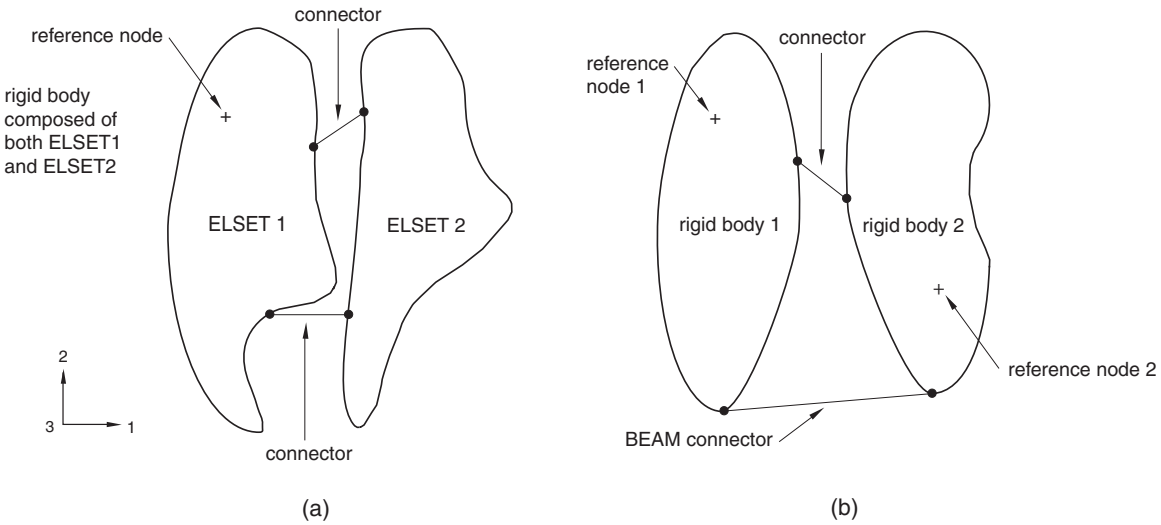


Figure 7: Redundant constraints involving rigid bodies and connector elements.

When connector elements are placed between two rigid bodies (as in *Figure 7*(b)), the model might be redundantly constrained. As shown in *Figure 7*(b), if a connector element of type BEAM (or WELD) is placed between two rigid bodies, the connection is rigid and any additional connector elements between the two rigid bodies are redundant. Abaqus/Standard will automatically remove these redundant connector elements.

When the ensemble of connector elements placed between two rigid bodies enforces more than the necessary translational and rotational constraints between the two rigid bodies, but none of the connectors is of type BEAM (or WELD), only warning messages are issued to signal the overconstraint situation. In these cases none of the connector elements can be eliminated automatically since the connection between the two rigid bodies might become underconstrained. To illustrate this situation, assume that in *Figure 7*(b) the two connectors were of type SLOT and TRANSLATOR. Thus, four translational constraints (in three dimensions) are enforced between the two rigid bodies, rendering the system overconstrained since only three translational constraints are needed to fully constrain the relative translation between the two bodies. However, if the SLOT were eliminated from the model, the model would become underconstrained and different from the original one. Only a warning message is issued in this case.

Coupling Constraints and Rigid Bodies

When all or some of the nodes involved in a kinematic coupling constraint belong to the same rigid body, the coupling constraint becomes redundant. The situation is illustrated in *Figure 8*. Node 101 is the reference node

for the coupling constraint involving nodes 1001–1005. At the same time nodes 1001–1003 are included in the rigid body definition with reference node 102.

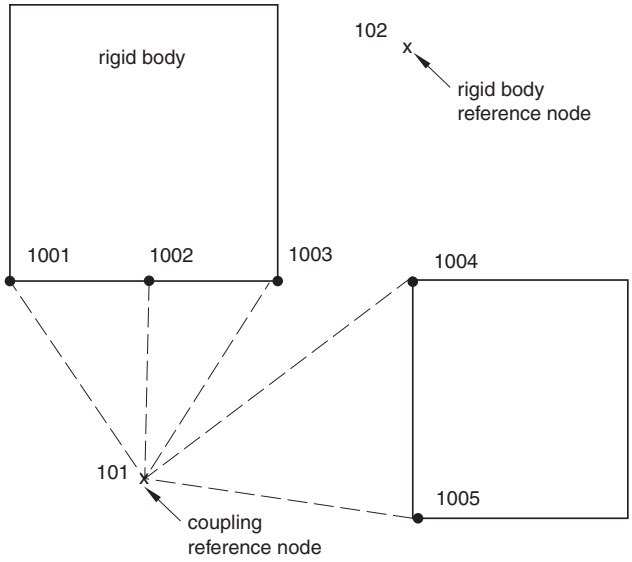


Figure 8: Redundant constraints involving coupling constraints and rigid bodies.

