OpenSeesPy Documentation

Release 1.0.0b1

Minjie Zhu

Oct 26, 2018

Installation

Note: If you use OpenSeesPy, I would like very much to hear from you. A short email to zhum@oregonstate.edu describing who you are and how you use OpenSeesPy will mean a lot to me. I can justify spending time on improvements that I hope will benefit you.

Note: The OpenSeesPy library is still in beta version. Please send any questions to zhum@oregonstate.edu or github issues.

You are very welcome to contribute to OpenSeesPy with new command documents and examples by sending pull requests through github pulls.

OpenSeesPy is a Python 3 interpreter of OpenSees. A minimum script is shown below:

```
##########################################
# If installed directly with library files
import sys

# for Linux
sys.path.append('/path/to/OpenSeesPy')

# for Windows
sys.path.append('C:/path/to/OpenSeesPy')

from opensees import *
###########################################

###########################################
# If installed with PyPi

from openseespy.opensees import *
###########################################

# Using OpenSees ...

# wipe before exiting
wipe()
```

Most of OpenSeesPy commands have the same syntax and arguments as the OpenSees Tcl commands. The conversion from Tcl to Python is easy and straightforward as demonstrated with commands below.

CHAPTER 1

Author

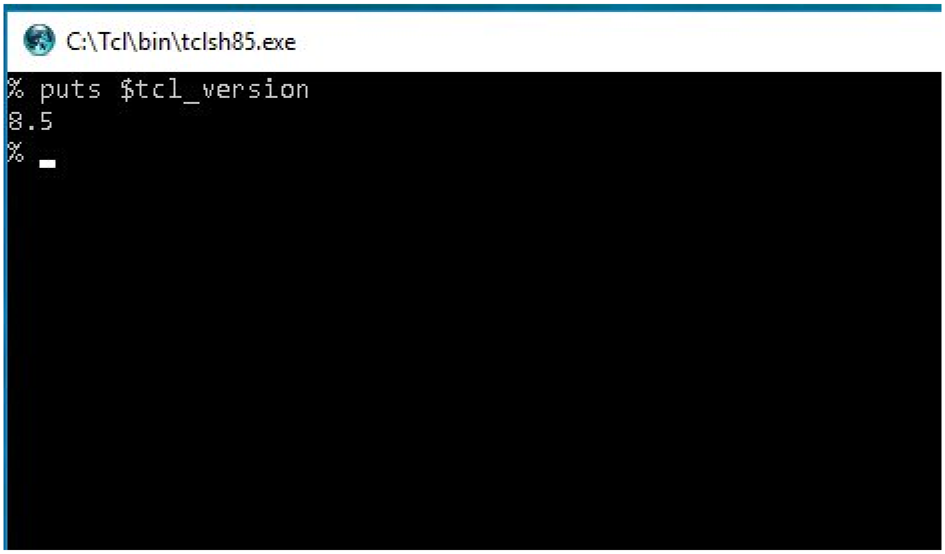
Minjie Zhu **<zhum@oregonstate.edu>**

Faculty Research Assistant
Civil and Construction Engineering
Oregon State University

1.1 Install OpenSeesPy in Windows:

1.1.1 Install ActiveStateTcl 8.5

Check the Tcl version

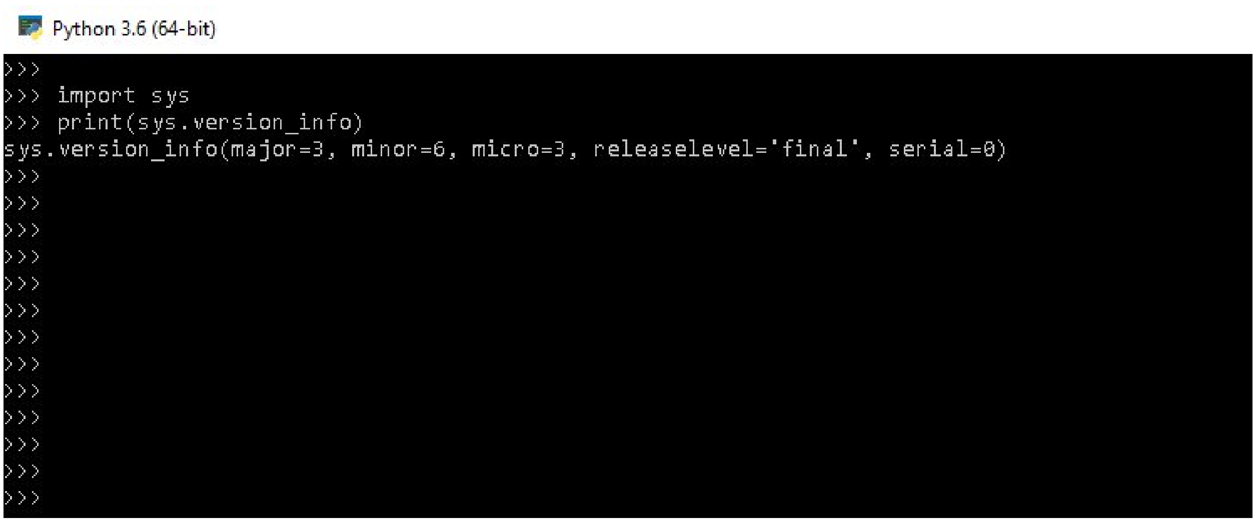


1.1.2 Install Python 3.6 Windows or Anaconda 5.0 Windows

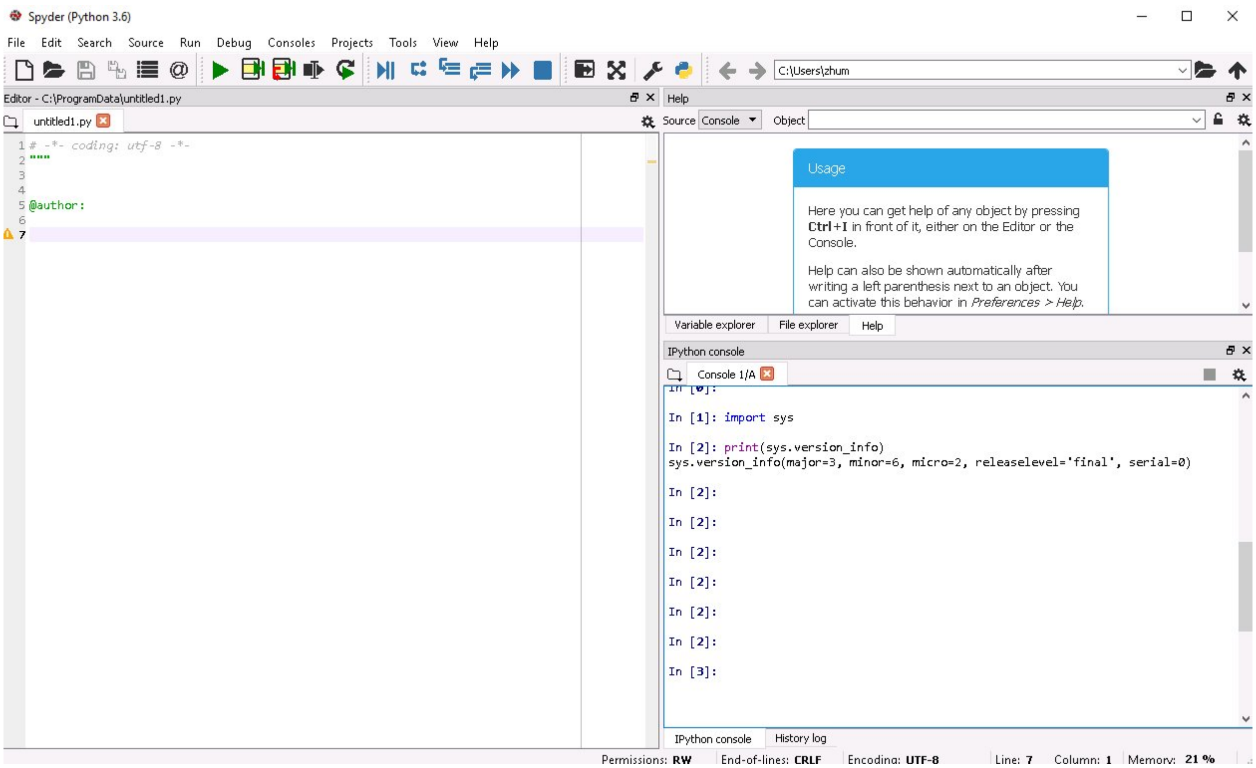
Note: 64bit Python 3.6 version is required!

Both work, but Anaconda comes with many libraries and editors

Check the python version



An anaconda environment



1.1.3 Download OpenSeesPy Windows Library

Two files, `opensees.pyd` and `LICENSE.rst`, are included in the zip file. Put the library file `opensees.pyd` in a directoy, which path should be copied to

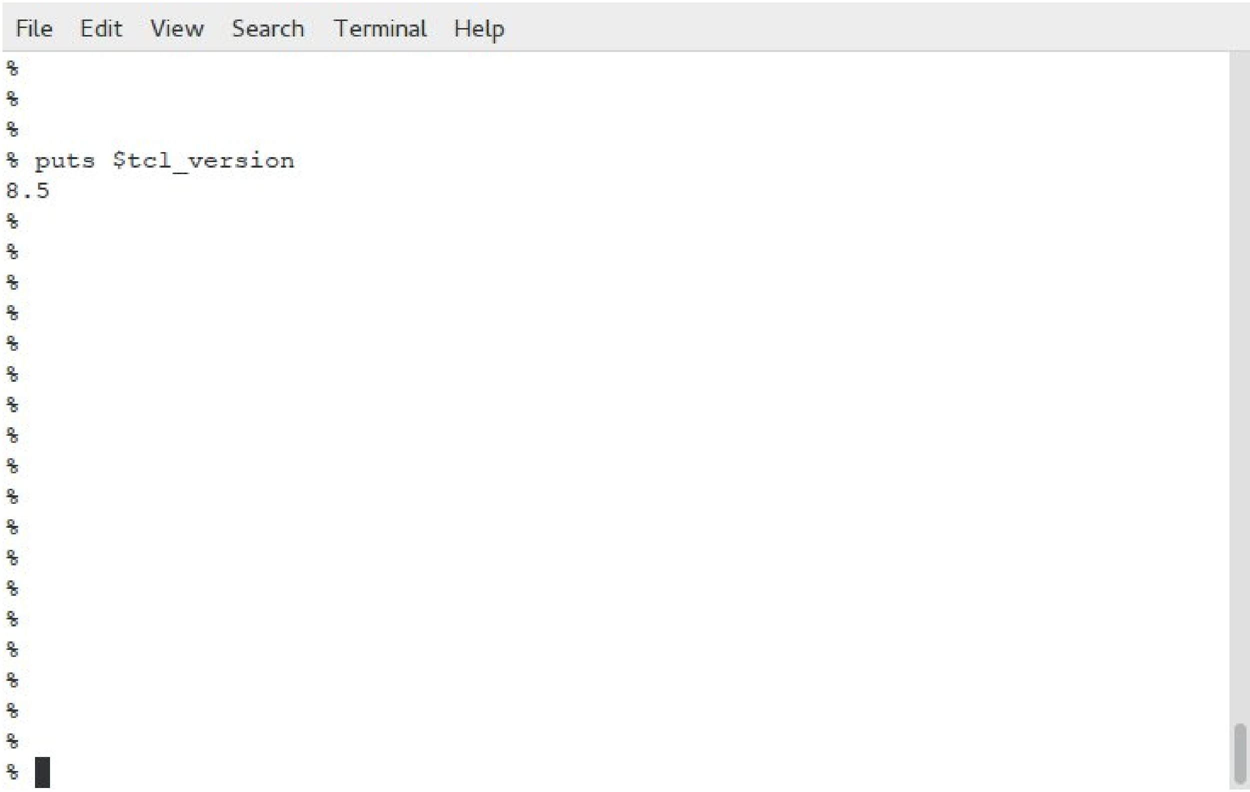
```
sys.path.append('C:/path/to/OpenSeesPy')
```

1.2 Install OpenSeesPy in Linux:

1.2.1 Install Tcl 8.5

Usually, the Tcl 8.5 is already installed in a Linux system.

Check the Tcl version

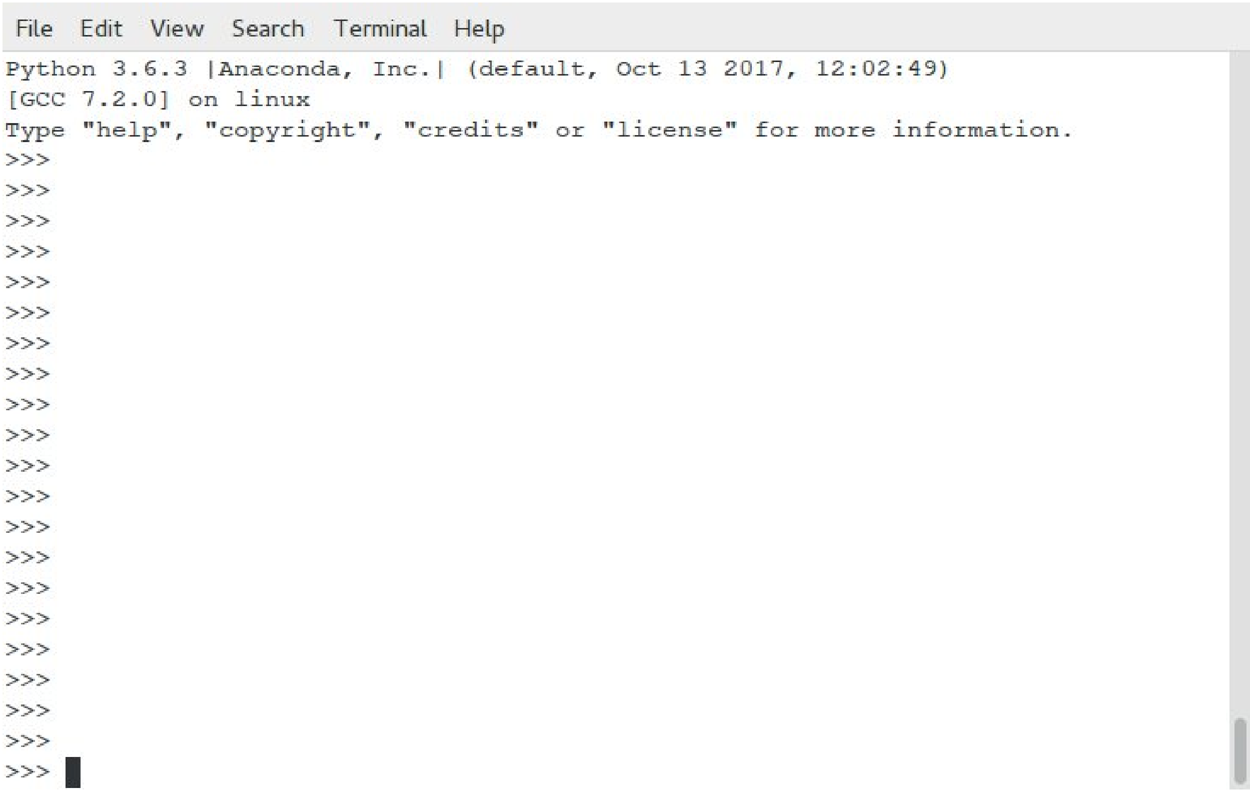


1.2.2 Install Python 3.6 Linux or Anaconda 5.0 Linux

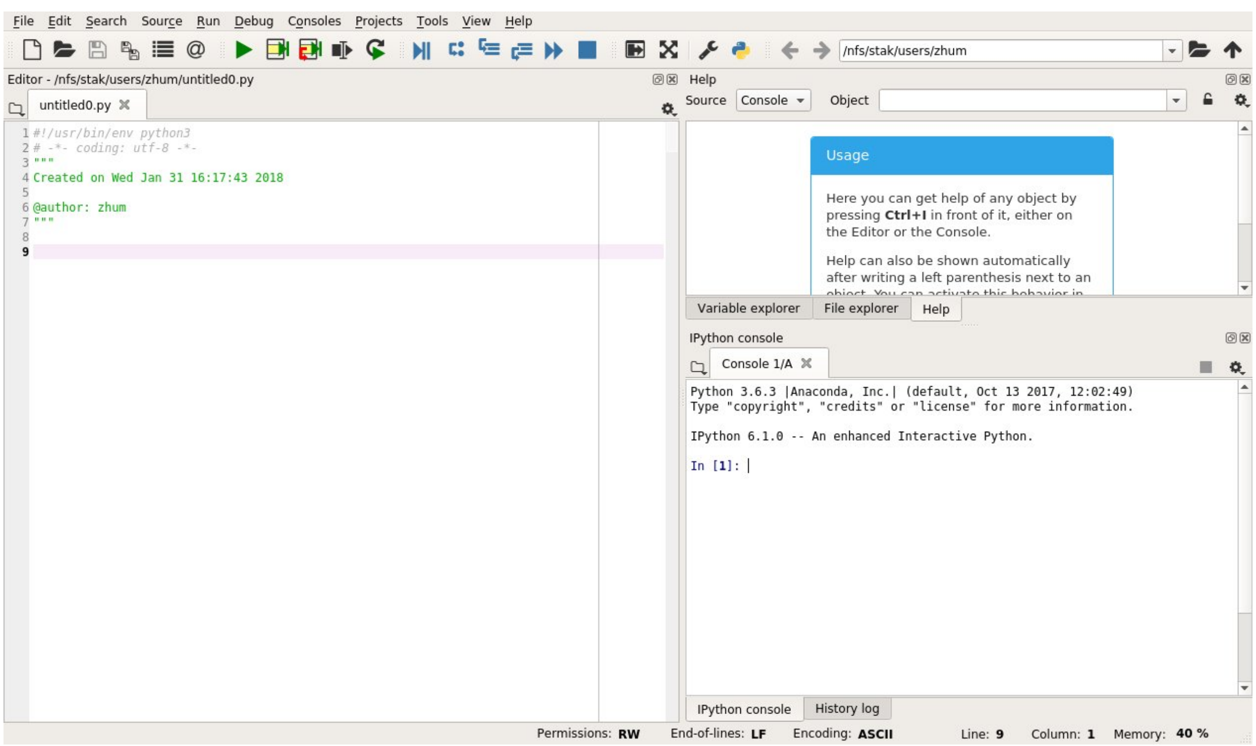
Note: 64bit Python 3.6 version is required!