If the coupling constraint was defined as kinematic, it will not be enforced at nodes 1001–1003 to avoid overconstraining the model. The removed overconstraint might be inconsistent such as when incompatible boundary conditions are prescribed at the two reference nodes. However, the constraint will be enforced at nodes 1004 and 1005 since these nodes do not belong to the rigid body.

If a distributing coupling constraint was used instead, the model would not be overconstrained. However, if node 101 was added to the rigid body definition and nodes 1004 and 1005 were not included in the coupling constraint, the model would be overconstrained. Indeed, all nodes involved in the coupling constraint would be already constrained by the rigid body definition, making the coupling constraint redundant. To avoid the overconstraint, Abaqus/Standard will not enforce the coupling constraint in this case.

Coupling Constraints and Boundary Conditions

When boundary conditions are specified at all nodes involved in a distributing coupling constraint, the model might become overconstrained. Abaqus/Standard will issue a warning message outlining the cause of the potential overconstraint.

Spot Welds and Rigid Bodies

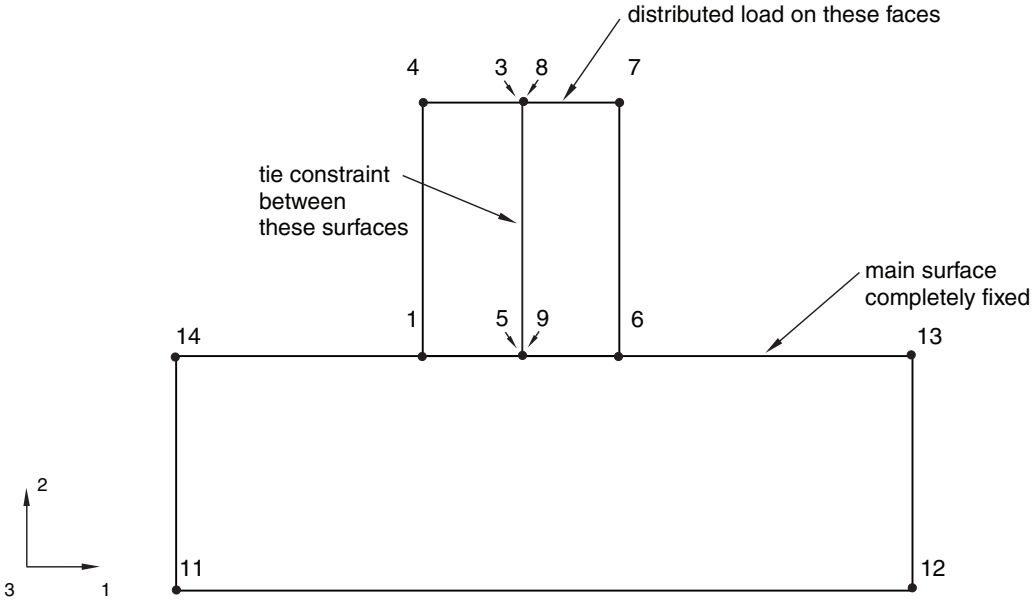
Potential overconstraints that might arise when a rigid body is involved in a mesh-independent spot weld definition are discussed in *Mesh-Independent Fasteners*.

Overconstraints Detected and Resolved during Analysis

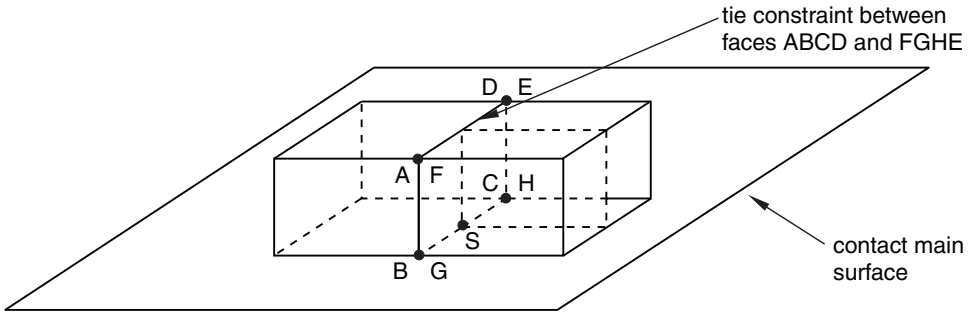
There are numerous situations when contact interactions in combination with other constraint types might lead to overconstraints. Since contact status typically changes during the analysis, it is not possible to detect redundant constraints associated with contact in the model preprocessor. Instead, these checks are performed during the analysis. Because of the complexities associated with contact interactions, only a limited number of redundant constraint cases are resolved automatically.

Contact Interactions and Tie Constraints

Redundant constraints are common in cases when secondary nodes used in surface-based tie constraints (*Mesh Tie Constraints*) are also secondary nodes in contact, as illustrated in *Figure 9*. In *Figure 9*(a) nodes 5 and 9 are connected with a tie constraint, and both are in contact with a main surface. Since the two nodes are tied together, one of the contact constraints is redundant. A similar situation is presented in *Figure 9*(b): two mismatched solid meshes are connected with a tie constraint, and contact is defined with a flat rigid surface. Node S is a dependent node in the tie constraint, so its motion is determined by that of nodes B and C. Therefore, any contact constraint applied at node S is redundant. Moreover, the contact constraints at nodes G and H are redundant, since the motion of these nodes is determined by nodes B and C, respectively.



(a)



(b)

Figure 9: Redundant constraints arising from contact interactions and tie constraints.

To eliminate these redundancies when all nodes involved in the tie constraint are in contact, Abaqus/Standard will automatically apply a tie-type constraint between the Lagrange multipliers associated with the contact constraint. The redundant contact constraint is eliminated. The contact pressure and the friction forces at the secondary node are recovered from the pressures and friction forces at the associated tie-independent nodes.

Deleting Contact Elements to Remove Overconstraints

Instead of letting Abaqus remove overconstraints by tying Lagrange multipliers, you can apply constraint controls that delete the contact elements associated with tied secondary nodes. If you use this technique, contact-related output is not available for the tied secondary nodes.

Input File Usage: **CONSTRAINT CONTROLS*, DELETE SECONDARY

Contact Interactions and Prescribed Boundary Conditions

Contact interactions and prescribed boundary conditions might lead to redundant constraints if either normal contact with the default "hard contact" formulation (*Contact Pressure-Overclosure Relationships*) or frictional contact with the Lagrange multiplier formulation (see *Frictional Behavior*) is invoked. Abaqus/Standard attempts to resolve these types of redundant constraints for contact pairs involving rigid surfaces.

Checks Related to Normal Contact Interactions

In *Figure 10* the fixed analytical rigid main surface is in contact with a secondary node that has a fixed boundary condition specified in the direction normal to the contact surface. If during a particular increment in the analysis the node is in contact, the contact constraint is redundant and will not be enforced during that increment. If the boundary condition at the secondary node is in conflict with the boundary conditions at the rigid surface's reference node, an error message is issued and the analysis is stopped.

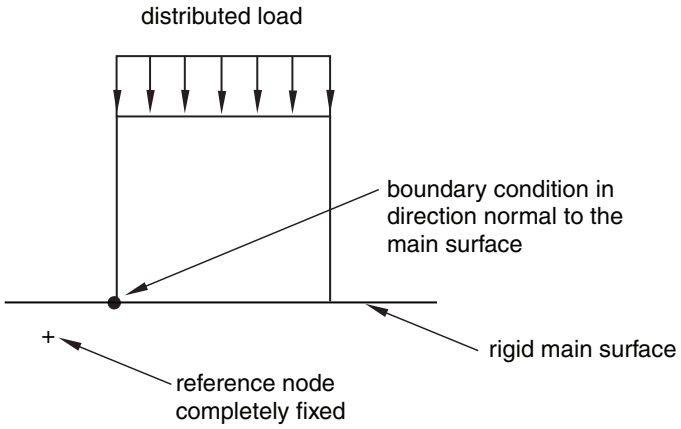


Figure 10: Overconstraints involving normal contact interactions and boundary conditions.