Both work, but Anaconda comes with many libraries and editors

Check the python version



An anaconda environment



1.2.3 Download OpenSeesPy Linux Library

Two files, `opensees.so` and `LICENSE.rst`, are included in the zip file. Put the library file `opensees.so` in a directoy, which path should be copied to

```
sys.path.append('/path/to/OpenSeesPy')
```

1.3 Install OpenSeesPy though PyPi

You should have Python 3.6 or higher installed already.

Use following command to install

```
pip install openseespy
```

and import OpenSeesPy as

```
import openseespy.opensees as ops
```

1.4 Model Commands

The model or domain in OpenSees is a collection (an aggregation in object-oriented terms) of elements, nodes, single- and multi-point constraints and load patterns. It is the aggregation of these components which define the type of model that is being analyzed.

1.4.1 model command

model (*'basic'*, *'-ndm'*, *ndm*, *'-ndf'*, *ndf=ndm*(ndm+1)/2*)
Set the default model dimensions and number of dofs.

`ndm` (int)	number of dimensions (1,2,3)
`ndf` (int)	number of dofs (optional)

1.4.2 element commands

element (*eleType*, *eleTag*, **eleNodes*, **eleArgs*)
Create a OpenSees element.

`eleType` (str)	element type
`eleTag` (int)	element tag.
`eleNodes` (list (int))	a list of element nodes, must be preceded with `*`.
`eleArgs` (list)	a list of element arguments, must be preceded with `*`.

For example,

```
eleType = 'truss'
eleTag = 1
eleNodes = [iNode, jNode]
eleArgs = [A, matTag]
element(eleType, eleTag, *eleNodes, *eleArgs)
```

The following contain information about available `eleType`:

zeroLength Element

element (*'zeroLength'*, *eleTag*, **eleNodes*, *'-mat'*, **matTags*, *'-dir'*, **dirs*[, *'-doRayleigh'*, *rFlag=0*][, *'-orient'*, **vecx*, **vecyp*])

This command is used to construct a zeroLength element object, which is defined by two nodes at the same location. The nodes are connected by multiple UniaxialMaterial objects to represent the force-deformation relationship for the element.

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`matTags` (list (int))	a list of tags associated with previously-defined UniaxialMaterials
`dirs` (list (int))	a list of material directions: • 1,2,3 - translation along local x,y,z axes, respectively; • 4,5,6 - rotation about local x,y,z axes, respectively
`vecx` (list (float))	a list of vector components in global coordinates defining local x-axis (optional)
`vecyp` (list (float))	a list of vector components in global coordinates defining vector yp which lies in the local x-y plane for the element. (optional)
`rFlag` (float)	optional, default = 0 • `rFlag` = 0 NO RAYLEIGH DAMPING (default) • `rFlag` = 1 include rayleigh damping

Note: If the optional orientation vectors are not specified, the local element axes coincide with the global axes. Otherwise the local z-axis is defined by the cross product between the vectors x and yp vectors specified on the command line.

See also:

Notes

zeroLengthND Element

element (*'zeroLengthND'*, *eleTag*, **eleNodes*, *matTag*[, *uniTag*][, *'-orient'*, **vecx*, *vecyp*])

This command is used to construct a zeroLengthND element object, which is defined by two nodes at the same location. The nodes are connected by a single NDMaterial object to represent the force-deformation relationship for the element.

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`matTag` (int)	tag associated with previously-defined ndMaterial object
`uniTag` (int)	tag associated with previously-defined UniaxialMaterial object which may be used to represent uncoupled behavior orthogonal to the plane of the NDmaterial response. SEE NOTES 2 and 3.
`vecx` (list (float))	a list of vector components in global coordinates defining local x-axis (optional)
`vecyp` (list (float))	a list of vector components in global coordinates defining vector yp which lies in the local x-y plane for the element. (optional)

Note:

1. The zeroLengthND element only represents translational response between its nodes
2. If the NDMaterial object is of order two, the response lies in the element local x-y plane and the UniaxialMaterial object may be used to represent the uncoupled behavior orthogonal to this plane, i.e. along the local z-axis.
3. If the NDMaterial object is of order three, the response is along each of the element local exes.
4. If the optional orientation vectors are not specified, the local element axes coincide with the global axes. Otherwise the local z-axis is defined by the cross product between the vectors x and yp vectors specified on the command line.
5. The valid queries to a zero-length element when creating an ElementRecorder object are 'force', 'deformation', and 'material matArg1 matArg2 . . . '

See also:

Notes

zeroLengthSection Element

element (*'zeroLengthSection'*, *eleTag*, **eleNodes*, *secTag*[, *'-orient'*, **vecx*, **vecyp*][, *'-doRayleigh'*, *rFlag*])

This command is used to construct a zero length element object, which is defined by two nodes at the same location. The nodes are connected by a single section object to represent the force-deformation relationship for the element.

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`secTag` (int)	tag associated with previously-defined Section object
`vecx` (list (float))	a list of vector components in global coordinates defining local x-axis (optional)
`vecyp` (list (float))	a list of vector components in global coordinates defining vector yp which lies in the local x-y plane for the element. (optional)
`rFlag` (float)	optional, default = 0 • `rFlag` = 0 NO RAYLEIGH DAMPING (default) • `rFlag` = 1 include rayleigh damping

See also:

Notes

CoupledZeroLength Element

element (*'CoupledZeroLength'*, *eleTag*, **eleNodes*, *dirn1*, *dirn2*, *matTag*[, *rFlag=1*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`matTag` (float)	tags associated with previously-defined UniaxialMaterial
`dir1 dir2` (int)	the two directions, 1 through ndof.
`rFlag` (float)	optional, default = 0 • `rFlag` = 0 NO RAYLEIGH DAMPING (default) • `rFlag` = 1 include rayleigh damping

See also:

Notes

zeroLengthContact Element

element (*'zeroLengthContact2D'*, *eleTag*, **eleNodes*, *Kn*, *Kt*, *mu*, *'-normal'*, *Nx*, *Ny*)

This command is used to construct a zeroLengthContact2D element, which is Node-to-node frictional contact element used in two dimensional analysis and three dimensional analysis:

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of a slave and a master nodes
`Kn` (float)	Penalty in normal direction
`Kt` (float)	Penalty in tangential direction
`mu` (float)	friction coefficient

element (*'zeroLengthContact3D'*, *eleTag*, **eleNodes*, *Kn*, *Kt*, *mu*, *c*, *dir*)

This command is used to construct a zeroLengthContact3D element, which is Node-to-node frictional contact element used in two dimensional analysis and three dimensional analysis:

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of a slave and a master nodes
`Kn` (float)	Penalty in normal direction
`Kt` (float)	Penalty in tangential direction
`mu` (float)	friction coefficient
`c` (float)	cohesion (not available in 2D)
`dir` (int)	Direction flag of the contact plane (3D), it can be: • 1 Out normal of the master plane pointing to +X direction • 2 Out normal of the master plane pointing to +Y direction • 3 Out normal of the master plane pointing to +Z direction

See also:

Notes

zeroLengthContactNTS2D

element (*'zeroLengthContactNTS2D'*, *eleTag*, *'-sNdNum'*, *sNdNum*, *'-mNdNum'*, *mNdNum*, *'-Nodes'*, **Nodes*, *Kn*, *kt*, *phi*)

`eleTag` (int)	unique element object tag
`sNdNum` (int)	Number of Slave Nodes
`mNdNum` (int)	Number of Master nodes
`Nodes` (list (int))	Slave and master node tags respectively
`Kn` (float)	Penalty in normal direction
`Kt` (float)	Penalty in tangential direction
`phi` (float)	Friction angle in degrees

Note:

1. The contact element is node-to-segment (NTS) contact. The relation follows Mohr-Coulomb frictional law: $T = N \times \tan(\phi)$, where T is the tangential force, N is normal force across the interface and ϕ is friction angle.
2. For 2D contact, slave nodes and master nodes must be 2 DOF and notice that the slave and master nodes must be entered in counterclockwise order.
3. The resulting tangent from the contact element is non-symmetric. Switch to the non-symmetric matrix solver if convergence problem is experienced.
4. As opposed to node-to-node contact, predefined normal vector for node-to-segment (NTS) element is not required because contact normal will be calculated automatically at each step.
5. contact element is implemented to handle large deformations.

See also:

Notes

zeroLengthInterface2D

element (*'zeroLengthInterface2D'*, *eleTag*, *'-sNdNum'*, *sNdNum*, *'-mNdNum'*, *mNdNum*, *'-dof'*, *sdof*, *mdof*, *'-Nodes'*, **Nodes*, *Kn*, *kt*, *phi*)

`eleTag` (int)	unique element object tag
`sNdNum` (int)	Number of Slave Nodes
`mNdNum` (int)	Number of Master nodes
`sdof`, `mdof` (int)	Slave and Master degree of freedom
`Nodes` (list (int))	Slave and master node tags respectively
`Kn` (float)	Penalty in normal direction
`Kt` (float)	Penalty in tangential direction
`phi` (float)	Friction angle in degrees

Note:

1. The contact element is node-to-segment (NTS) contact. The relation follows Mohr-Coulomb frictional law: $T = N \times \tan(\phi)$, where T is the tangential force, N is normal force across the interface and ϕ is friction angle.
2. For 2D contact, slave nodes and master nodes must be 2 DOF and notice that the slave and master nodes must be entered in counterclockwise order.
3. The resulting tangent from the contact element is non-symmetric. Switch to the non-symmetric matrix solver if convergence problem is experienced.
4. As opposed to node-to-node contact, predefined normal vector for node-to-segment (NTS) element is not required because contact normal will be calculated automatically at each step.
5. contact element is implemented to handle large deformations.

See also:

Notes

zeroLengthImpact3D

element (*'zeroLengthImpact3D'*, *eleTag*, **eleNodes*, *direction*, *initGap*, *frictionRatio*, *Kt*, *Kn*, *Kn2*, *Delta_y*, *cohesion*)

This command constructs a node-to-node zero-length contact element in 3D space to simulate the impact/pounding and friction phenomena.

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of a slave and a master nodes
`direction` (int)	• `1` if out-normal vector of master plane points to +X direction • `2` if out-normal vector of master plane points to +Y direction • `3` if out-normal vector of master plane points to +Z direction
`initGap` (float)	Initial gap between master plane and slave plane
`frictionRatio` (float)	Friction ratio in two tangential directions (parallel to master and slave planes)
`Kt` (float)	Penalty in two tangential directions
`Kn` (float)	Penalty in normal direction (normal to master and slave planes)
`Kn2` (float)	Penalty in normal direction after yielding based on Hertz impact model
`Delta_y` (float)	Yield deformation based on Hertz impact model
`cohesion` (float)	Cohesion, if no cohesion, it is zero

Note:

1. This element has been developed on top of the "zeroLengthContact3D". All the notes available in "zeroLength-Contact3D" wiki page would apply to this element as well. It includes the definition of master and slave nodes, the number of degrees of freedom in the domain, etc.
2. Regarding the number of degrees of freedom (DOF), the end nodes of this element should be defined in 3DOF domain. For getting information on how to use 3DOF and 6DOF domain together, please refer to OpenSees documentation and forums or see the zip file provided in the EXAMPLES section below.
3. This element adds the capabilities of "ImpactMaterial" to "zeroLengthContact3D."
4. For simulating a surface-to-surface contact, the element can be defined for connecting the nodes on slave surface to the nodes on master surface.
5. The element was found to be fast-converging and eliminating the need for extra elements and nodes in the modeling process.

See also:

Notes

Truss Element

This command is used to construct a truss element object. There are two ways to construct a truss element object:

element (*'Truss'*, *eleTag*, **eleNodes*, *A*, *matTag*[, *'-rho'*, *rho*][, *'-cMass'*, *cFlag*][, *'-doRayleigh'*, *rFlag*])
One way is to specify an area and a UniaxialMaterial identifier:

element (*'TrussSection'*, *eleTag*, **eleNodes*, *A*, *secTag*[, *'-rho'*, *rho*][, *'-cMass'*, *cFlag*][, *'-doRayleigh'*, *rFlag*])
the other is to specify a Section identifier:

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`A` (float)	cross-sectional area of element
`matTag` (int)	tag associated with previously-defined UniaxialMaterial
`secTag` (int)	tag associated with previously-defined Section
`rho` (float)	mass per unit length, optional, default = 0.0
`cFlag` (float)	consistent mass flag, optional, default = 0 • `cFlag` = 0 lumped mass matrix (default) • `cFlag` = 1 consistent mass matrix
`rFlag` (float)	Rayleigh damping flag, optional, default = 0 • `rFlag` = 0 NO RAYLEIGH DAMPING (default) • `rFlag` = 1 include Rayleigh damping

Note:

1. The truss element DOES NOT include geometric nonlinearities, even when used with beam-columns utilizing P-Delta or Corotational transformations.
2. When constructed with a UniaxialMaterial object, the truss element considers strain-rate effects, and is thus suitable for use as a damping element.
3. The valid queries to a truss element when creating an ElementRecorder object are 'axialForce,' 'forces,' 'localForce', deformations,' 'material matArg1 matArg2…,' 'section sectArg1 sectArg2…' There will be more queries after the interface for the methods involved have been developed further.

See also:

Notes

Corotational Truss Element

This command is used to construct a corotational truss element object. There are two ways to construct a corotational truss element object:

element(*'corotTruss', eleTag, *eleNodes, A, matTag*[, *'-rho', rho*][, *'-cMass', cFlag*][, *'-doRayleigh', rFlag*])

One way is to specify an area and a UniaxialMaterial identifier:

element(*'corotTrussSection', eleTag, *eleNodes, A, secTag*[, *'-rho', rho*][, *'-cMass', cFlag*][, *'-doRayleigh', rFlag*])

the other is to specify a Section identifier:

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`A` (float)	cross-sectional area of element
`matTag` (int)	tag associated with previously-defined UniaxialMaterial
`secTag` (int)	tag associated with previously-defined Section
`rho` (float)	mass per unit length, optional, default = 0.0
`cFlag` (float)	consistent mass flag, optional, default = 0 • `cFlag` = 0 lumped mass matrix (default) • `cFlag` = 1 consistent mass matrix
`rFlag` (float)	Rayleigh damping flag, optional, default = 0 • `rFlag` = 0 NO RAYLEIGH DAMPING (default) • `rFlag` = 1 include Rayleigh damping

Note:

1. When constructed with a UniaxialMaterial object, the corotational truss element considers strain-rate effects, and is thus suitable for use as a damping element.
2. The valid queries to a truss element when creating an ElementRecorder object are 'axialForce,' 'stiff,' deformations,' 'material matArg1 matArg2. . . ,' 'section sectArg1 sectArg2. . . ' There will be more queries after the interface for the methods involved have been developed further.
3. CorotTruss DOES NOT include Rayleigh damping by default.

See also:

Notes

Elastic Beam Column Element

This command is used to construct an elasticBeamColumn element object. The arguments for the construction of an elastic beam-column element depend on the dimension of the problem, ndm:

element (*'elasticBeamColumn'*, *eleTag*, **eleNodes*, *A*, *E*, *Iz*, *transfTag*[, *'-mass'*, *massDens*][, *'-cMass'*])
For a two-dimensional problem

element (*'elasticBeamColumn'*, *eleTag*, **eleNodes*, *A*, *E*, *G*, *J*, *Iy*, *Iz*, *transfTag*[, *'-mass'*, *massDens*][, *'-cMass'*])
For a three-dimensional problem

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
A (float)	cross-sectional area of element
E (float)	Young's Modulus
G (float)	Shear Modulus
J (float)	torsional moment of inertia of cross section
Iz (float)	second moment of area about the local z-axis
Iy (float)	second moment of area about the local y-axis
transfTag (int)	identifier for previously-defined coordinate-transformation (CrdTransf) object
massDens (float)	element mass per unit length (optional, default = 0.0)
'-cMass' (str)	to form consistent mass matrix (optional, default = lumped mass matrix)

See also:

Notes

Elastic Beam Column Element with Stiffness Modifiers

This command is used to construct a ModElasticBeam2d element object. The arguments for the construction of an elastic beam-column element with stiffness modifiers is applicable for 2-D problems. This element should be used for modelling of a structural element with an equivalent combination of one elastic element with stiffness-proportional damping, and two springs at its two ends with no stiffness proportional damping to represent a prismatic section. The modelling technique is based on a number of analytical studies discussed in Zareian and Medina (2010) and Zareian and Krawinkler (2009) and is utilized in order to solve problems related to numerical damping in dynamic analysis of frame structures with concentrated plasticity springs.

element (*'ModElasticBeam2d'*, *eleTag*, **eleNodes*, *A*, *E*, *Iz*, *K11*, *K33*, *K44*, *transfTag*[, *'-mass'*, *massDens*][, *'-cMass'*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
A (float)	cross-sectional area of element
E (float)	Young's Modulus
Iz (float)	second moment of area about the local z-axis
K11 (float)	stiffness modifier for translation
K33 (float)	stiffness modifier for translation
K44 (float)	stiffness modifier for rotation
transfTag (int)	identifier for previously-defined coordinate-transformation (CrdTransf) object
massDens (float)	element mass per unit length (optional, default = 0.0)
'-cMass' (str)	to form consistent mass matrix (optional, default = lumped mass matrix)

See also:

Notes

Elastic Timoshenko Beam Column Element

This command is used to construct an ElasticTimoshenkoBeam element object. A Timoshenko beam is a frame member that accounts for shear deformations. The arguments for the construction of an elastic Timoshenko beam element depend on the dimension of the problem, ndm:

element (*'ElasticTimoshenkoBeam'*, *eleTag*, **eleNodes*, *E*, *G*, *A*, *Iz*, *Avy*, *transfTag*[, *'-mass'*, *massDens*][, *'-cMass*])
For a two-dimensional problem:

element (*'ElasticTimoshenkoBeam'*, *eleTag*, **eleNodes*, *E*, *G*, *A*, *Iz*, *Jx*, *Iy*, *Iz*, *Avy*, *Avz*, *transfTag*[, *'-mass'*, *massDens*][, *'-cMass*])
For a three-dimensional problem:

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`E` (float)	Young's Modulus
`G` (float)	Shear Modulus
`A` (float)	cross-sectional area of element
`Jx` (float)	torsional moment of inertia of cross section
`Iy` (float)	second moment of area about the local y-axis
`Iz` (float)	second moment of area about the local z-axis
`Avy` (float)	Shear area for the local y-axis
`Avz` (float)	Shear area for the local z-axis
`transfTag` (int)	identifier for previously-defined coordinate-transformation (CrdTransf) object
`massDens` (float)	element mass per unit length (optional, default = 0.0)
`'-cMass'` (str)	to form consistent mass matrix (optional, default = lumped mass matrix)

See also:

Notes

Beam With Hinges Element

This command is used to construct a *forceBeamColumn* element object, which is based on the non-iterative (or iterative) flexibility formulation. The locations and weights of the element integration points are based on so-called plastic hinge integration, which allows the user to specify plastic hinge lenghts at the element ends. Two-point Gauss integration is used on the element interior while two-point Gauss-Radau integration is applied over lengths of 4LpI and 4LpJ at the element ends, viz. "modified Gauss-Radau plastic hinge integration". A total of six integration points are used in the element state determination (two for each hinge and two for the interior).