The contact and boundary conditions related to overconstraints are removed automatically only if the main surface is defined as an analytical rigid surface. In all other cases, if an overconstraint occurs during the analysis, a zero pivot message is issued by the equation solver (see below) and the chains of constraints responsible for the overconstraint are clearly outlined.

Checks Related to Lagrange Friction

A common redundant constraint case is depicted in *Figure 11*. The symmetry boundary conditions combined with the Lagrange friction are redundant.

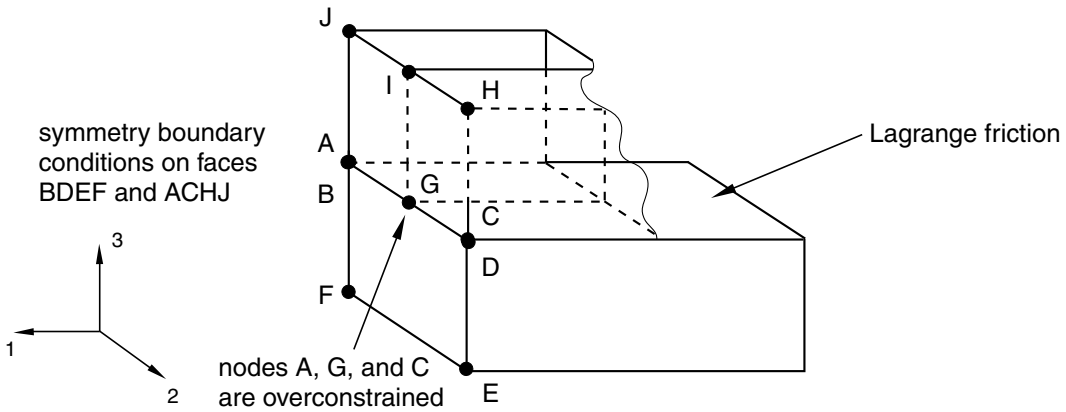


Figure 11: Lagrange friction and boundary conditions.

The secondary node is in contact and the tangent to the surface is in approximately the same direction as the specified boundary condition at the secondary node. To avoid redundancy, at this node Abaqus/Standard will switch from the Lagrange friction formulation to the default penalty formulation (*Frictional Behavior*) if the motion of the main nodes is prescribed in the tangent direction.

Overconstraints Detected in the Equation Solver

All overconstraints that cannot be identified and resolved during preprocessing or during the analysis need to be detected by the equation solver. Examples include models with contact interactions where secondary nodes are driven by specified boundary conditions into partially fixed rigid surfaces; contact with multiple main surfaces; closed-loop and multiple-loop mechanisms in which rigid bodies are connected by connector elements; and many more. By default, equation solver overconstraint checks are performed continuously during the analysis.

Abaqus/Standard will not resolve overconstraints detected by the equation solver. Instead, detailed messages with information regarding the kinematic constraints involved in the overconstraint will be issued. The message first identifies the nodes involved in either a consistent or an inconsistent overconstraint by using zero pivot information from the Gauss elimination in the solver (*Direct Linear Equation Solver*). A detailed message containing constraint information is then issued.

The 4-bar mechanism shown in *Figure 12* illustrates this strategy.

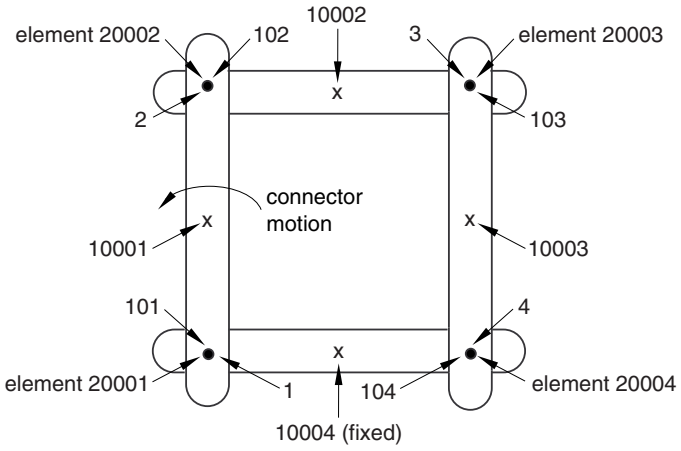


Figure 12: Hard-to-detect redundant constraints.

Four three-dimensional rigid bodies are defined as follows: the rigid body with reference node 10001 includes nodes 2 and 101; the rigid body with reference node 10002 includes nodes 3 and 102; the rigid body with reference node 10003 includes nodes 4 and 103; and the rigid body with reference node 10004 includes nodes 1 and 104. The four rigid bodies are connected with four JOIN and REVOLUTE combination connector elements defined as follows: element 20001 between nodes 1 and 101; element 20002 between nodes 2 and 102; element 20003 between nodes 3 and 103; and element 20004 between nodes 4 and 104. Each connector element enforces three translation and two rotation constraints (*About Connectors*), and all four revolute axis directions are parallel. The bottom rigid body (with reference node 10004) is fixed. The motion of the bottom left REVOLUTE connector (element 20001) is prescribed to rotate the mechanism.

When Abaqus/Standard attempts to find a solution for this model, three zero pivots are identified in the first increment of the analysis suggesting that there are three constraints too many in the model. Eventually, one would have to remove three constraints to render the model properly constrained. In this simple example a count of the degrees of freedom and constraints confirms the number of overconstraints, as follows. There are four rigid bodies in the model, with a total of 24 degrees of freedom. The reference node 10004 is completely fixed with a boundary condition, constraining six degrees of freedom; and the prescribed connector motion enforces one rotational constraint, constraining one degree of freedom. Hence, there are 17 degrees of freedom remaining. Each of the four connector elements enforces five constraints, for a total of 20 constraints. Thus, there are three constraints too many in the model, which matches the number of zero pivots identified by the equation solver. To help you identify the constraints that should be removed, the following message is produced in the message (`.msg`) file outlining the chains of constraints that generated the overconstraint:

```
 ***WARNING: SOLVER PROBLEM. ZERO PIVOT WHEN PROCESSING ELEMENT 20004
             INTERNAL NODE 1 D.O.F. 4

OVERCONSTRAINT CHECKS:  An overconstraint was detected at one of the
Lagrange multipliers associated with element 20004. There are
multiple constraints applied directly or chained constraints that
are applied indirectly to this element. The following is a list of
nodes and chained constraints between these nodes that most likely
lead to the detected overconstraint.

LAGRANGE MULTIPLIER: 4 <-> 104: connector element 20004 type
                     JOIN REVOLUTE constraining 3 translations
                     and  2 rotations
..4 -> 10003: *RIGID BODY (or *COUPLING-KINEMATIC)
....10003 -> 103: *RIGID BODY (or *COUPLING-KINEMATIC)
......103 -> 3: connector element 20003 type JOIN REVOLUTE
                constraining 3 translations  and  2 rotations
........3 -> 10002: *RIGID BODY (or *COUPLING-KINEMATIC)
..........10002 -> 102: *RIGID BODY (or *COUPLING-KINEMATIC)
............102 -> 2: connector element 20002 type JOIN REVOLUTE
                      constraining 3 translations and 2 rotations
..............2 -> 10001: *RIGID BODY (or *COUPLING-KINEMATIC)
................10001 -> 101: *RIGID BODY (or *COUPLING-KINEMATIC)
..................101 -> 1: connector element 20001 type
                            JOIN REVOLUTE constraining  3
                            translations  and 2 rotations
....................1 -> 10004: *RIGID BODY (or *COUPLING-KINEMATIC)
......................10004 -> *BOUNDARY in degrees of freedom
                                 1  2  3  4  5  6
......................10004 -> 104: *RIGID BODY
                                      (or *COUPLING-KINEMATIC)
....................1 -> 101: connector element 20001 with
                              *CONNECTOR MOTION in components  4

 Please analyze these constraint loops and remove unnecessary
 constraints.
```

First, the message identifies the user-defined or, in this case, the internally defined (Lagrange multiplier) node at which a zero pivot was identified. A typical line in this output issues information related to one constraint. For example, the first line in this output

```
LAGRANGE MULTIPLIER: 4 <-> 104: connector element 20004 type
                     JOIN REVOLUTE constraining 3 translations
                     and  2 rotations
```

informs you that the Lagrange multiplier on which the zero pivot occurs enforces one of the five constraints (JOIN and REVOLUTE) associated with connector element 20004 between user-defined nodes 4 and 104. Each of the subsequent lines conveys information related to one constraint in the chains of constraints originating at the zero pivot node or in chains adjacent to them. For example, the line

```
....10003 -> 103: *RIGID BODY (or *COUPLING - KINEMATIC)
```

informs you that there is a rigid body constraint between nodes 10003 and 103, while the line

```
.....................10004 -> *BOUNDARY in degrees of freedom
                               1  2  3  4  5  6
```

states that there is a boundary condition constraint fixing degrees of freedom 1 through 6 at node 10004.