Users may be familiar with the beamWithHinges command format (see below); however, the format shown here allows for the simple but important case of using a material nonlinear section model on the element interior. The previous beamWithHinges command constrained the user to an elastic interior, which often led to unconservative estimates of the element resisting force when plasticity spread beyond the plastic hinge regions in to the element interior.

The advantages of this new format over the previous beamWithHinges command are

- Plasticity can spread beyond the plastic hinge regions
- Hinges can form on the element interior, e.g., due to distributed member loads

To create a beam element with hinges, one has to use a *forceBeamColumn* element with following `beamIntegration()`.

Note:

- `'HingeRadau'` – two-point Gauss-Radau applied to the hinge regions over 4LpI and 4LpJ (six element integration points)
- `'HingeRadauTwo'` – two-point Gauss-Radau in the hinge regions applied over LpI and LpJ (six element integration points)

- `'HingeMidpoint'` – midpoint integration over the hinge regions (four element integration points)
- `'HingeEndpoint'` – endpoint integration over the hinge regions (four element integration points)

See also:

For more information on the behavior, advantages, and disadvantages of these approaches to plastic hinge integration, see

Scott, M.H. and G.L. Fenves. "Plastic Hinge Integration Methods for Force-Based Beam-Column Elements", Journal of Structural Engineering, 132(2):244-252, February 2006.

Scott, M.H. and K.L. Ryan. "Moment-Rotation Behavior of Force-Based Plastic Hinge Elements", Earthquake Spectra, 29(2):597-607, May 2013.

The primary advantages of HingeRadau are

- The user can specify a physically meaningful plastic hinge length
- The largest bending moment is captured at the element ends
- The exact numerical solution is recovered for a linear-elastic prismatic beam
- The characteristic length is equal to the user-specified plastic hinge length when deformations localize at the element ends

while the primary disadvantages are

- The element post-yield response is too flexible for strain-hardening section response (consider using HingeRadauTwo)
- The user needs to know the plastic hinge length a priori (empirical equations are available)

dispBeamColumn

element (*'dispBeamColumn'*, *eleTag*, *iNode*, *jNode*, *transfTag*, *integrationTag*, *'-cMass'*, *'-mass'*, *mass=0.0*)
Create a ForceBeamColumn element.

`eleTag` (int)	tag of the element
`iNode` (int)	tag of node i
`jNode` (int)	tag of node j
`transfTag` (int)	tag of transformation
`integrationTag` (int)	tag of `beamIntegration()`
`'-cMass'`	to form consistent mass matrix (optional, default = lumped mass matrix)
`mass` (float)	element mass density (per unit length), from which a lumped-mass matrix is formed (optional)

forceBeamColumn

element (*'forceBeamColumn'*, *eleTag*, *iNode*, *jNode*, *transfTag*, *integrationTag*, *'-iter'*, *maxIter=10*, *tol=1e-12*, *'-mass'*, *mass=0.0*)
Create a ForceBeamColumn element.

`eleTag` (int)	tag of the element
`iNode` (int)	tag of node i
`jNode` (int)	tag of node j
`transfTag` (int)	tag of transformation
`integrationTag` (int)	tag of *beamIntegration()*
`maxIter` (float)	maximum number of iterations to undertake to satisfy element compatibility (optional)
`tol` (float)	tolerance for satisfaction of element compatibility (optional)
`mass` (float)	element mass density (per unit length), from which a lumped-mass matrix is formed (optional)

Flexure-Shear Interaction Displacement-Based Beam-Column Element

This command is used to construct a dispBeamColumnInt element object, which is a distributed-plasticity, displacement-based beam-column element which includes interaction between flexural and shear components.

element (*'dispBeamColumnInt', eleTag,*eleNodes,numIntgrPts, secTag, transfTag, cRot['-mass', massDens]*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`numIntgrPts` (int)	number of integration points along the element.
`secTag` (int)	identifier for previously-defined section object
`transfTag` (int)	identifier for previously-defined coordinate-transformation (CrdTransf) object
`cRot` (float)	identifier for element center of rotation (or center of curvature distribution). Fraction of the height distance from bottom to the center of rotation (0 to 1)
`massDens` (float)	element mass density (per unit length), from which a lumped-mass matrix is formed (optional, default=0.0)

See also:

Notes

MVLEM - Multiple-Vertical-Line-Element-Model for RC Walls

The MVLEM element command is used to generate a two-dimensional Multiple-Vertical-Line-Element-Model (MVLEM; Vulcano et al., 1988; Orakcal et al., 2004, Kolozvari et al., 2015) for simulation of flexure-dominated RC wall behavior. A single model element incorporates six global degrees of freedom, three of each located at the center of rigid top and bottom beams, as illustrated in Figure 1a. The axial/flexural response of the MVLEM is simulated by a series of uniaxial elements (or macro-fibers) connected to the rigid beams at the top and bottom (e.g., floor) levels, whereas the shear response is described by a shear spring located at height ch from the bottom of the wall element (Figure 1a). Shear and flexural responses of the model element are uncoupled. The relative rotation between top and bottom faces of the wall element occurs about the point located on the central axis of the element at height ch (Figure 1b). Rotations and resulting transverse displacements are calculated based on the wall curvature, derived from section and material properties, corresponding to the bending moment at height ch of each element (Figure

1b). A value of c=0.4 was recommended by Vulcano et al. (1988) based on comparison of the model response with experimental results.

element (*'MVLEM'*, *eleTag*, *Dens*, **eleNodes*, *m*, *c*, *'-thick'*, **Thicknesses*, *'-width'*, **Widths*, *'-rho'*, **Reinforcing_ratios*, *'-matConcrete'*, **Concrete_tags*, *'-matSteel'*, **Steel_tags*, *'-matShear'*, *Shear_tag*)

`eleTag` (int)	unique element object tag
`Dens` (float)	Wall density
`eleNodes` (list (int))	a list of two element nodes
`m` (int)	Number of element macro-fibers
`c` (float)	Location of center of rotation from the iNode, `c` = 0.4 (recommended)
`Thicknesses` (list (float))	a list of `m` macro-fiber thicknesses
`Widths` (list (float))	a list of `m` macro-fiber widths
`Reinforcing_ratios` (list (float))	a list of m reinforcing ratios corresponding to macro-fibers; for each fiber: $rho_i = A_{s,i}/A_{gross,i}(1 < i < m)$
`Concrete _tags` (list (int))	a list of `m` uniaxialMaterial tags for concrete
`Steel_tags` (list (int))	a list of `m` uniaxialMaterial tags for steel
`Shear_tag` (int)	Tag of uniaxialMaterial for shear material

See also:

Notes

SFI MVLEM - Cyclic Shear-Flexure Interaction Model for RC Walls

The SFI_MVLEM command is used to construct a Shear-Flexure Interaction Multiple-Vertical-Line-Element Model (SFI-MVLEM, Kolozvari et al., 2015a, b, c), which captures interaction between axial/flexural and shear behavior of RC structural walls and columns under cyclic loading. The SFI_MVLEM element (Figure 1) incorporates 2-D RC panel behavior described by the Fixed-Strut-Angle-Model (nDMaterial FSAM; Ulugtekin, 2010; Orakcal et al., 2012), into a 2-D macroscopic fiber-based model (MVLEM). The interaction between axial and shear behavior is captured at each RC panel (macro-fiber) level, which further incorporates interaction between shear and flexural behavior at the SFI_MVLEM element level.

element (*''*, *eleTag*, **eleNodes*, *m*, *c*, *'-thick'*, **Thicknesses*, *'-width'*, **Widths*, *'-mat'*, **Material_tags*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`m` (int)	Number of element macro-fibers
`c` (float)	Location of center of rotation with from the iNode, `c` = 0.4 (recommended)
`Thicknesses` (list (float))	a list of m macro-fiber thicknesses
`Widths` (list (float))	a list of m macro-fiber widths
`Material_tags` (list (int))	a list of m macro-fiber nDMaterial1 tags

See also:

Notes

BeamColumnJoint Element

This command is used to construct a two-dimensional beam-column-joint element object. The element may be used with both two-dimensional and three-dimensional structures; however, load is transferred only in the plane of the element.

element (*'beamColumnJoint'*, *eleTag*, **eleNodes*, *Mat1*, *Mat2*, *Mat3*, *Mat4*, *Mat5*, *Mat6*, *Mat7*, *Mat8*, *Mat9*, *Mat10*, *Mat11*, *Mat12*, *Mat13*[, *eleHeightFac=1.0*, *eleWidthFac=1.0*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes
`Mat1` (int)	uniaxial material tag for left bar-slip spring at node 1
`Mat2` (int)	uniaxial material tag for right bar-slip spring at node 1
`Mat3` (int)	uniaxial material tag for interface-shear spring at node 1
`Mat4` (int)	uniaxial material tag for lower bar-slip spring at node 2
`Mat5` (int)	uniaxial material tag for upper bar-slip spring at node 2
`Mat6` (int)	uniaxial material tag for interface-shear spring at node 2
`Mat7` (int)	uniaxial material tag for left bar-slip spring at node 3
`Mat8` (int)	uniaxial material tag for right bar-slip spring at node 3
`Mat9` (int)	uniaxial material tag for interface-shear spring at node 3
`Mat10` (int)	uniaxial material tag for lower bar-slip spring at node 4
`Mat11` (int)	uniaxial material tag for upper bar-slip spring at node 4
`Mat12` (int)	uniaxial material tag for interface-shear spring at node 4
`Mat13` (int)	uniaxial material tag for shear-panel
`eleHeightFac` (float)	floating point value (as a ratio to the total height of the element) to be considered for determination of the distance in between the tension-compression couples (optional, default: 1.0)
`eleWidthFac` (float)	floating point value (as a ratio to the total width of the element) to be considered for determination of the distance in between the tension-compression couples (optional, default: 1.0)

See also:

Notes

ElasticTubularJoint Element

This command is used to construct an ElasticTubularJoint element object, which models joint flexibility of tubular joints in two dimensional analysis of any structure having tubular joints.

element (*'ElasticTubularJoint'*, *eleTag*, **eleNodes*, *Brace_Diameter*, *Brace_Angle*, *E*, *Chord_Diameter*, *Chord_Thickness*, *Chord_Angle*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`Brace_Diameter` (float)	outer diameter of brace
`Brace_Angle` (float)	angle between brace and chord axis 0 < Brace_Angle < 90
`E` (float)	Young's Modulus
`Chord_Diameter` (float)	outer diameter of chord
`Chord_Thickness` (float)	thickness of chord
`Chord_Angle` (float)	angle between chord axis and global x-axis 0 < Chord_Angle < 180

See also:

Notes

Joint2D Element

This command is used to construct a two-dimensional beam-column-joint element object. The two dimensional beam-column joint is idealized as a parallelogram shaped shear panel with adjacent elements connected to its mid-points. The midpoints of the parallelogram are referred to as external nodes. These nodes are the only analysis components that connect the joint element to the surrounding structure.

element (*'Joint2D'*, *eleTag*, **eleNodes*[, *Mat1*, *Mat2*, *Mat3*, *Mat4*], *MatC*, *LrgDspTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of five element nodes = `[nd1,nd2,nd3,nd4,ndC]`. `ndC` is the central node of beam-column joint. (the tag `ndC` is used to generate the internal node, thus, the node should not exist in the domain or be used by any other node)
`Mat1` (int)	uniaxial material tag for interface rotational spring at node 1. Use a zero tag to indicate the case that a beam-column element is rigidly framed to the joint. (optional)
`Mat2` (int)	uniaxial material tag for interface rotational spring at node 2. Use a zero tag to indicate the case that a beam-column element is rigidly framed to the joint. (optional)
`Mat3` (int)	uniaxial material tag for interface rotational spring at node 3. Use a zero tag to indicate the case that a beam-column element is rigidly framed to the joint. (optional)
`Mat4` (int)	uniaxial material tag for interface rotational spring at node 4. Use a zero tag to indicate the case that a beam-column element is rigidly framed to the joint. (optional)
`MatC` (int)	uniaxial material tag for rotational spring of the central node that describes shear panel behavior
`LrgDspTag` (int)	an integer indicating the flag for considering large deformations: * `0` - for small deformations and constant geometry * `1` - for large deformations and time varying geometry * `2` - for large deformations ,time varying geometry and length correction

See also:

Notes

Two Node Link Element

This command is used to construct a twoNodeLink element object, which is defined by two nodes. The element can have zero or non-zero length. This element can have 1 to 6 degrees of freedom, where only the transverse and rotational degrees of freedom are coupled as long as the element has non-zero length. In addition, if the element length is larger than zero, the user can optionally specify how the P-Delta moments around the local x- and y-axis are distributed among a moment at node i, a moment at node j, and a shear couple. The sum of these three ratios is always equal to 1. In addition the shear center can be specified as a fraction of the element length from the iNode. The element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized. It is important to recognize that if this element has zero length, it does not consider the geometry as given by the nodal coordinates, but utilizes the user-defined orientation vectors to determine the directions of the springs.

element (*'twoNodeLink'*, *eleTag*, **eleNodes*, *'-mat'*, **matTags*, *'-dir'*, **dirs*[, *'-orient'*, **vecx*, **vecy*][, *'-pDelta'*, **Mratio*][, *'-shearDist'*, **sDratios*][, *'-doRayleigh'*][, *'-mass'*, *m*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`matTags` (list (int))	a list of tags associated with previously-defined UniaxialMaterial objects
`dirs` (list (int))	a list material directions: • 2D-case: `1` , `2` - translations along local x,y axes; `3` - rotation about local z axis • 3D-case: `1`, `2`, `3` - translations along local x,y,z axes; `4`, `5`, `6` - rotations about local x,y,z axes
`vecx` (list (float))	vector components in global coordinates defining local x-axis (optional)
`vecy` (list (float))	vector components in global coordinates defining local y-axis (optional)
`Mratios` (list (float))	P-Delta moment contribution ratios, size of ratio vector is 2 for 2D-case and 4 for 3D-case (entries: `[My_iNode, My_jNode, Mz_iNode, Mz_jNode]`) `My_iNode` + `My_jNode` <= 1.0, `Mz_iNode` + `Mz_jNode` <= 1.0. Remaining P-Delta moments are resisted by shear couples. (optional)
`sDratios` (list (float))	shear distances from iNode as a fraction of the element length, size of ratio vector is 1 for 2D-case and 2 for 3D-case. (entries: `[dy_iNode, dz_iNode]`) (optional, default = `[0.5, 0.5]`)
`'-doRayleigh'` (str)	to include Rayleigh damping from the element (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)

See also:

Notes

Elastomeric Bearing (Plasticity) Element

This command is used to construct an elastomericBearing element object, which is defined by two nodes. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) plasticity properties for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. By default (sDratio = 0.5) P-Delta moments are equally distributed to the two end-nodes. To avoid the introduction of artificial viscous damping in the isolation system (sometimes referred to as "damping leakage in the isolation system"), the bearing element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

element (*'elastomericBearingPlasticity'*, *eleTag*, **eleNodes*, *kInit*, *qd*, *alpha1*, *alpha2*, *mu*, *'-P'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*])

For a two-dimensional problem

element (*'elastomericBearingPlasticity'*, *eleTag*, **eleNodes*, *kInit*, *qd*, *alpha1*, *alpha2*, *mu*, *'-P'*, *matTag*, *'-T'*, *matTag*, *'-My'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*])

For a three-dimensional problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`kInit` (float)	initial elastic stiffness in local shear direction
`qd` (float)	characteristic strength
`alpha1` (float)	post yield stiffness ratio of linear hardening component
`alpha2` (float)	post yield stiffness ratio of non-linear hardening component
`mu` (float)	exponent of non-linear hardening component
`'-P' matTag` (int)	tag associated with previously-defined UniaxialMaterial in axial direction
`'-T' matTag` (int)	tag associated with previously-defined UniaxialMaterial in torsional direction
`'-My' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local y-axis
`'-Mz' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local z-axis
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.5)
`'-doRayleigh'` (str)	to include Rayleigh damping from the bearing (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)

See also:

Notes

Elastomeric Bearing (Bouc-Wen) Element

This command is used to construct an elastomericBearing element object, which is defined by two nodes. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) plasticity properties for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. By default (sDratio = 0.5) P-Delta moments are equally distributed to the two end-nodes. To avoid the introduction of artificial viscous damping in the isolation system (sometimes referred to as “damping leakage in the isolation system”), the bearing element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

element (*'elastomericBearingBoucWen'*, *eleTag*, **eleNodes*, *kInit*, *qd*, *alpha1*, *alpha2*, *mu*, *eta*, *beta*, *gamma* *'-P'*, *matTag* *'-Mz'*, *matTag*[, *'-orient'*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*])

For a two-dimensional problem

element (*'elastomericBearingBoucWen'*, *eleTag*, **eleNodes*, *kInit*, *qd*, *alpha1*, *alpha2*, *mu*, *eta*, *beat*, *gamma*, *'-P'*, *matTag*, *'-T'*, *matTag*, *'-My'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass*, *m*])

For a three-dimensional problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`kInit` (float)	initial elastic stiffness in local shear direction
`qd` (float)	characteristic strength
`alpha1` (float)	post yield stiffness ratio of linear hardening component
`alpha2` (float)	post yield stiffness ratio of non-linear hardening component
`mu` (float)	exponent of non-linear hardening component
`eta` (float)	yielding exponent (sharpness of hysteresis loop corners) (default = 1.0)
`beta` (float)	first hysteretic shape parameter (default = 0.5)
`gamma` (float)	second hysteretic shape parameter (default = 0.5)
`'-P' matTag` (int)	tag associated with previously-defined UniaxialMaterial in axial direction
`'-T' matTag` (int)	tag associated with previously-defined UniaxialMaterial in torsional direction
`'-My' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local y-axis
`'-Mz' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local z-axis
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.5)
`'-doRayleigh'` (str)	to include Rayleigh damping from the bearing (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)

See also:

Notes

Flat Slider Bearing Element

This command is used to construct a flatSliderBearing element object, which is defined by two nodes. The iNode represents the flat sliding surface and the jNode represents the slider. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) friction properties for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. To capture the uplift behavior of the bearing, the user-specified UniaxialMaterial in the axial direction is modified for no-tension behavior. By default (sDratio = 0.0) P-Delta moments are entirely transferred to the flat sliding surface (iNode). It is important to note that rotations of the flat sliding surface (rotations at the iNode) affect the shear behavior of the bearing. To avoid the introduction of artificial viscous damping in the isolation system (sometimes referred to as “damping leakage in the isolation system”), the bearing element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

element (*'flatSliderBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *kInit*, *'-P'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])

For a two-dimensional problem

element (*'flatSliderBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *kInit*, *'-P'*, *matTag*, *'-T'*, *matTag*, *'-My'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])

For a three-dimensional problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`frnMdlTag` (float)	tag associated with previously-defined FrictionModel
`kInit` (float)	initial elastic stiffness in local shear direction
`'-P'` `matTag` (int)	tag associated with previously-defined UniaxialMaterial in axial direction
`'-T'` `matTag` (int)	tag associated with previously-defined UniaxialMaterial in torsional direction
`'-My'` `matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local y-axis
`'-Mz'` `matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local z-axis
`x1` `x2` `x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1` `y2` `y3` (float)	vector components in global coordinates defining local y-axis (optional)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.0)
`'-doRayleigh'` (str)	to include Rayleigh damping from the bearing (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)
`maxIter` (int)	maximum number of iterations to undertake to satisfy element equilibrium (optional, default = 20)
`tol` (float)	convergence tolerance to satisfy element equilibrium (optional, default = 1E-8)

See also:

Notes

Single Friction Pendulum Bearing Element

This command is used to construct a singleFPBearing element object, which is defined by two nodes. The iNode represents the concave sliding surface and the jNode represents the articulated slider. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) friction properties (with post-yield stiffening due to the concave sliding surface) for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. To capture the uplift behavior of the bearing, the user-specified UniaxialMaterial in the axial direction is modified for no-tension behavior. By default (sDratio = 0.0) P-Delta moments are entirely transferred to the concave sliding surface (iNode). It is important to note that rotations of the concave sliding surface (rotations at the iNode) affect the shear behavior of the bearing. To avoid the introduction of artificial viscous damping in the isolation system (sometimes referred to as “damping leakage in the isolation system”), the bearing element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

element (*'singleFPBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *Reff*, *kInit*, *'-P'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])

For a two-dimensional problem

element (*'singleFPBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *Reff*, *kInit*, *'-P'*, *matTag*, *'-T'*, *matTag*, *'-My'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])

For a three-dimensional problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`frnMdlTag` (float)	tag associated with previously-defined FrictionModel
`Reff` (float)	effective radius of concave sliding surface
`kInit` (float)	initial elastic stiffness in local shear direction
`'-P' matTag` (int)	tag associated with previously-defined UniaxialMaterial in axial direction
`'-T' matTag` (int)	tag associated with previously-defined UniaxialMaterial in torsional direction
`'-My' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local y axis
`'-Mz' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local z-axis
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.0)
`'-doRayleigh'` (str)	to include Rayleigh damping from the bearing (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)
`maxIter` (int)	maximum number of iterations to undertake to satisfy element equilibrium (optional, default = 20)
`tol` (float)	convergence tolerance to satisfy element equilibrium (optional, default = 1E-8)

See also:

Notes

Triple Friction Pendulum Bearing Element

This command is used to construct a Triple Friction Pendulum Bearing element object, which is defined by two nodes. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) friction properties (with post-yield stiffening due to the concave sliding surface) for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. To capture the uplift behavior of the bearing, the user-specified UniaxialMaterial in the axial direction is modified for no-tension behavior. P-Delta moments are entirely transferred to the concave sliding surface (iNode). It is important to note that rotations of the concave sliding surface (rotations at the iNode) affect the shear behavior of the bearing. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

`element`(*'TFP'*, *eleTag*, **eleNodes*, *R1*, *R2*, *R3*, *R4*, *D1*, *D2*, *D3*, *D4*, *d1*, *d2*, *d3*, *d4*, *mu1*, *mu2*, *mu3*, *mu4*, *h1*, *h2*, *h3*, *h4*, *H0*, *colLoad*[, *K*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
R1 (float)	Radius of inner bottom sliding surface
R2 (float)	Radius of inner top sliding surface
R3 (float)	Radius of outer bottom sliding surface
R4 (float)	Radius of outer top sliding surface
D1 (float)	Diameter of inner bottom sliding surface
D2 (float)	Diameter of inner top sliding surface
D3 (float)	Diameter of outer bottom sliding surface
D4 (float)	Diameter of outer top sliding surface
d1 (float)	diameter of inner slider
d2 (float)	diameter of inner slider
d3 (float)	diameter of outer bottom slider
d4 (float)	diameter of outer top slider
mu1 (float)	friction coefficient of inner bottom sliding surface
mu2 (float)	friction coefficient of inner top sliding surface
mu3 (float)	friction coefficient of outer bottom sliding surface
mu4 (float)	friction coefficient of outer top sliding surface
h1 (float)	height from inner bottom sliding surface to center of bearing
h2 (float)	height from inner top sliding surface to center of bearing
h3 (float)	height from outer bottom sliding surface to center of bearing
h4 (float)	height from inner top sliding surface to center of bearing
H0 (float)	total height of bearing
colLoad (float)	initial axial load on bearing (only used for first time step then load come from model)
K (float)	optional, stiffness of spring in vertical dirn (dof 2 if ndm= 2, dof 3 if ndm = 3) (default=1.0e15)

See also:

Notes

Triple Friction Pendulum Element

element (*'TripleFrictionPendulum'*, *eleTag*, **eleNodes*, *frnTag1*, *frnTag2*, *frnTag3*, *vertMatTag*, *rotZMatTag*, *rotXMatTag*, *rotYMatTag*, *L1*, *L2*, *L3*, *d1*, *d2*, *d3*, *W*, *uy*, *kvt*, *minFv*, *tol*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`frnTag1, frnTag2 frnTag3` (int)	= tags associated with previously-defined FrictionModels at the three sliding interfaces
`vertMatTag` (int)	= Pre-defined material tag for COMPRESSION behavior of the bearing
`rotZMatTag rotXMatTag rotYMatTag` (int)	= Pre-defined material tags for rotational behavior about 3-axis, 1-axis and 2-axis, respectively.
`L1 L2 L3` (float)	= effective radii. Li = R_i - h_i (see Figure 1)
`d1 d2 d3` (float)	= displacement limits of pendulums (Figure 1). Displacement limit of the bearing is 2 `d1` + `d2` + `d3` + `L1`. `d3`/ `L3` - `L1`. `d2`/ `L2`
`W` (float)	= axial force used for the first trial of the first analysis step.
`uy` (float)	= lateral displacement where sliding of the bearing starts. Recommended value = 0.25 to 1 mm. A smaller value may cause convergence problem.
`kvt` (float)	= Tension stiffness k_vt of the bearing.
`minFv (>=0)` (float)	= minimum vertical compression force in the bearing used for computing the horizontal tangent stiffness matrix from the normalized tangent stiffness matrix of the element. `minFv` is substituted for the actual compressive force when it is less than `minFv`, and prevents the element from using a negative stiffness matrix in the horizontal direction when uplift occurs. The vertical nodal force returned to nodes is always computed from `kvc` (or `kvt`) and vertical deformation, and thus is not affected by `minFv`.
`tol` (float)	= relative tolerance for checking the convergence of the element. Recommended value = 1.e-10 to 1.e-3.

See also:

Notes

MultipleShearSpring Element

This command is used to construct a multipleShearSpring (MSS) element object, which is defined by two nodes. This element consists of a series of identical shear springs arranged radially to represent the isotropic behavior in the local y-z plane.

element (*'multipleShearSpring'*, *eleTag*, **eleNodes*, *nSpring*, *'-mat'*, *matTag*[, *'-lim'*, *dsp*][, *'-orient'*[, *x1*, *x2*, *x3*], *yp1*, *yp2*, *yp3*][, *'-mass'*, *m*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`nSpring` (int)	number of springs
`matTag` (int)	tag associated with previously-defined UniaxialMaterial object
`dsp` (float)	minimum deformation to calculate equivalent coefficient (see note 1)
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis
`yp1 yp2 yp3` (float)	vector components in global coordinates defining vector yp which lies in the local x-y plane for the element
`m` (float)	element mass

Note: If `dsp` is positive and the shear deformation of MSS exceeds `dsp`, this element calculates equivalent coefficient to adjust force and stiffness of MSS. The adjusted MSS force and stiffness reproduce the behavior of the previously defined uniaxial material under monotonic loading in every direction. If `dsp` is zero, the element does not calculate the equivalent coefficient.

See also:

Notes

KikuchiBearing Element

This command is used to construct a KikuchiBearing element object, which is defined by two nodes. This element consists of multiple shear spring model (MSS) and multiple normal spring model (MNS).

element (*'KikuchiBearing'*, *eleTag*, **eleNodes*, *'-shape'*, *shape*, *'-size'*, *size*, *totalRubber*[, *'-totalHeight'*, *totalHeight*], *'-nMSS'*, *nMSS*, *'-matMSS'*, *matMSSTag*[, *'-limDisp'*, *limDisp*], *'-nMNS'*, *nMNS*, *'-matMNS'*, *matMNSTag*[, *'-lambda'*, *lambda*][, *'-orient'*[, *x1*, *x2*, *x3*], *yp1*, *yp2*, *yp3*][, *'-mass'*, *m*][, *'-noPDInput'*][, *'-noTilt'*][, *'-adjustPDOutput'*, *ci*, *cj*][, *'-doBalance'*, *limFo*, *limFi*, *nIter*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`shape` (float)	following shapes are available: round, square
`size` (float)	diameter (round shape), length of edge (square shape)
`totalRubber` (float)	total rubber thickness
`totalHeight` (float)	total height of the bearing (defaulut: distance between iNode and jNode)
`nMSS` (int)	number of springs in MSS = nMSS
`matMSSTag` (int)	matTag for MSS
`limDisp` (float)	minimum deformation to calculate equivalent coefficient of MSS (see note 1)
`nMNS` (int)	number of springs in MNS = nMNS*nMNS (for round and square shape)
`matMNSTag` (int)	matTag for MNS
`lambda` (float)	parameter to calculate compression modulus distribution on MNS (see note 2)
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis
`yp1 yp2 yp3` (float)	vector components in global coordinates defining vector yp which lies in the local x-y plane for the element
`m` (float)	element mass
`'-noPDInput'` (str)	not consider P-Delta moment
`'-noTilt'` (str)	not consider tilt of rigid link
`ci cj` (float)	P-Delta moment adjustment for reaction force (default: `ci` =0.5, `cj` =0.5)
`limFo limFi nIter` (float)	tolerance of external unbalanced force (`limFo`), tolorance of internal unbalanced force (`limFi`), number of iterations to get rid of internal unbalanced force (`nIter`)

See also:

Notes

YamamotoBiaxialHDR Element

This command is used to construct a YamamotoBiaxialHDR element object, which is defined by two nodes. This element can be used to represent the isotropic behavior of high-damping rubber bearing in the local y-z plane.

element (*'YamamotoBiaxialHDR'*, *eleTag*, **eleNodes*, *Tp*, *DDo*, *DDi*, *Hr*[, *'-coRS'*, *cr*, *cs*][, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-mass'*, *m*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
Tp (int)	compound type = 1 : X0.6R manufactured by Bridgestone corporation.
DDo (float)	outer diameter [m]
DDi (float)	bore diameter [m]
Hr (float)	total thickness of rubber layer [m] Optional Data
cr cs (float)	coefficients for shear stress components of τr and τs
x1 x2 x3 (float)	vector components in global coordinates defining local x-axis
yp1 yp2 yp3 (float)	vector components in global coordinates defining vector yp which lies in the local x-y plane for the element
m (float)	element mass [kg]

See also:

Notes

ElastomericX

This command is used to construct an ElastomericX bearing element object in three-dimension. The 3D continuum geometry of an elastomeric bearing is modeled as a 2-node, 12 DOF discrete element. This elements extends the formulation of Elastomeric_Bearing_(Bouc-Wen)_Element element. However, instead of the user providing material models as input arguments, it only requires geometric and material properties of an elastomeric bearing as arguments. The material models in six direction are formulated within the element from input arguments. The time-dependent values of mechanical properties (e.g., shear stiffness, buckling load capacity) can also be recorded using the "parameters" recorder.

element (*'ElastomericX'*, *eleTag*, **eleNodes*, *Fy*, *alpha*, *Gr*, *Kbulk*, *D1*, *D2*, *ts*, *tr*, *n*[[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *kc*][, *PhiM*][, *ac*][, *sDratio*][, *m*][, *cd*][, *tc*][, *tag1*][, *tag2*][, *tag3*][, *tag4*])

For 3D problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`Fy` (float)	yield strength
`alpha` (float)	post-yield stiffness ratio
`Gr` (float)	shear modulus of elastomeric bearing
`Kbulk` (float)	bulk modulus of rubber
`D1` (float)	internal diameter
`D2` (float)	outer diameter (excluding cover thickness)
`ts` (float)	single steel shim layer thickness
`tr` (float)	single rubber layer thickness
`n` (int)	number of rubber layers
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`kc` (float)	cavitation parameter (optional, default = 10.0)
`PhiM` (float)	damage parameter (optional, default = 0.5)
`ac` (float)	strength reduction parameter (optional, default = 1.0)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.5)
`m` (float)	element mass (optional, default = 0.0)
`cd` (float)	viscous damping parameter (optional, default = 0.0)
`tc` (float)	cover thickness (optional, default = 0.0)
`tag1` (float)	Tag to include cavitation and post-cavitation (optional, default = 0)
`tag2` (float)	Tag to include buckling load variation (optional, default = 0)
`tag3` (float)	Tag to include horizontal stiffness variation (optional, default = 0)
`tag4` (float)	Tag to include vertical stiffness variation (optional, default = 0)

Note: Because default values of heating parameters are in SI units, user must override the default heating parameters values if using Imperial units

User should distinguish between yield strength of elastomeric bearing (F_y) and characteristic strength (Q_d): $Q_d = F_y * (1 - alpha)$

See also:

Notes

LeadRubberX

This command is used to construct a LeadRubberX bearing element object in three-dimension. The 3D continuum geometry of a lead rubber bearing is modeled as a 2-node, 12 DOF discrete element. It extends the formulation of ElastomericX by including strength degradation in lead rubber bearing due to heating of the lead-core. The LeadRubberX element requires only the geometric and material properties of an elastomeric bearing as arguments. The material models in six direction are formulated within the element from input arguments. The time-dependent values of mechanical properties (e.g., shear stiffness, buckling load capacity, temperature in the lead-core, yield strength) can also be recorded using the "parameters" recorder.

element (*'LeadRubberX'*, *eleTag*, **eleNodes*, *Fy*, *alpha*, *Gr*, *Kbulk*, *D1*, *D2*, *ts*, *tr*, *n*[[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *kc*][, *PhiM*][, *ac*][, *sDratio*][, *m*][, *cd*][, *tc*][, *qL*][, *cL*][, *kS*][, *aS*][, *tag1*][, *tag2*][, *tag3*][, *tag4*][, *tag5*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
Fy (float)	yield strength
alpha (float)	post-yield stiffness ratio
Gr (float)	shear modulus of elastomeric bearing
Kbulk (float)	bulk modulus of rubber
D1 (float)	internal diameter
D2 (float)	outer diameter (excluding cover thickness)
ts (float)	single steel shim layer thickness
tr (float)	single rubber layer thickness
n (int)	number of rubber layers
x1 x2 x3 (float)	vector components in global coordinates defining local x-axis (optional)
y1 y2 y3 (float)	vector components in global coordinates defining local y-axis (optional)
kc (float)	cavitation parameter (optional, default = 10.0)
PhiM (float)	damage parameter (optional, default = 0.5)
ac (float)	strength reduction parameter (optional, default = 1.0)
sDratio (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.5)
m (float)	element mass (optional, default = 0.0)
cd (float)	viscous damping parameter (optional, default = 0.0)
tc (float)	cover thickness (optional, default = 0.0)
qL (float)	density of lead (optional, default = 11200 kg/m3)
cL (float)	specific heat of lead (optional, default = 130 N-m/kg oC)
kS (float)	thermal conductivity of steel (optional, default = 50 W/m oC)
aS (float)	thermal diffusivity of steel (optional, default = 1.41e-05 m2/s)
tag1 (int)	Tag to include cavitation and post-cavitation (optional, default = 0)
tag2 (int)	Tag to include buckling load variation (optional, default = 0)
tag3 (int)	Tag to include horizontal stiffness variation (optional, default = 0)
tag4 (int)	Tag to include vertical stiffness variation (optional, default = 0)
tag5 (int)	Tag to include strength degradation in shear due to heating of lead core (optional, default = 0)

Note: Because default values of heating parameters are in SI units, user must override the default heating parameters values if using Imperial units

User should distinguish between yield strength of elastomeric bearing (F_y) and characteristic strength (Q_d): $Q_d = F_y * (1 - alpha)$

See also:

Notes

HDR

This command is used to construct an HDR bearing element object in three-dimension. The 3D continuum geometry of an high damping rubber bearing is modeled as a 2-node, 12 DOF discrete element. This is the third element in the series of elements developed for analysis of base-isolated structures under extreme loading (others being ElastomericX and LeadRubberX). The major difference between HDR element with ElastomericX is the hysteresis model in shear. The HDR element uses a model proposed by Grant et al. (2004) to capture the shear behavior of a high damping rubber bearing. The time-dependent values of mechanical properties (e.g., vertical stiffness, buckling load capacity) can also be recorded using the "parameters" recorder.

element (*'HDR'*, *eleTag*, **eleNodes*, *Gr*, *Kbulk*, *D1*, *D2*, *ts*, *tr*, *n*, *a1*, *a2*, *a3*, *b1*, *b2*, *b3*, *c1*, *c2*, *c3*, *c4*[[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *kc*][, *PhiM*][, *ac*][, *sDratio*][, *m*][, *tc*])
For 3D problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`Gr` (float)	shear modulus of elastomeric bearing
`Kbulk` (float)	bulk modulus of rubber
`D1` (float)	internal diameter
`D2` (float)	outer diameter (excluding cover thickness)
`ts` (float)	single steel shim layer thickness
`tr` (float)	single rubber layer thickness
`n` (int)	number of rubber layers
`a1 a2 a3 b1 b2 b3 c1 c2 c3 c4` (float)	parameters of the Grant model
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`kc` (float)	cavitation parameter (optional, default = 10.0)
`PhiM` (float)	damage parameter (optional, default = 0.5)
`ac` (float)	strength reduction parameter (optional, default = 1.0)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.5)
`m` (float)	element mass (optional, default = 0.0)
`tc` (float)	cover thickness (optional, default = 0.0)

See also:

Notes

RJ-Watson EQS Bearing Element

This command is used to construct a RJWatsonEqsBearing element object, which is defined by two nodes. The iNode represents the masonry plate and the jNode represents the sliding surface plate. The element can have zero length or the appropriate bearing height. The bearing has unidirectional (2D) or coupled (3D) friction properties (with post-yield stiffening due to the mass-energy-regulator (MER) springs) for the shear deformations, and force-deformation behaviors defined by UniaxialMaterials in the remaining two (2D) or four (3D) directions. To capture the uplift behavior of the bearing, the user-specified UniaxialMaterial in the axial direction is modified for no-tension behavior. By default (sDratio = 1.0) P-Delta moments are entirely transferred to the sliding surface (jNode). It is important to note that rotations of the sliding surface (rotations at the jNode) affect the shear behavior of the bearing. To avoid the introduction of artificial viscous damping in the isolation system (sometimes referred to as "damping leakage in the isolation system"), the bearing element does not contribute to the Rayleigh damping by default. If the element has non-zero length, the local x-axis is determined from the nodal geometry unless the optional x-axis vector is specified in which case the nodal geometry is ignored and the user-defined orientation is utilized.

element (*'RJWatsonEqsBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *kInit*, *'-P'*, *matTag*, *'-Vy'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])
For a two-dimensional problem

element (*'RJWatsonEqsBearing'*, *eleTag*, **eleNodes*, *frnMdlTag*, *kInit*, *'-P'*, *matTag*, *'-Vy'*, *matTag*, *'-Vz'*, *matTag*, *'-T'*, *matTag*, *'-My'*, *matTag*, *'-Mz'*, *matTag*[, *'-orient'*[, *x1*, *x2*, *x3*], *y1*, *y2*, *y3*][, *'-shearDist'*, *sDratio*][, *'-doRayleigh'*][, *'-mass'*, *m*][, *'-iter'*, *maxIter*, *tol*])

For a three-dimensional problem

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of two element nodes
`frnMdlTag` (float)	tag associated with previously-defined FrictionModel
`kInit` (float)	initial stiffness of sliding friction component in local shear direction
`'-P' matTag` (int)	tag associated with previously-defined UniaxialMaterial in axial direction
`'-Vy' matTag` (int)	tag associated with previously-defined UniaxialMaterial in shear direction along local y-axis (MER spring behavior not including friction)
`'-Vz' matTag` (int)	tag associated with previously-defined UniaxialMaterial in shear direction along local z-axis (MER spring behavior not including friction)
`'-T' matTag` (int)	tag associated with previously-defined UniaxialMaterial in torsional direction
`'-My' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local y-axis
`'-Mz' matTag` (int)	tag associated with previously-defined UniaxialMaterial in moment direction around local z-axis
`x1 x2 x3` (float)	vector components in global coordinates defining local x-axis (optional)
`y1 y2 y3` (float)	vector components in global coordinates defining local y-axis (optional)
`sDratio` (float)	shear distance from iNode as a fraction of the element length (optional, default = 0.0)
`'-doRayleigh'` (str)	to include Rayleigh damping from the bearing (optional, default = no Rayleigh damping contribution)
`m` (float)	element mass (optional, default = 0.0)
`maxIter` (int)	maximum number of iterations to undertake to satisfy element equilibrium (optional, default = 20)
`tol` (float)	convergence tolerance to satisfy element equilibrium (optional, default = 1E-8)

See also:

Notes

FPBearingPTV

The FPBearingPTV command creates a single Friction Pendulum bearing element, which is capable of accounting for the changes in the coefficient of friction at the sliding surface with instantaneous values of the sliding velocity, axial pressure and temperature at the sliding surface. The constitutive modelling is similar to the existing singleFPBearing element, otherwise. The FPBearingPTV element has been verified and validated in accordance with the ASME guidelines, details of which are presented in Chapter 4 of Kumar et al. (2015a).

element(*'FPBearingPTV'*, *eleTag*, **eleNodes*, *MuRef*, *IsPressureDependent*, *pRef*, *isTemperatureDependent*, *Diffusivity*, *Conductivity*, *IsVelocityDependent*, *rateParameter*, *ReffectiveFP*, *Radius_Contact*, *kInitial*, *theMaterialA*, *theMaterialB*, *theMaterialC*, *theMaterialD*, *x1*, *x2*, *x3*, *y1*, *y2*, *y3*, *shearDist*, *doRayleigh*, *mass*, *iter*, *tol*, *unit*)

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of two element nodes
MuRef (float)	Reference coefficient of friction
IsPressureDependent (int)	1 if the coefficient of friction is a function of instantaneous axial pressure
pRef (float)	Reference axial pressure (the bearing pressure under static loads)
IsTemperatureDependent (int)	1 if the coefficient of friction is a function of instantaneous temperature at the sliding surface
Diffusivity (float)	Thermal diffusivity of steel
Conductivity (float)	Thermal conductivity of steel
IsVelocityDependent (int)	1 if the coefficient of friction is a function of instantaneous velocity at the sliding surface
rateParameter (float)	The exponent that determines the shape of the coefficient of friction vs. sliding velocity curve
ReffectiveFP (float)	Effective radius of curvature of the sliding surface of the FPbearing
Radius_Contact (float)	Radius of contact area at the sliding surface
kInitial (float)	Lateral stiffness of the sliding bearing before sliding begins
theMaterialA (int)	Tag for the uniaxial material in the axial direction
theMaterialB (int)	Tag for the uniaxial material in the torsional direction
theMaterialC (int)	Tag for the uniaxial material for rocking about local Y axis
theMaterialD (int)	Tag for the uniaxial material for rocking about local Z axis
x1 x2 x3 (float)	Vector components to define local X axis
y1 y2 y3 (float)	Vector components to define local Y axis
shearDist (float)	Shear distance from iNode as a fraction of the length of the element
doRayleigh (int)	To include Rayleigh damping from the bearing
mass (float)	Element mass
iter (int)	Maximum number of iterations to satisfy the equilibrium of element
tol (float)	Convergence tolerance to satisfy the equilibrium of the element
unit (int)	Tag to identify the unit from the list below. • 1: N, m, s, C • 2: kN, m, s, C • 3: N, mm, s, C • 4: kN, mm, s, C • 5: lb, in, s, C • 6: kip, in, s, C • 7: lb, ft, s, C • 8: kip, ft, s, C

See also:

Notes

Quad Element

This command is used to construct a FourNodeQuad element object which uses a bilinear isoparametric formulation.

element (*'quad'*, *eleTag*, **eleNodes*, *thick*, *type*, *matTag*[, *pressure*, *rho*, *b1*, *b2*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	element thickness
`type` (str)	string representing material behavior. The type parameter can be either `'PlaneStrain'` or `'PlaneStress'`
`matTag` (int)	tag of nDMaterial
`pressure` (float)	surface pressure (optional, default = 0.0)
`rho` (float)	element mass density (per unit volume) from which a lumped element mass matrix is computed (optional, default=0.0)
`b1` `b2` (float)	constant body forces defined in the isoparametric domain (optional, default=0.0)

Note:

1. Consistent nodal loads are computed from the pressure and body forces.
2. The valid queries to a Quad element when creating an ElementRecorder object are 'forces', 'stresses,' and 'material $matNum matArg1 matArg2 . . . ' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the isoparametric domain.

See also:

Notes

Shell Element

This command is used to construct a ShellMITC4 element object, which uses a bilinear isoparametric formulation in combination with a modified shear interpolation to improve thin-plate bending performance.

element (*'ShellMITC4'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. Currently must be either a `'PlateFiberSection'`, or `'ElasticMembranePlateSection'`

Note:

1. The valid queries to a Quad element when creating an ElementRecorder object are 'forces', 'stresses,' and 'material $matNum matArg1 matArg2 . . . ' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the isoparametric domain.

2. It is a 3D element with 6 dofs and CAN NOT be used in 2D domain.

See also:

Notes

ShellDKGQ

This command is used to construct a ShellDKGQ element object, which is a quadrilateral shell element based on the theory of generalized conforming element.

element(*'ShellDKGQ'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. Currently can be a `'PlateFiberSection'`, a `'ElasticMembranePlateSection'` and a `'LayeredShell'` section

See also:

Notes

ShellDKGT

This command is used to construct a ShellDKGT element object, which is a triangular shell element based on the theory of generalized conforming element.

element(*'ShellDKGT'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of three element nodes in clockwise or counter-clockwise order
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. currently can be a `'PlateFiberSection'`, a `'ElasticMembranePlateSection'` and a `'LayeredShell'` section

See also:

Notes

ShellNLDKGQ

This command is used to construct a ShellNLDKGQ element object accounting for the geometric nonlinearity of large deformation using the updated Lagrangian formula, which is developed based on the ShellDKGQ element.

element (*'ShellNLDKGQ'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. currently can be a `'PlateFiberSection'`, a `'ElasticMembranePlateSection'` and a `'LayeredShell'` section

See also:

Notes

ShellNLDKGT

This command is used to construct a ShellNLDKGT element object accounting for the geometric nonlinearity of large deformation using the updated Lagrangian formula, which is developed based on the ShellDKGT element.

element (*'ShellNLDKGT'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of three element nodes in clockwise or counter-clockwise order around the element
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. currently can be a `'PlateFiberSection'`, a `'ElasticMembranePlateSection'` and a `'LayeredShell'` section

See also:

Notes

ShellNL

element (*'ShellNL'*, *eleTag*, **eleNodes*, *secTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of nine element nodes, input is the typical, firstly four corner nodes counter-clockwise, then mid-side nodes counter-clockwise and finally the central node
`secTag` (int)	tag associated with previously-defined SectionForceDeformation object. currently can be a `'PlateFiberSection'`, a `'ElasticMembranePlateSection'` and a `'LayeredShell'` section

See also:

Notes

Bbar Plane Strain Quadrilateral Element

This command is used to construct a four-node quadrilateral element object, which uses a bilinear isoparametric formulation along with a mixed volume/pressure B-bar assumption. This element is for plane strain problems only.

element (*'bbarQuad'*, *eleTag*, **eleNodes*, *thick*, *matTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	element thickness
`matTag` (int)	tag of nDMaterial

Note:

1. PlainStrain only.
2. The valid queries to a Quad element when creating an ElementRecorder object are 'forces', 'stresses,' and 'material $matNum matArg1 matArg2 ...' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the isoparametric domain.

See also:

Notes

Enhanced Strain Quadrilateral Element

This command is used to construct a four-node quadrilateral element, which uses a bilinear isoparametric formulation with enhanced strain modes.

element (*'enhancedQuad'*, *eleTag*, **eleNodes*, *thick*, *type*, *matTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	element thickness
`type` (str)	string representing material behavior. Valid options depend on the NDMaterial object and its available material formulations. The type parameter can be either `'PlaneStrain'` or `'PlaneStress'`
`matTag` (int)	tag of nDMaterial

See also:

Notes

SSPquad Element

This command is used to construct a SSPquad element object.

element (*'SSPquad'*, *eleTag*, **eleNodes*, *matTag*, *type*, *thick*[, *b1*, *b2*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	thickness of the element in out-of-plane direction
`type` (str)	string to relay material behavior to the element, can be either `'PlaneStrain'` or `'PlaneStress'`
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`b1` `b2` (float)	constant body forces in global x- and y-directions, respectively (optional, default = 0.0)

The SSPquad element is a four-node quadrilateral element using physically stabilized single-point integration (SSP –> Stabilized Single Point). The stabilization incorporates an assumed strain field in which the volumetric dilation and the shear strain associated with the the hourglass modes are zero, resulting in an element which is free from volumetric and shear locking. The elimination of shear locking results in greater coarse mesh accuracy in bending dominated problems, and the elimination of volumetric locking improves accuracy in nearly-incompressible problems. Analysis times are generally faster than corresponding full integration elements. The formulation for this element is identical to the solid phase portion of the SSPquadUP element as described by McGann et al. (2012).

Note:

1. Valid queries to the SSPquad element when creating an ElementalRecorder object correspond to those for the nDMaterial object assigned to the element (e.g., 'stress', 'strain'). Material response is recorded at the single integration point located in the center of the element.
2. The SSPquad element was designed with intentions of duplicating the functionality of the Quad Element. If an example is found where the SSPquad element cannot do something that works for the Quad Element, e.g., material updating, please contact the developers listed below so the bug can be fixed.

See also:

Notes

Tri31 Element

This command is used to construct a constant strain triangular element (Tri31) which uses three nodes and one integration points.

element (*'Tri31'*, *eleTag*, **eleNodes*, *thick*, *type*, *matTag*[, *pressure*, *rho*, *b1*, *b2*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of three element nodes in counter-clockwise order
thick (float)	element thickness
type (str)	string representing material behavior. The type parameter can be either 'PlaneStrain' or 'PlaneStress'
matTag (int)	tag of nDMaterial
pressure (float)	surface pressure (optional, default = 0.0)
rho (float)	element mass density (per unit volume) from which a lumped element mass matrix is computed (optional, default=0.0)
b1 b2 (float)	constant body forces defined in the domain (optional, default=0.0)

Note:

1. Consistent nodal loads are computed from the pressure and body forces.
2. The valid queries to a Tri31 element when creating an ElementRecorder object are 'forces', 'stresses,' and 'material $matNum matArg1 matArg2 ...' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the domain.

See also:

Notes

Standard Brick Element

This element is used to construct an eight-node brick element object, which uses a trilinear isoparametric formulation.

element (*'stdBrick'*, *eleTag*, **eleNodes*, *matTag*[, *b1*, *b2*, *b3*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of eight element nodes in bottom and top faces and in counter-clockwise order
matTag (int)	tag of nDMaterial
b1 b2 b3 (float)	body forces in global x,y,z directions

Note:

1. The valid queries to a Brick element when creating an ElementRecorder object are 'forces', 'stresses,' ('strains' version > 2.2.0) and 'material $matNum matArg1 matArg2 ...' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the isoparametric domain.
2. This element can only be defined in -ndm 3 -ndf 3

See also:

Notes

Bbar Brick Element

This command is used to construct an eight-node mixed volume/pressure brick element object, which uses a trilinear isoparametric formulation.

element (*'bbarBrick'*, *eleTag*, **eleNodes*, *matTag*[, *b1*, *b2*, *b3*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of eight element nodes in bottom and top faces and in counter-clockwise order
`matTag` (int)	tag of nDMaterial
`b1 b2 b3` (float)	body forces in global x,y,z directions

Note:

1. Node numbering for this element is different from that for the eight-node brick (Brick8N) element.
2. The valid queries to a Quad element when creating an ElementRecorder object are 'forces', 'stresses', 'strains', and 'material $matNum matArg1 matArg2 . . . ' Where $matNum refers to the material object at the integration point corresponding to the node numbers in the isoparametric domain.

See also:

Notes

Twenty Node Brick Element

The element is used to construct a twenty-node three dimensional element object

element (*'Brick20N'*, *eleTag*, **eleNodes*, *matTag*, *bf1*, *bf2*, *bf3*, *massDen*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of twenty element nodes, input order is shown in notes below
`matTag` (int)	material tag associated with previsouly-defined NDMaterial object
`bf1 bf2 bf3` (float)	body force in the direction of global coordinates x, y and z
`massDen` (float)	mass density (mass/volume)

Note: The valid queries to a Brick20N element when creating an ElementRecorder object are 'force,' 'stiffness,' stress', 'gausspoint' or 'plastic'. The output is given as follows:

1. 'stress'

   the six stress components from each Gauss points are output by the order: sigma_xx, sigma_yy, sigma_zz, sigma_xy, sigma_xz,sigma_yz

2. 'gausspoint'

   the coordinates of all Gauss points are printed out

3. 'plastic'

   the equivalent deviatoric plastic strain from each Gauss point is output in the same order as the coordinates are printed

---

See also:

Notes

SSPbrick Element

This command is used to construct a SSPbrick element object.

element (*'SSPbrick'*, *eleTag*, **eleNodes*, *matTag*[, *b1*, *b2*, *b3*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of eight element nodes in bottom and top faces and in counter-clockwise order
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`b1` `b2` `b3` (float)	constant body forces in global x-, y-, and z-directions, respectively (optional, default = 0.0)

The SSPbrick element is an eight-node hexahedral element using physically stabilized single-point integration (SSP –> Stabilized Single Point). The stabilization incorporates an enhanced assumed strain field, resulting in an element which is free from volumetric and shear locking. The elimination of shear locking results in greater coarse mesh accuracy in bending dominated problems, and the elimination of volumetric locking improves accuracy in nearly-incompressible problems. Analysis times are generally faster than corresponding full integration elements.

Note:

1. Valid queries to the SSPbrick element when creating an ElementalRecorder object correspond to those for the nDMaterial object assigned to the element (e.g., 'stress', 'strain'). Material response is recorded at the single integration point located in the center of the element.
2. The SSPbrick element was designed with intentions of duplicating the functionality of the stdBrick Element. If an example is found where the SSPbrick element cannot do something that works for the stdBrick Element, e.g., material updating, please contact the developers listed below so the bug can be fixed.

See also:

Notes

FourNodeTetrahedron

This command is used to construct a standard four-node tetrahedron element objec with one-point Gauss integration.

element (*'FourNodeTetrahedron'*, *eleTag*, **eleNodes*, *matTag*[, *b1*, *b2*, *b3*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes
`matTag` (int)	tag of nDMaterial
`b1` `b2` `b3` (float)	body forces in global x,y,z directions

See also:

Notes

Four Node Quad u-p Element

FourNodeQuadUP is a four-node plane-strain element using bilinear isoparametric formulation. This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium. Each element node has 3 degrees-of-freedom (DOF): DOF 1 and 2 for solid displacement (u) and DOF 3 for fluid pressure (p).

element (*'', eleTag, *eleNodes, thick, matTag, bulk, fmass, hPerm, vPerm*[, *b1=0, b2=0, t=0*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	Element thickness
`matTag` (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
`bulk` (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
`fmass` (float)	Fluid mass density
`hPerm, vPerm` (float)	Permeability coefficient in horizontal and vertical directions respectively.
`b1, b2` (float)	Optional gravity acceleration components in horizontal and vertical directions respectively (defaults are 0.0)
`t` (float)	Optional uniform element normal traction, positive in tension (default is 0.0)

See also:

Notes

Brick u-p Element

BrickUP is an 8-node hexahedral linear isoparametric element. Each node has 4 degrees-of-freedom (DOF): DOFs 1 to 3 for solid displacement (u) and DOF 4 for fluid pressure (p). This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium.

element (*'brickUP', eleTag, *eleNodes, matTag, bulk, fmass, PermX, PermY, PermZ*[, *bX=0, bY=0, bZ=0*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of eight element nodes
`matTag` (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
`bulk` (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
`fmass` (float)	Fluid mass density
`permX`, `permY`, `permZ` (float)	Permeability coefficients in x, y, and z directions respectively.
`bX`, `bY`, `bZ` (float)	Optional gravity acceleration components in x, y, and z directions directions respectively (defaults are 0.0)

See also:

Notes

BbarQuad u-p Element

bbarQuadUP is a four-node plane-strain mixed volume/pressure element, which uses a tri-linear isoparametric formulation. This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium. Each element node has 3 degrees-of-freedom (DOF): DOF 1 and 2 for solid displacement (u) and DOF 3 for fluid pressure (p).

element (*'bbarQuadUP'*, *eleTag*, **eleNodes*, *thick*, *matTag*, *bulk*, *fmass*, *hPerm*, *vPerm*[, *b1=0*, *b2=0*, *t=0*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`thick` (float)	Element thickness
`matTag` (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
`bulk` (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
`fmass` (float)	Fluid mass density
`hPerm, vPerm` (float)	Permeability coefficient in horizontal and vertical directions respectively.
`b1, b2` (float)	Optional gravity acceleration components in horizontal and vertical directions respectively (defaults are 0.0)
`t` (float)	Optional uniform element normal traction, positive in tension (default is 0.0)

See also:

Notes

BbarBrick u-p Element

bbarBrickUP is a 8-node mixed volume/pressure element, which uses a tri-linear isoparametric formulation.

Each node has 4 degrees-of-freedom (DOF): DOFs 1 to 3 for solid displacement (u) and DOF 4 for fluid pressure (p). This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium.

element(*'bbarBrickUP'*, *eleTag*, **eleNodes*, *matTag*, *bulk*, *fmass*, *PermX*, *PermY*, *PermZ*[, *bX=0*, *bY=0*, *bZ=0*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of eight element nodes
matTag (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
bulk (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
fmass (float)	Fluid mass density
permX, permY, permZ (float)	Permeability coefficients in x, y, and z directions respectively.
bX, bY, bZ (float)	Optional gravity acceleration components in x, y, and z directions directions respectively (defaults are 0.0)

See also:

Notes

Nine Four Node Quad u-p Element

Nine_Four_Node_QuadUP is a 9-node quadrilateral plane-strain element. The four corner nodes have 3 degrees-of-freedom (DOF) each: DOF 1 and 2 for solid displacement (u) and DOF 3 for fluid pressure (p). The other five nodes have 2 DOFs each for solid displacement. This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium.

element (*'9_4_QuadUP'*, *eleTag*, **eleNodes*, *thick*, *matTag*, *bulk*, *fmass*, *hPerm*, *vPerm*[, *b1=0*, *b2=0*])

eleTag (int)	unique element object tag
eleNodes (list (int))	a list of nine element nodes
thick (float)	Element thickness
matTag (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
bulk (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
fmass (float)	Fluid mass density
hPerm, vPerm (float)	Permeability coefficient in horizontal and vertical directions respectively.
b1, b2 (float)	Optional gravity acceleration components in horizontal and vertical directions respectively (defaults are 0.0)

See also:

Notes

Twenty Eight Node Brick u-p Element

wenty_Eight_Node_BrickUP is a 20-node hexahedral isoparametric element.

The eight corner nodes have 4 degrees-of-freedom (DOF) each: DOFs 1 to 3 for solid displacement (u) and DOF 4 for fluid pressure (p). The other nodes have 3 DOFs each for solid displacement. This element is implemented for simulating dynamic response of solid-fluid fully coupled material, based on Biot's theory of porous medium.

element (*'bbarBrickUP'*, *eleTag*, **eleNodes*, *matTag*, *bulk*, *fmass*, *PermX*, *PermY*, *PermZ*[, *bX=0*, *bY=0*, *bZ=0*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of twenty element nodes
`matTag` (int)	Tag of an NDMaterial object (previously defined) of which the element is composed
`bulk` (float)	Combined undrained bulk modulus Bc relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2×10^6 kPa (or 3.191×10^5 psi) for water), and n the initial porosity.
`fmass` (float)	Fluid mass density
`permX`, `permY`, `permZ` (float)	Permeability coefficients in x, y, and z directions respectively.
`bX`, `bY`, `bZ` (float)	Optional gravity acceleration components in x, y, and z directions directions respectively (defaults are 0.0)

See also:

Notes

SSPquadUP Element

This command is used to construct a SSPquadUP element object.

element (*'SSPquadUP'*, *eleTag*, **eleNodes*, *matTag*, *thick*, *fBulk*, *fDen*, *k1*, *k2*, *void*, *alpha*[, *b1*, *b2*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of four element nodes in counter-clockwise order
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`thick` (float)	thickness of the element in out-of-plane direction
`fBulk` (float)	bulk modulus of the pore fluid
`fDen` (float)	mass density of the pore fluid
`k1` `k2` (float)	permeability coefficients in global x- and y-directions, respectively
`void` (float)	voids ratio
`alpha` (float)	spatial pressure field stabilization parameter (see discussion below for more information)
`b1` `b2` (float)	constant body forces in global x- and y-directions, respectively (optional, default = 0.0) - See Note 3

> The SSPquadUP element is an extension of the SSPquad Element for use in dynamic plane strain analysis of fluid saturated porous media. A mixed displacement-pressure (u-p) formulation is used, based upon the work of Biot as extended by Zienkiewicz and Shiomi (1984).

The physical stabilization necessary to allow for reduced integration incorporates an assumed strain field in which the volumetric dilation and the shear strain associated with the the hourglass modes are zero, resulting in an element which is free from volumetric and shear locking. The elimination of shear locking results in greater coarse mesh accuracy in bending dominated problems, and the elimination of volumetric locking improves accuracy in nearly-incompressible problems. Analysis times are generally faster than corresponding full integration elements.

Equal-order interpolation is used for the displacement and pressure fields, thus, the SSPquadUP element does not inherently pass the inf-sup condition, and is not fully acceptable in the incompressible-impermeable limit (the QuadUP Element has the same issue). A stabilizing parameter is employed to permit the use of equal-order interpolation for the SSPquadUP element. This parameter $alpha can be computed as

$$\alpha = 0.25 * (h^2)/(den * c^2)$$

where h is the element size, c is the speed of elastic wave propagation in the solid phase, and den is the mass density of the solid phase. The $alpha parameter should be a small number. With a properly defined $alpha parameter, the SSPquadUP element can produce comparable results to a higher-order element such as the 9_4_QuadUP Element at a significantly lower computational cost and with a greater ease in mesh generation.

The full formulation for the SSPquadUP element can be found in McGann et al. (2012) along with several example applications.

Note:

1. The SSPquadUP element will only work in dynamic analysis.
2. For saturated soils, the mass density input into the associated nDMaterial object should be the saturated mass density.
3. When modeling soil, the body forces input into the SSPquadUP element should be the components of the gravitational vector, not the unit weight.
4. Fixing the pore pressure degree-of-freedom (dof 3) at a node is a drainage boundary condition at which zero pore pressure will be maintained throughout the analysis. Leaving the third dof free allows pore pressures to build at that node.
5. Valid queries to the SSPquadUP element when creating an ElementalRecorder object correspond to those for the nDMaterial object assigned to the element (e.g., 'stress', 'strain'). Material response is recorded at the single integration point located in the center of the element.

6. The SSPquadUP element was designed with intentions of duplicating the functionality of the QuadUP Element. If an example is found where the SSPquadUP element cannot do something that works for the QuadUP Element, e.g., material updating, please contact the developers listed below so the bug can be fixed.

See also:

Notes

SSPbrickUP Element

This command is used to construct a SSPbrickUP element object.

element (*'SSPbrickUP'*, *eleTag*, **eleNodes*, *matTag*, *fBulk*, *fDen*, *k1*, *k2*, *k3*, *void*, *alpha*[, *b1*, *b2*, *b3*])

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	a list of eight element nodes in counter-clockwise order
`matTag` (float)	unique integer tag associated with previously-defined nDMaterial object
`fBulk` (float)	bulk modulus of the pore fluid
`fDen` (float)	mass density of the pore fluid
`k1 k2 k3` (float)	permeability coefficients in global x-, y-, and z-directions, respectively
`void` (float)	voids ratio
`alpha` (float)	spatial pressure field stabilization parameter (see discussion below for more information)
`b1 b2 b3` (float)	constant body forces in global x-, y-, and z-directions, respectively (optional, default = 0.0) - See Note 3

The SSPbrickUP element is an extension of the SSPbrick Element for use in dynamic 3D analysis of fluid saturated porous media. A mixed displacement-pressure (u-p) formulation is used, based upon the work of Biot as extended by Zienkiewicz and Shiomi (1984).

The physical stabilization necessary to allow for reduced integration incorporates an enhanced assumed strain field, resulting in an element which is free from volumetric and shear locking. The elimination of shear locking results in greater coarse mesh accuracy in bending dominated problems, and the elimination of volumetric locking improves accuracy in nearly-incompressible problems. Analysis times are generally faster than corresponding full integration elements.

Equal-order interpolation is used for the displacement and pressure fields, thus, the SSPbrickUP element does not inherently pass the inf-sup condition, and is not fully acceptable in the incompressible-impermeable limit (the brickUP Element has the same issue). A stabilizing parameter is employed to permit the use of equal-order interpolation for the SSPbrickUP element. This parameter $alpha can be computed as

$$\alpha = h^2/(4 * (K_s + (4/3) * G_s))$$

where h is the element size, and K_s and G_s are the bulk and shear moduli for the solid phase. The α parameter should be a small number. With a properly defined α parameter, the SSPbrickUP element can produce comparable results to a higher-order element such as the 20_8_BrickUP Element at a significantly lower computational cost and with a greater ease in mesh generation.

Note:

1. The SSPbrickUP element will only work in dynamic analysis.

2. For saturated soils, the mass density input into the associated nDMaterial object should be the saturated mass density.

3. When modeling soil, the body forces input into the SSPbrickUP element should be the components of the gravitational vector, not the unit weight.

4. Fixing the pore pressure degree-of-freedom (dof 4) at a node is a drainage boundary condition at which zero pore pressure will be maintained throughout the analysis. Leaving the fourth dof free allows pore pressures to build at that node.

5. Valid queries to the SSPbrickUP element when creating an ElementalRecorder object correspond to those for the nDMaterial object assigned to the element (e.g., 'stress', 'strain'). Material response is recorded at the single integration point located in the center of the element.

6. The SSPbrickUP element was designed with intentions of duplicating the functionality of the brickUP Element. If an example is found where the SSPbrickUP element cannot do something that works for the brickUP Element, e.g., material updating, please contact the developers listed below so the bug can be fixed.

See also:

Notes

SimpleContact2D

This command is used to construct a SimpleContact2D element object.

element (*'SimpleContact2D'*, *eleTag*, *iNode*, *jNode*, *sNode*, *lNode*, *matTag*, *gTol*, *fTol*)

`eleTag` (int)	unique element object tag
`iNode jNode` (int)	master nodes (-ndm 2 -ndf 2)
`sNode` (int)	slave node (-ndm 2 -ndf 2)
`lNode` (int)	Lagrange multiplier node (-ndm 2 -ndf 2)
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`gTol` (float)	gap tolerance
`fTol` (float)	force tolerance

The SimpleContact2D element is a two-dimensional node-to-segment contact element which defines a frictional contact interface between two separate bodies. The master nodes are the nodes which define the endpoints of a line segment on the first body, and the slave node is a node from the second body. The Lagrange multiplier node is required to enforce the contact condition. This node should not be shared with any other element in the domain. Information on the theory behind this element can be found in, e.g. Wriggers (2002).

Note:

1. The SimpleContact2D element has been written to work exclusively with the ContactMaterial2D nDMaterial object.

2. The valid recorder queries for this element are:

   (a) force - returns the contact force acting on the slave node in vector form.

   (b) frictionforce - returns the frictional force acting on the slave node in vector form.

   (c) forcescalar - returns the scalar magnitudes of the normal and tangential contact forces.

   (d) The SimpleContact2D elements are set to consider frictional behavior as a default, but the frictional state of the SimpleContact2D element can be changed from the input file using the setParameter command. When updating, value of 0 corresponds to the frictionless condition, and a value of 1 signifies the inclusion of friction. An example command for this update procedure is provided below

3. The SimpleContact2D element works well in static and pseudo-static analysis situations.
4. In transient analysis, the presence of the contact constraints can effect the stability of commonly-used time integration methods in the HHT or Newmark family (e.g., Laursen, 2002). For this reason, use of alternative time-integration methods which numerically damp spurious high frequency behavior may be required. The TRBDF2 integrator is an effective method for this purpose. The Newmark integrator can also be effective with proper selection of the gamma and beta coefficients. The trapezoidal rule, i.e., Newmark with gamma = 0.5 and beta = 0.25, is particularly prone to instability related to the contact constraints and is not recommended.

See also:

Notes

SimpleContact3D

This command is used to construct a SimpleContact3D element object.

element (*'SimpleContact3D'*, *eleTag*, *iNode*, *jNode*, *kNode*, *lNode*, *sNode*, *LNode*, *matTag*, *gTol*, *fTol*)

`eleTag` (int)	unique element object tag
`iNode jNode kNode lNode` (int)	master nodes (-ndm 3 -ndf 3)
`sNode` (int)	slave node (-ndm 3 -ndf 3)
`LNode` (int)	Lagrange multiplier node (-ndm 3 -ndf 3)
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`gTol` (float)	gap tolerance
`fTol` (float)	force tolerance

The SimpleContact3D element is a three-dimensional node-to-surface contact element which defines a frictional contact interface between two separate bodies. The master nodes are the nodes which define a surface of a hexahedral element on the first body, and the slave node is a node from the second body. The Lagrange multiplier node is required to enforce the contact condition. This node should not be shared with any other element in the domain. Information on the theory behind this element can be found in, e.g. Wriggers (2002).

Note:

1. The SimpleContact3D element has been written to work exclusively with the ContactMaterial3D nDMaterial object.
2. The valid recorder queries for this element are:
   (a) force - returns the contact force acting on the slave node in vector form.
   (b) frictionforce - returns the frictional force acting on the slave node in vector form.
   (c) forcescalar - returns the scalar magnitudes of the single normal and two tangential contact forces.
   (d) The SimpleContact3D elements are set to consider frictional behavior as a default, but the frictional state of the SimpleContact3D element can be changed from the input file using the setParameter command.

When updating, value of 0 corresponds to the frictionless condition, and a value of 1 signifies the inclusion of friction. An example command for this update procedure is provided below

3. The SimpleContact3D element works well in static and pseudo-static analysis situations.
4. In transient analysis, the presence of the contact constraints can effect the stability of commonly-used time integration methods in the HHT or Newmark family (e.g., Laursen, 2002). For this reason, use of alternative time-integration methods which numerically damp spurious high frequency behavior may be required. The TRBDF2 integrator is an effective method for this purpose. The Newmark integrator can also be effective with proper selection of the gamma and beta coefficients. The trapezoidal rule, i.e., Newmark with gamma = 0.5 and beta = 0.25, is particularly prone to instability related to the contact constraints and is not recommended.

See also:

Notes

BeamContact2D

This command is used to construct a BeamContact2D element object.

element (*'BeamContact2D'*, *eleTag*, *matTag*, *width*, *gTol*, *fTol*[, *cFlag*])

`eleTag` (int)	unique element object tag
`iNode` `jNode` (int)	master nodes (-ndm 2 -ndf 3)
`sNode` (int)	slave node (-ndm 2 -ndf 2)
`lNode` (int)	Lagrange multiplier node (-ndm 2 -ndf 2)
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`width` (float)	the width of the wall represented by the beam element in plane strain
`gTol` (float)	gap tolerance
`fTol` (float)	force tolerance
`cFlag` (int)	optional initial contact flag `cFlag` = 0 >> contact between bodies is initially assumed (DEFAULT) `cFlag` = 1 >> no contact between bodies is initially assumed

The BeamContact2D element is a two-dimensional beam-to-node contact element which defines a frictional contact interface between a beam element and a separate body. The master nodes (3 DOF) are the endpoints of the beam element, and the slave node (2 DOF) is a node from a second body. The Lagrange multiplier node (2 DOF) is required to enforce the contact condition. Each contact element should have a unique Lagrange multiplier node. The Lagrange multiplier node should not be fixed, otherwise the contact condition will not work.

Under plane strain conditions in 2D, a beam element represents a unit thickness of a wall. The width is the dimension of this wall in the 2D plane. This width should be built-in to the model to ensure proper enforcement of the contact condition. The Excavation Supported by Cantilevered Sheet Pile Wall practical example provides some further examples and discussion on the usage of this element.

Note:

1. The BeamContact2D element has been written to work exclusively with the ContactMaterial2D nDMaterial object.
2. The valid recorder queries for this element are:
   (a) force - returns the contact force acting on the slave node in vector form.
   (b) frictionforce - returns the frictional force acting on the slave node in vector form.
   (c) forcescalar - returns the scalar magnitudes of the normal and tangential contact forces.
   (d) masterforce - returns the reactions (forces and moments) acting on the master nodes.
   (e) The BeamContact2D elements are set to consider frictional behavior as a default, but the frictional state of the BeamContact2D element can be changed from the input file using the setParameter command. When updating, value of 0 corresponds to the frictionless condition, and a value of 1 signifies the inclusion of friction. An example command for this update procedure is provided below
3. The BeamContact2D element works well in static and pseudo-static analysis situations.
4. In transient analysis, the presence of the contact constraints can effect the stability of commonly-used time integration methods in the HHT or Newmark family (e.g., Laursen, 2002). For this reason, use of alternative time-integration methods which numerically damp spurious high frequency behavior may be required. The TRBDF2 integrator is an effective method for this purpose. The Newmark integrator can also be effective with proper selection of the gamma and beta coefficients. The trapezoidal rule, i.e., Newmark with gamma = 0.5 and beta = 0.25, is particularly prone to instability related to the contact constraints and is not recommended.

See also:

Notes

BeamContact3D

This command is used to construct a BeamContact3D element object.

element (*'BeamContact3D'*, *eleTag*, *iNode*, *jNode*, *sNode*, *lNode*, *radius*, *crdTransf*, *matTag*, *gTol*, *fTol*[, *cFlag*])

`eleTag` (int)	unique element object tag
`iNode` `jNode` (int)	master nodes (-ndm 3 -ndf 6)
`sNode` (int)	slave node (-ndm 3 -ndf 3)
`lNode` (int)	Lagrange multiplier node (-ndm 3 -ndf 3)
`radius` (float)	constant radius of circular beam associated with beam element
`crdTransf` (int)	unique integer tag associated with previously-defined geometricTransf object
`matTag` (int)	unique integer tag associated with previously-defined nDMaterial object
`gTol` (float)	gap tolerance
`fTol` (float)	force tolerance
`cFlag` (int)	optional initial contact flag `cFlag` = 0 >> contact between bodies is initially assumed (DEFAULT) `cFlag` = 1 >> no contact between bodies is initially assumed

The BeamContact3D element is a three-dimensional beam-to-node contact element which defines a frictional contact interface between a beam element and a separate body. The master nodes (6 DOF) are the endpoints of the beam element, and the slave node (3 DOF) is a node from a second body. The Lagrange multiplier node (3 DOF) is required to enforce the contact condition. Each contact element should have a unique Lagrange multiplier node. The Lagrange multiplier node should not be fixed, otherwise the contact condition will not work.

Note:

1. The BeamContact3D element has been written to work exclusively with the ContactMaterial3D nDMaterial object.
2. The valid recorder queries for this element are:
   (a) force - returns the contact force acting on the slave node in vector form.
   (b) frictionforce - returns the frictional force acting on the slave node in vector form.
   (c) forcescalar - returns the scalar magnitudes of the single normal and two tangential contact forces.
   (d) masterforce - returns the reactions (forces only) acting on the master nodes.
   (e) mastermoment - returns the reactions (moments only) acting on the master nodes.
   (f) masterreaction - returns the full reactions (forces and moments) acting on the master nodes.
   (g) The BeamContact3D elements are set to consider frictional behavior as a default, but the frictional state of the BeamContact3D element can be changed from the input file using the setParameter command. When updating, value of 0 corresponds to the frictionless condition, and a value of 1 signifies the inclusion of friction. An example command for this update procedure is provided below
3. The BeamContact3D element works well in static and pseudo-static analysis situations.
4. In transient analysis, the presence of the contact constraints can effect the stability of commonly-used time integration methods in the HHT or Newmark family (e.g., Laursen, 2002). For this reason, use of alternative time-integration methods which numerically damp spurious high frequency behavior may be required. The

TRBDF2 integrator is an effective method for this purpose. The Newmark integrator can also be effective with proper selection of the gamma and beta coefficients. The trapezoidal rule, i.e., Newmark with gamma = 0.5 and beta = 0.25, is particularly prone to instability related to the contact constraints and is not recommended.

See also:

Notes

BeamEndContact3D

This command is used to construct a BeamEndContact3D element object.

element (*'BeamEndContact3D'*, *eleTag*, *iNode*, *jNode*, *sNode*, *lNode*, *radius*, *gTol*, *fTol*[, *cFlag*])

`eleTag` (int)	unique element object tag
`iNode` (int)	master node from the beam (-ndm 3 -ndf 6)
`jNode` (int)	the remaining node on the beam element with `iNode` (-ndm 3 -ndf 6)
`sNode` (int)	slave node (-ndm 3 -ndf 3)
`lNode` (int)	Lagrange multiplier node (-ndm 3 -ndf 3)
`radius` (float)	radius of circular beam associated with beam element
`gTol` (float)	gap tolerance
`fTol` (float)	force tolerance
`cFlag` (float)	optional initial contact flag `cFlag` = 0 >> contact between bodies is initially assumed (DEFAULT) `cFlag1` = 1 >> no contact between bodies is initially assumed

The BeamEndContact3D element is a node-to-surface contact element which defines a normal contact interface between the end of a beam element and a separate body. The first master node ($iNode) is the beam node which is at the end of the beam (i.e. only connected to a single beam element), the second node ($jNode) is the remaining node on the beam element in question. The slave node is a node from a second body. The Lagrange multiplier node is required to enforce the contact condition. This node should not be shared with any other element in the domain, and should be created with the same number of DOF as the slave node.

The BeamEndContact3D element enforces a contact condition between a fictitious circular plane associated with a beam element and a node from a second body. The normal direction of the contact plane coincides with the endpoint tangent of the beam element at the master beam node ($iNode). The extents of this circular plane are defined by the radius input parameter. The master beam node can only come into contact with a slave node which is within the extents of the contact plane. There is a lag step associated with changing between the 'in contact' and 'not in contact' conditions.

This element was developed for use in establishing a contact condition for the tip of a pile modeled as using beam elements and the underlying soil elements in three-dimensional analysis.

Note:

1. The BeamEndContact3D element does not use a material object.
2. The valid recorder queries for this element are:
   (a) force - returns the contact force acting on the slave node in vector form.
   (b) masterforce - returns the reactions (forces and moments) acting on the master node.
   (c) The BeamEndContact3D element works well in static and pseudo-static analysis situations.
3. In transient analysis, the presence of the contact constraints can effect the stability of commonly-used time integration methods in the HHT or Newmark family (e.g., Laursen, 2002). For this reason, use of alternative time-integration methods which numerically damp spurious high frequency behavior may be required. The TRBDF2 integrator is an effective method for this purpose. The Newmark integrator can also be effective with proper selection of the gamma and beta coefficients. The trapezoidal rule, i.e., Newmark with gamma = 0.5 and beta = 0.25, is particularly prone to instability related to the contact constraints and is not recommended.

See also:

Notes

CatenaryCableElement

This command is used to construct a catenary cable element object.

element (*'CatenaryCable'*, *eleTag*, *iNode*, *jNode*, *weight*, *E*, *A*, *L0*, *alpha*, *temperature_change*, *rho*, *errorTol*, *Nsubsteps*, *massType*)

`eleTag` (int)	unique element object tag
`iNode jNode` (int)	end nodes (3 dof per node)
`E` (float)	elastic modulus of the cable material
`A` (float)	cross-sectional area of element
`L0` (float)	unstretched length of the cable
`alpha` (float)	coefficient of thermal expansion
`temperature_change` (float)	temperature change for the element
`rho` (float)	mass per unit length
`errortol` (float)	allowed tolerance for within-element equilbrium (Newton-Rhapson iterations)
`Nsubsteps` (int)	number of within-element substeps into which equilibrium iterations are subdivided (not number of steps to convergence)
`massType` (int)	Mass matrix model to use (`massType` = 0 lumped mass matrix, `massType` = 1 rigid-body mass matrix (in development))

This cable is a flexibility-based formulation of the catenary cable. An iterative scheme is used internally to compute equilibrium. At each iteration, node i is considered fixed while node j is free. End-forces are applied at node-j and its displacements computed. Corrections to these forces are applied iteratively using a Newton-Rhapson scheme (with optional sub-stepping via $Nsubsteps) until nodal displacements are within the provided tolerance ($errortol). When convergence is reached, a stiffness matrix is computed by inversion of the flexibility matrix and rigid-body mode injection.

Note:

1. The stiffness of the cable comes from the large-deformation interaction between loading and cable shape. Therefore, all cables must have distributed forces applied to them. See example. Should not work for only nodal forces.

2. Valid queries to the CatenaryCable element when creating an ElementalRecorder object correspond to 'forces', which output the end-forces of the element in global coordinates (3 for each node).
3. Only the lumped-mass formulation is currently available.
4. The element does up 100 internal iterations. If convergence is not achieved, will result in error and some diagnostic information is printed out.

See also:

Notes

SurfaceLoad Element

This command is used to construct a SurfaceLoad element object.

element (*'SurfaceLoad'*, *eleTag*, **eleNodes*, *p*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	the four nodes defining the element, input in counterclockwise order (-ndm 3 -ndf 3)
`p` (float)	applied pressure loading normal to the surface, outward is positive, inward is negative

The SurfaceLoad element is a four-node element which can be used to apply surface pressure loading to 3D brick elements. The SurfaceLoad element applies energetically-conjugate forces corresponding to the input scalar pressure to the nodes associated with the element. As these nodes are shared with a 3D brick element, the appropriate nodal loads are therefore applied to the brick.

Note:

1. There are no valid ElementalRecorder queries for the SurfaceLoad element. Its sole purpose is to apply nodal forces to the adjacent brick element.
2. The pressure loading from the SurfaceLoad element can be applied in a load pattern. See the analysis example below.

See also:

Notes

VS3D4

This command is used to construct a four-node 3D viscous-spring boundary quad element object based on a bilinear isoparametric formulation.

element (*'VS3D4'*, *eleTag*, **eleNodes*, *E*, *G*, *rho*, *R*, *alphaN*, *alphaT*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	4 end nodes
`E` (float)	Young's Modulus of element material
`G` (float)	Shear Modulus of element material
`rho` (float)	Mass Density of element material
`R` (float)	distance from the scattered wave source to the boundary
`alphaN` (float)	correction parameter in the normal direction
`alphaT` (float)	correction parameter in the tangential direction

Note: Reference: Liu J, Du Y, Du X, et al. 3D viscous-spring artificial boundary in time domain. Earthquake Engineering and Engineering Vibration, 2006, 5(1):93-102

See also:

Notes

AC3D8

This command is used to construct an eight-node 3D brick acoustic element object based on a trilinear isoparametric formulation.

element(*'AC3D8'*, *eleTag*, **eleNodes*, *matTag*)

`eleTag` (int)	unique element object tag
`eleNdoes` (list (int))	8 end nodes
`matTag` (int)	Material Tag of previously defined nD material

Note: Reference: ABAQUS theory manual. (2.9.1 Coupled acoustic-structural medium analysis)

See also:

Notes

ASI3D8

This command is used to construct an eight-node zero-thickness 3D brick acoustic-structure interface element object based on a bilinear isoparametric formulation. The nodes in the acoustic domain share the same coordinates with the nodes in the solid domain.

element(*'ASI3D8'*, *eleTag*, **eleNodes1*, **eleNodes2*)

`eleTag` (int)	unique element object tag
`*eleNodes`` (list (int))	four nodes defining structure domain of element boundaries
`*eleNodes2` (list (int))	four nodes defining acoustic domain of element boundaries

Note: Reference: ABAQUS theory manual. (2.9.1 Coupled acoustic-structural medium analysis)

See also:

Notes

AV3D4

This command is used to construct a four-node 3D acoustic viscous boundary quad element object based on a bilinear isoparametric formulation.

element (*'AV3D4'*, *eleTag*, **eleNodes*, *matTag*)

`eleTag` (int)	unique element object tag
`eleNodes` (list (int))	4 end nodes
`matTag` (int)	Material Tag of previously defined nD material

See also:

Notes

1.4.3 node command

node (*nodeTag*, **crds*, *'-ndf'*, *ndf*, *'-mass'*, **mass*, *'-disp'*, **disp*, *'-vel'*, **vel*, *'-accel'*, **accel*)
Create a OpenSees node.

`nodeTag` (int)	node tag.
`crds` (list (float))	nodal coordinates.
`ndf` (float)	nodal ndf. (optional)
`mass` (list (float))	nodal mass. (optional)
`vel` (list (float))	nodal velocities. (optional)
`accel` (list (float))	nodal accelerations. (optional)

1.4.4 sp constraint commands

Create constraints for a single dof of a node.

fix command

fix (*nodeTag*, **constrValues*)
Create a homogeneous SP constriant.

`nodeTag` (int)	tag of node to be constrained
`constrValues` (list (int))	a list of constraint values (0 or 1), must be preceded with `*`. • `0` free • `1` fixed

For example,

```
# fully fixed
vals = [1,1,1]
fix(nodeTag, *vals)
```

fixX command

fixX (*x*, **constrValues*, *'-tol'*, *tol=1e-10*)
Create homogeneous SP constriants.

`x` (float)	x-coordinate of nodes to be constrained
`constrValues` (list (int))	a list of constraint values (0 or 1), must be preceded with `*`. • `0` free • `1` fixed
`tol` (float)	user-defined tolerance (optional)

fixY command

fixY (*y*, **constrValues*, *'-tol'*, *tol=1e-10*)
Create homogeneous SP constriants.

`y` (float)	y-coordinate of nodes to be constrained
`constrValues` (list (int))	a list of constraint values (0 or 1), must be preceded with `*`. • `0` free • `1` fixed
`tol` (float)	user-defined tolerance (optional)

fixZ command

fixZ (*z*, **constrValues*, *'-tol'*, *tol=1e-10*)
Create homogeneous SP constriants.

`z` (float)	z-coordinate of nodes to be constrained
`constrValues` (list (int))	a list of constraint values (0 or 1), must be preceded with `*`. • `0` free • `1` fixed
`tol` (float)	user-defined tolerance (optional)

1.4.5 mp constraint commands

Create constraints for multiple dofs of multiple nodes.

equalDOF command

equalDOF(*rNodeTag*, *cNodeTag*, **dofs*)
Create a multi-point constraint between nodes.

rNodeTag (int)	integer tag identifying the retained, or master node.
cNodeTag (int)	integer tag identifying the constrained, or slave node.
dofs (list (int))	nodal degrees-of-freedom that are constrained at the cNode to be the same as those at the rNode Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

equalDOF_Mixed command

equalDOF_Mixed(*rNodeTag*, *cNodeTag*, *numDOF*, **rcdofs*)
Create a multi-point constraint between nodes.

rNodeTag (int)	integer tag identifying the retained, or master node.
cNodeTag (int)	integer tag identifying the constrained, or slave node.
numDOF (int)	number of dofs to be constrained
rcdofs (list (int))	nodal degrees-of-freedom that are constrained at the cNode to be the same as those at the rNode Valid range is from 1 through ndf, the number of nodal degrees-of-freedom. `rcdofs = [rdof1, cdof1, rdof2, cdof2, ...]`

rigidDiaphragm command

rigidDiaphragm(*perpDirn*, *rNodeTag*, **cNodeTags*)
Create a multi-point constraint between nodes. These objects will constraint certain degrees-of-freedom at the listed slave nodes to move as if in a rigid plane with the master node. To enforce this constraint, `Transformation` constraint is recommended.

perpDirn (int)	direction perpendicular to the rigid plane (i.e. direction 3 corresponds to the 1-2 plane)
rNodeTag (int)	integer tag identifying the master node
cNodeTags (list (int))	integar tags identifying the slave nodes

rigidLink command

rigidLink(*type*, *rNodeTag*, *cNodeTag*)
Create a multi-point constraint between nodes.

`type` (str)	string-based argument for rigid-link type: • `'bar'`: only the translational degree-of-freedom will be constrained to be exactly the same as those at the master node • `'beam'`: both the translational and rotational degrees of freedom are constrained.
`rNodeTag` (int)	integer tag identifying the master node
`cNodeTag` (int)	integar tag identifying the slave node

1.4.6 timeSeries commands

timeSeries (*tsType*, *tsTag*, **tsArgs*)

This command is used to construct a TimeSeries object which represents the relationship between the time in the domain, t, and the load factor applied to the loads, λ, in the load pattern with which the TimeSeries object is associated, i.e. $\lambda = F(t)$.

`tsType` (str)	time series type.
`tsTag` (int)	time series tag.
`tsArgs` (list)	a list of time series arguments

The following contain information about available `tsType`:

Constant TimeSeries

timeSeries (*'Constant'*, *tag*, *'-factor'*, *factor=1.0*)

This command is used to construct a TimeSeries object in which the load factor applied remains constant and is independent of the time in the domain, i.e. $\lambda = f(t) = C$.

`tag` (int)	unique tag among TimeSeries objects.
`factor` (float)	the load factor applied (optional)

Linear TimeSeries

timeSeries (*'Linear'*, *tag*, *'-factor'*, *factor=1.0*)

This command is used to construct a TimeSeries object in which the load factor applied is linearly proportional to the time in the domain, i.e.

$\lambda = f(t) = cFactor * t$.

`tag` (int)	unique tag among TimeSeries objects.
`factor` (float)	Linear factor. (optional)

Trigonometric TimeSeries

timeSeries (*'Trig'*, *tag*, *tStart*, *tEnd*, *period*, *'-factor'*, *factor=1.0*, *'-shift'*, *shift=0.0*, *'-zeroShift'*, *zeroShift=0.0*)

This command is used to construct a TimeSeries object in which the load factor is some trigonemtric function of the time in the domain

$$\lambda = f(t) = \begin{cases} cFactor * sin(\frac{2.0\pi(t-tStart)}{period} + \phi), & tStart <= t <= tEnd \\ 0.0, & otherwise \end{cases}$$

$$\phi = shift - \frac{period}{2.0\pi} * \arcsin(\frac{zeroShift}{cFactor})$$

`tag` (int)	unique tag among TimeSeries objects.
`tStart` (float)	Starting time of non-zero load factor.
`tEnd` (float)	Ending time of non-zero load factor.
`period` (float)	Characteristic period of sine wave.
`shift` (float)	Phase shift in radians. (optional)
`factor` (float)	Load factor. (optional)
`zeroShift` (float)	Zero shift. (optional)

Triangular TimeSeries

timeSeries (*'Triangle'*, *tag*, *tStart*, *tEnd*, *period*, *'-factor'*, *factor=1.0*, *'-shift'*, *shift=0.0*, *'-zeroShift'*, *zeroShift=0.0*)

This command is used to construct a TimeSeries object in which the load factor is some triangular function of the time in the domain.

$$\lambda = f(t) = \begin{cases} slope * k * period + zeroShift, & k < 0.25 \\ cFactor - slope * (k - 0.25) * period + zeroShift, & k < 0.75 \\ -cFactor + slope * (k - 0.75) * period + zeroShift, & k < 1.0 \\ 0.0, & otherwise \end{cases}$$

$$slope = \frac{cFactor}{period/4}$$

$$k = \frac{t + \phi - tStart}{period} - floor(\frac{t + \phi - tStart}{period})$$

$$\phi = shift - \frac{zeroShift}{slope}$$

`tag` (int)	unique tag among TimeSeries objects.
`tStart` (float)	Starting time of non-zero load factor.
`tEnd` (float)	Ending time of non-zero load factor.
`period` (float)	Characteristic period of sine wave.
`shift` (float)	Phase shift in radians. (optional)
`factor` (float)	Load factor. (optional)
`zeroShift` (float)	Zero shift. (optional)

Rectangular TimeSeries

timeSeries (*'Rectangular'*, *tag*, *tStart*, *tEnd*, *'-factor'*, *factor=1.0*)

This command is used to construct a TimeSeries object in which the load factor is constant for a specified period and 0 otherwise, i.e.

$$\lambda = f(t) = \begin{cases} cFactor, & tStart <= t <= tEnd \\ 0.0, & otherwise \end{cases}$$

`tag` (int)	unique tag among TimeSeries objects.
`tStart` (float)	Starting time of non-zero load factor.
`tEnd` (float)	Ending time of non-zero load factor.
`factor` (float)	Load factor. (optional)

Pulse TimeSeries

timeSeries (*'Pulse'*, *tag*, *tStart*, *tEnd*, *period*, *'-width'*, *width=0.5*, *'-shift'*, *shift=0.0*, *'-factor'*, *factor=1.0*, *'-zeroShift'*, *zeroShift=0.0*)

This command is used to construct a TimeSeries object in which the load factor is some pulse function of the time in the domain.

$$\lambda = f(t) = \begin{cases} cFactor + zeroShift, & k < width \\ zeroshift, & k < 1 \\ 0.0, & otherwise \end{cases}$$

$$k = \frac{t + shift - tStart}{period} - floor(\frac{t + shift - tStart}{period})$$

`tag` (int)	unique tag among TimeSeries objects.
`tStart` (float)	Starting time of non-zero load factor.
`tEnd` (float)	Ending time of non-zero load factor.
`period` (float)	Characteristic period of pulse.
`width` (float)	Pulse width as a fraction of the period. (optinal)
`shift` (float)	Phase shift in seconds. (optional)
`factor` (float)	Load factor. (optional)
`zeroShift` (float)	Zero shift. (optional)

Path TimeSeries

timeSeries (*'Path'*, *tag*, *'-dt'*, *dt=0.0*, *'-values'*, *values=[]*, *'-time'*, *time=[]*, *'-filepath'*, *filePath=''*, *'-fileTime'*, *fileTime=''*, *'-factor'*, *factor=1.0*, *'-startTime'*, *startTime=0.0*, *'-useLast'*, *'-prependZero'*)

The relationship between load factor and time is input by the user as a series of discrete points in the 2d space (load factor, time). The input points can come from a file or from a list in the script. When the time specified does not match any of the input points, linear interpolation is used between points. There are many ways to specify the load path, for example, the load factors set with `values` or `filePath`, and the time set with `dt`, `time`, or `fileTime`.

`tag` (int)	unique tag among TimeSeries objects.
`dt` (float)	Time interval between specified points. (optional)
`values` (list (float))	Load factor values in a (list). (optional)
`time` (list (float))	Time values in a (list). (optional)
`filePath` (str)	File containing the load factors values. (optional)
`fileTime` (str)	File containing the time values for corresponding load factors. (optional)
`factor` (float)	A factor to multiply load factors by. (optional)
`startTime` (float)	Provide a start time for provided load factors. (optional)
`'-useLast'` (str)	Use last value after the end of the series. (optional)
`'-prependZero'` (str)	Prepend a zero value to the series of load factors. (optional)

- Linear interpolation between points.

- If the specified time is beyond last point (AND WATCH FOR NUMERICAL ROUNDOFF), 0.0 is returned. Specify `'-useLast'` to use the last data point instead of 0.0.
- The transient integration methods in OpenSees assume zero initial conditions. So it is important that any timeSeries that is being used in a transient analysis‘ starts from zero (first data point in the timeSeries = 0.0). To guarantee that this is the case the optional parameter `'-prependZero'` can be specified to prepend a zero value to the provided TimeSeries.

1.4.7 pattern commands

pattern (*patternType*, *patternTag*, **patternArgs*)

The pattern command is used to construct a LoadPattern and add it to the Domain. Each LoadPattern in OpenSees has a TimeSeries associated with it. In addition it may contain ElementLoads, NodalLoads and SinglePointConstraints. Some of these SinglePoint constraints may be associated with GroundMotions.

`patternType` (str)	pattern type.
`patternTag` (int)	pattern tag.
`patternArgs` (list)	a list of pattern arguments

The following contain information about available `patternType`:

Plain Pattern

pattern (*'Plain'*, *patternTag*, *tsTag*, *'-fact'*, *factor*)

This commnand allows the user to construct a LoadPattern object. Each plain load pattern is associated with a TimeSeries object and can contain multiple NodalLoads, ElementalLoads and SP_Constraint objects. The command to generate LoadPattern object contains in { } the commands to generate all the loads and the single-point constraints in the pattern. To construct a load pattern and populate it, the following command is used:

`patternTag` (int)	unique tag among load patterns.
`tsTag` (int)	the tag of the time series to be used in the load pattern
`factor` (float)	constant factor. (optional)

Note: the commands below to generate all the loads and sp constraints will be included in last called pattern command.

load command

load (*nodeTag*, **loadValues*)

This command is used to construct a NodalLoad object and add it to the enclosing LoadPattern.

`nodeTag` (int)	tag of node to which load is applied.
`loadValues` (list (float))	ndf reference load values.

Note: The load values are reference loads values. It is the time series that provides the load factor. The load factor times the reference values is the load that is actually applied to the node.

eleLoad command

eleLoad(*'-ele'*, **eleTags*, *'-range'*, *eleTag1*, *eleTag2*, *'-type'*, *'-beamUniform'*, *Wy*, *Wz=0.0*, *Wx=0.0*, *'-beamPoint'*, *Py*, *Pz=0.0*, *xL*, *Px=0.0*, *'-beamThermal'*, **tempPts*)

The eleLoad command is used to construct an ElementalLoad object and add it to the enclosing LoadPattern.

`eleTags` (list (int))	tag of PREVIOUSLY DEFINED element
`eleTag1` (int)	element tag
`eleTag2` (int)	element tag
`Wx` (float)	mag of uniformily distributed ref load acting in direction along member length. (optional)
`Wy` (float)	mag of uniformily distributed ref load acting in local y direction of element
`Wz` (float)	mag of uniformily distributed ref load acting in local z direction of element. (optional and only for 3D)
`Px` (float)	mag of ref point load acting in direction along member length. (optional)
`Py` (float)	mag of ref point load acting in local y direction of element
`Pz` (float)	mag of ref point load acting in local z direction of element. (optional and only for 3D)
`xL` (float)	location of point load relative to node I, prescribed as fraction of element length
`tempPts` (list (float))	temperature points: `temPts = [T1, y1, T2, y2, ..., T9, y9]` Each point `(T1, y1)` define a temperature and location. This command may accept 2,5 or 9 temperature points.

Note:

1. The load values are reference loads values, it is the time sereries that provides the load factor. The load factor times the reference values is the load that is actually applied to the node.
2. At the moment, eleLoads do not work with 3D beam-column elements if Corotational geometric transformation is used.

sp command

sp(*nodeTag*, *dof*, **dofValues*)

This command is used to construct a single-point constraint object and add it to the enclosing LoadPattern.

`nodeTag` (int)	tag of node to which load is applied.
`dof` (int)	the degree-of-freedom at the node to which constraint is applied (1 through ndf)
`dofValues` (list (float))	ndf reference constraint values.

Note: The dofValue is a reference value, it is the time series that provides the load factor. The load factor times the reference value is the constraint that is actually applied to the node.

UniformExcitation Pattern

pattern(*'UniformExcitation'*, *patternTag*, *dir*, *'-disp'*, *dispSeriesTag*, *'-vel'*, *velSeriesTag*, *'-accel'*, *accelSeriesTag*, *'-vel0'*, *vel0*, *'-fact'*, *factor*)

The UniformExcitation pattern allows the user to apply a uniform excitation to a model acting in a certain direction. The command is as follows:

`patternTag` (int)	unique tag among load patterns
`dir` (int)	direction in which ground motion acts 1. corresponds to translation along the global X axis 2. corresponds to translation along the global Y axis 3. corresponds to translation along the global Z axis 4. corresponds to rotation about the global X axis 5. corresponds to rotation about the global Y axis 6. corresponds to rotation about the global Z axis
`dispSeriesTag` (int)	tag of the TimeSeries series defining the displacement history. (optional)
`velSeriesTag` (int)	tag of the TimeSeries series defining the velocity history. (optional)
`accelSeriesTag` (int)	tag of the TimeSeries series defining the acceleration history. (optional)
`vel0` (float)	the initial velocity (optional, default=0.0)
`factor` (float)	constant factor (optional, default=1.0)

Note:

1. The responses obtained from the nodes for this type of excitation are RELATIVE values, and not the absolute values obtained from a multi-support case.
2. must set one of the disp, vel or accel time series

Multi-Support Excitation Pattern

pattern(*'MultipleSupport'*, *patternTag*)

The Multi-Support pattern allows similar or different prescribed ground motions to be input at various supports in the structure. In OpenSees, the prescribed motion is applied using single-point constraints, the single-point constraints taking their constraint value from user created ground motions.

Note:

1. The results for the responses at the nodes are the ABSOLUTE values, and not relative values as in the case of a UniformExciatation.
2. The non-homogeneous single point constraints require an appropriate choice of constraint handler.

Plain Ground Motion

groundMotion(*gmTag*, *'Plain'*, *'-disp'*, *dispSeriesTag*, *'-vel'*, *velSeriesTag*, *'-accel'*, *accelSeriesTag*, *'-int'*, *tsInt='Trapezoidal'*, *'-fact'*, *factor=1.0*)

This command is used to construct a plain GroundMotion object. Each GroundMotion object is associated with a number of TimeSeries objects, which define the acceleration, velocity and displacement records for that ground motion. T

`gmTag` (int)	unique tag among ground motions in load pattern
`dispSeriesTag` (int)	tag of the TimeSeries series defining the displacement history. (optional)
`velSeriesTag` (int)	tag of the TimeSeries series defining the velocity history. (optional)
`accelSeriesTag` (int)	tag of the TimeSeries series defining the acceleration history. (optional)
`tsInt` (str)	`'Trapezoidal'` or `'Simpson'` numerical integration method
`factor` (float)	constant factor. (optional)

Note:

1. The displacements are the ones used in the ImposedMotions to set nodal response.
2. If only the acceleration TimeSeries is provided, numerical integration will be used to determine the velocities and displacements.
3. For earthquake excitations it is important that the user provide the displacement time history, as the one generated using the trapezoidal method will not provide good results.
4. Any combination of the acceleration, velocity and displacement time-series can be specified.

Interpolated Ground Motion

groundMotion(*gmTag*, *'Interpolated'*, **gmTags*, *'-fact'*, *facts*)

This command is used to construct an interpolated GroundMotion object, where the motion is determined by combining several previously defined ground motions in the load pattern.

`gmTag` (int)	unique tag among ground motions in load pattern
`gmTags` (list (int))	the tags of existing ground motions in pattern to be used for interpolation
`facts` (list (float))	the interpolation factors. (optional)

Imposed Motion

imposedMotion(*nodeTag*, *dof*, *gmTag*)

This command is used to construct an ImposedMotionSP constraint which is used to enforce the response of a dof at a node in the model. The response enforced at the node at any give time is obtained from the GroundMotion object associated with the constraint.

`nodeTag` (int)	tag of node on which constraint is to be placed
`dof` (int)	dof of enforced response. Valid range is from 1 through ndf at node.
`gmTag` (int)	pre-defined GroundMotion object tag

1.4.8 mass command

mass (*nodeTag*, **massValues*)

This command is used to set the mass at a node

nodeTag (int)	integer tag identifying node whose mass is set
massValues (list (float))	ndf nodal mass values corresponding to each DOF

1.4.9 region command

region (*regTag*, *'-ele'*, **eles*, *'-eleOnly'*, **eles*, *'-eleRange'*, *startEle*, *endEle*, *'-eleOnlyRange'*, *startEle*, *endEle*, *'-node'*, **nodes*, *'-nodeOnly'*, **nodes*, *'-nodeRange'*, *startNode*, *endNode*, *'-nodeOnlyRange'*, *startNode*, *endNode*, *'-rayleigh'*, *alphaM*, *betaK*, *betaKinit*, *betaKcomm*)

The region command is used to label a group of nodes and elements. This command is also used to assign rayleigh damping parameters to the nodes and elements in this region. The region is specified by either elements or nodes, not both. If elements are defined, the region includes these elements and the all connected nodes, unless the -eleOnly option is used in which case only elements are included. If nodes are specified, the region includes these nodes and all elements of which all nodes are prescribed to be in the region, unless the -nodeOnly option is used in which case only the nodes are included.

regTag (int)	unique integer tag
eles (list (int))	tags of selected elements in domain to be included in region (optional)
nodes (list (int))	tags of selected nodes in domain to be included in region (optional)
startEle (int)	tag for start element (optional)
endEle (int)	tag for end element (optional)
startNode (int)	tag for start node (optional)
endNode (int)	tag for end node (optional)
alphaM (float)	factor applied to elements or nodes mass matrix (optional)
betaK (float)	factor applied to elements current stiffness matrix (optional)
betaKinit (float)	factor applied to elements initial stiffness matrix (optional)
betaKcomm (float)	factor applied to elements committed stiffness matrix (optional)

Note: The user cannot prescribe the region by BOTH elements and nodes.

1.4.10 rayleigh command

rayleigh (*alphaM*, *betaK*, *betaKinit*, *betaKcomm*)

This command is used to assign damping to all previously-defined elements and nodes. When using rayleigh damping in OpenSees, the damping matrix for an element or node, D is specified as a combination of stiffness and mass-proportional damping matrices:

$$D = \alpha_M * M + \beta_K * K_{curr} + \beta_{Kinit} * K_{init} + \beta_{Kcomm} * K_{commit}$$

alphaM (float)	factor applied to elements or nodes mass matrix
betaK (float)	factor applied to elements current stiffness matrix.
betaKinit (float)	factor applied to elements initial stiffness matrix.
betaKcomm (float)	factor applied to elements committed stiffness matrix.

1.4.11 block commands

Create a block of mesh

block2D command

block2D (*numX*, *numY*, *startNode*, *startEle*, *eleType*, **eleArgs*, **crds*)
Create mesh of quadrilateral elements

`numX` (int)	number of elements in local x directions of the block.
`numY` (int)	number of elements in local y directions of the block.
`startNode` (int)	node from which the mesh generation will start.
`startEle` (int)	element from which the mesh generation will start.
`eleType` (str)	element type (`'quad'`, `'shell'`, `'bbarQuad'`, `'enhancedQuad'`, or `'SSPquad'`)
`eleArgs` (list)	a list of element parameters.
`crds` (list)	coordinates of the block elements with the format: [1, x1, y1, <z1>, 2, x2, y2, <z2>, 3, x3, y3, <z3>, 4, x4, y4, <z4>, <5>, <x5>, <y5>, <z5>, <6>, <x6>, <y6>, <z6>, <7>, <x7>, <y7>, <z7>, <8>, <x8>, <y8>, <z8>, <9>, <x9>, <y9>, <z9>] <> means optional

block3D command

block3D (*numX*, *numY*, *numZ*, *startNode*, *startEle*, *eleType*, **eleArgs*, **crds*)
Create mesh of quadrilateral elements

numX (int)	number of elements in local x directions of the block.
numY (int)	number of elements in local y directions of the block.
numZ (int)	number of elements in local z directions of the block.
startNode (int)	node from which the mesh generation will start.
startEle (int)	element from which the mesh generation will start.
eleType (str)	element type ('stdBrick', 'bbarBrick', 'Brick20N')
eleArgs (list)	a list of element parameters.
crds (list)	coordinates of the block elements with the format: [1, x1, y1, z1, 2, x2, y2, z2, 3, x3, y3, z3, 4, x4, y4, z4, 5, x5, y5, z5, 6, x6, y6, z6, 7, x7, y7, z7, 8, x8, y8, z8, 9, x9, y9, z9, <10>, <x10>, <y10>, <z10>, <11>, <x11>, <y11>, <z11>, <12>, <x12>, <y12>, <z12>, <13>, <x13>, <y13>, <z13>, <14>, <x14>, <y14>, <z14>, <15>, <x15>, <y15>, <z15>, <16>, <x16>, <y16>, <z16>, <17>, <x17>, <y17>, <z17>, <18>, <x18>, <y18>, <z18>, <19>, <x19>, <y19>, <z19>, <20>, <x20>, <y20>, <z20>, <21>, <x21>, <y21>, <z21>, <22>, <x22>, <y22>, <z22>, <23>, <x23>, <y23>, <z23>, <24>, <x24>, <y24>, <z24>, <25>, <x25>, <y25>, <z25>, <26>, <x26>, <y26>, <z26>, <27>, <x27>, <y27>, <z27>] <> means optional

1.4.12 beamIntegration commands

beamIntegration (*type*, *tag*, **args*)

A wide range of numerical integration options are available in OpenSees to represent distributed plasticity or non-prismatic section details in Beam-Column Elements, i.e., across the entire element domain [0, L].

Following are beamIntegration types available in the OpenSees:

Integration Methods for Distributed Plasticity. Distributed plasticity methods permit yielding at any integration point along the element length.

Lobatto

beamIntegration (*'Lobatto'*, *tag*, *secTag*, *N*)

Create a Gauss-Lobatto beamIntegration object. Gauss-Lobatto integration is the most common approach for evaluating the response of *forceBeamColumn* (Neuenhofer and Filippou 1997) because it places an integration point at each end of the element, where bending moments are largest in the absence of interior element loads.

`tag` (int)	tag of the beam integration.
`secTag` (int)	A previous-defined section object.
`N` (int)	Number of integration points along the element.

Legendre

beamIntegration (*'Legendre'*, *tag*, *secTag*, *N*)

Create a Gauss-Legendre beamIntegration object. Gauss-Legendre integration is more accurate than Gauss-Lobatto; however, it is not common in force-based elements because there are no integration points at the element ends.

Places `N` Gauss-Legendre integration points along the element. The location and weight of each integration point are tabulated in references on numerical analysis. The force deformation response at each integration point is defined by the section. The order of accuracy for Gauss-Legendre integration is 2N-1.

Arguments and examples see *Lobatto*.

NewtonCotes

beamIntegration (*'NewtonCotes'*, *tag*, *secTag*, *N*)

Create a Newton-Cotes beamIntegration object. Newton-Cotes places integration points uniformly along the element, including a point at each end of the element.

Places `N` Newton-Cotes integration points along the element. The weights for the uniformly spaced integration points are tabulated in references on numerical analysis. The force deformation response at each integration point is defined by the section. The order of accuracy for Gauss-Radau integration is N-1.

Arguments and examples see *Lobatto*.

Radau

beamIntegration (*'Radau'*, *tag*, *secTag*, *N*)

Create a Gauss-Radau beamIntegration object. Gauss-Radau integration is not common in force-based elements because it places an integration point at only one end of