Indentation levels (the dots in front of the node numbers) identify links in a chain of constraints. Each time a constraint is found to link another node in a particular chain, the indentation is increased by two dots and the constraint information is printed out. For example, starting from the top of the message, the Lagrange multiplier is connected to node 4, node 4 is connected to node 10003, node 10003 is connected to node 103, and so on. When the indentation on a certain line is less than or equal to the indentation on the previous line, a chain of constraints has ended on the previous line. For example, a chain has ended on the line

```
.....................10004 -> *BOUNDARY in degrees of freedom
                               1  2  3  4  5  6
```

since the next line has equal indentation.

Three chains of constraints (in correspondence with the three zero pivots that were found) that most likely generated the overconstraint can be identified in the model above. Starting from the top, one can first identify a chain of constraints that terminates in a boundary condition (ground):

```
Lagrange multiplier: 4 -> 10003 -> 103 -> 3 -> 10002 -> 2 ->
10001 -> 101 -> 1 -> 10004 -> *BOUNDARY
```

Since the indentation of the two lines starting with node 10004 is the same, one should expect another chain of constraints to include the constraint output on the second of the two lines. Indeed, one can identify a closed loop of constraints:

```
Lagrange multiplier : 4-> 10003 -> 103 -> 3 -> 10002 -> 2 ->
10001 -> 101 -> 1 -> 10004 -> 104 <-> 4
```

Finally, since the two lines starting with node 1 have the same indentation, one expects that a separate chain of constraints will include the last line in the output. A third (closed) loop

```
101 -> 1 -> 101
```

is identified.

If the chains of constraints terminate in a free end (not ending in a constraint), the chain does not have any contribution in generating the overconstraint. There are no such chains in this example.

Correcting an Overconstrained Model

A node set containing all the nodes in the chains of constraints associated with a particular zero pivot is generated automatically and can be displayed in the Visualization module of Abaqus/CAE.

There is no unique way to remove the overconstraints in this model. For example, if one JOIN and REVOLUTE (five constraints) combination is replaced with a SLOT connector element, which enforces only the two translation constraints in the plane of the mechanism, there are no redundancies. Alternatively, you could remove the REVOLUTE from one of the connector elements and also use a SLOT connection instead of a JOIN in one of the other connector elements.

Another alternative is to relax some of the constraints. In the example outlined here, an elastic body could replace one or more of the rigid bodies. You could also relax the Lagrange multiplier-based constraints (e.g., JOIN or REVOLUTE) by using CARTESIAN and CARDAN connection types with appropriate elastic stiffnesses (see *Connector Behavior*).

After analyzing the chains of constraints, you have to decide which constraints have to be removed to render the model properly constrained and also best fit the modeling goals. For this example the three constraints cannot be removed randomly. Removing any three combinations of the six boundary conditions, for example, would make the problem worse: the model is still overconstrained, and three rigid body modes have been added to the model. Moreover, you should remove the constraints that do not affect the kinematics of the model. For example, you cannot completely remove a JOIN connection from any of the connector elements since the model would be different from that originally intended.

Controlling the Overconstraint Checks

By default, Abaqus/Standard will attempt to remove as many redundant constraints as possible, as discussed in the sections above. When it is not possible to remove a redundant constraint or an inconsistent overconstraint is detected, a detailed message is issued identifying the constraints contributing to the overconstraint. You can modify this default behavior by prescribing constraint controls for the model or the step.

Overconstraints might produce damaging and unpredictable behavior. Therefore, it is strongly recommended that overconstraint checking be used in both the preprocessor and during the analysis at least during the first running of a model. Furthermore, it is recommended that the original model be changed to correct any overconstraints identified by Abaqus/Standard. Only after establishing confidence that the model is free of overconstraints should constraint checks be turned off. The only advantage of turning off the constraint checks is a minor speedup of the analysis.

Bypassing the Overconstraint Checks

The overconstraint checks performed during input file preprocessing and during the analysis can be bypassed. Bypassing these checks is not recommended, as it might allow a model with overconstraints to enter into the analysis code. Bypassing the overconstraint checks is not step dependent; i.e., the setting is defined as model data and affects the entire analysis.

Input File Usage: **CONSTRAINT CONTROLS*, NO CHECKS

Preventing Automatic Redundant Constraint Resolution

Automatic model modifications in the model preprocessor can be prevented. In this case Abaqus/Standard will still perform overconstraint checks, but no automatic redundant constraint resolution will be performed; only

appropriate error messages will be issued. Preventing constraint resolution is not step dependent; i.e., the setting is defined as model data and affects the entire analysis.

Input File Usage: **CONSTRAINT CONTROLS*, NO CHANGES

Changing the Frequency of the Overconstraint Checks

By default, the overconstraint checks are performed at every increment during the analysis. You can modify the frequency of these checks (in increments) for each step in the analysis. If the frequency is set equal to zero, no overconstraint checks are performed during that analysis step. The frequency specification is maintained in subsequent steps until the value is reset.

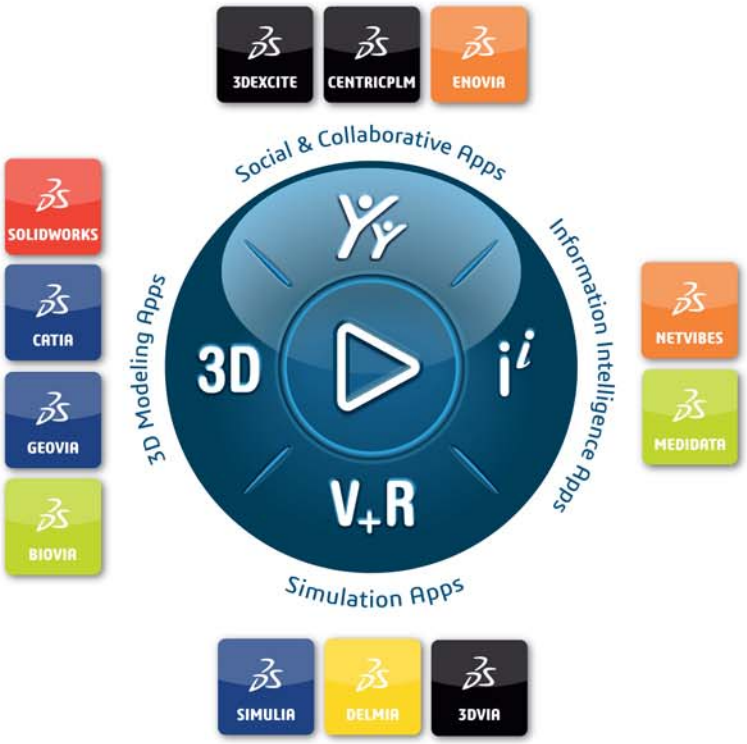
Input File Usage: **CONSTRAINT CONTROLS*, CHECK FREQUENCY=*n*

Stopping the Analysis When Overconstraints Are Detected

By default, the analysis continues even though an overconstraint is detected. This behavior can be changed on a step-dependent basis. The analysis can be stopped the first time an overconstraint is detected in a step, or it can be stopped only if a converged solution is obtained despite the fact that overconstraints exist. This setting is maintained in subsequent steps until it is reset.

Input File Usage: Use one of the following options:

**CONSTRAINT CONTROLS*, TERMINATE ANALYSIS=FIRST OCCURRENCE
**CONSTRAINT CONTROLS*, TERMINATE ANALYSIS=CONVERGED



Our **3D**EXPERIENCE® platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes is a catalyst for human progress. We provide business and people with collaborative virtual environments to imagine sustainable innovations. By creating virtual twin experiences of the real world with our **3DEXPERIENCE** platform and applications, our customers can redefine the creation, production and life-cycle-management processes of their offer and thus have a meaningful impact to make the world more sustainable. The beauty of the Experience Economy is that it is a human-centered economy for the benefit of all – consumers, patients and citizens.

Dassault Systèmes brings value to more than 300,000 customers of all sizes, in all industries, in more than 150 countries. For more information, visit **www.3ds.com**.

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes
17F, Foxconn Building,
No. 1366, Lujiazui Ring Road
Pilot Free Trade Zone, Shanghai 200120
China

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

DASSAULT SYSTEMES

©2025 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries.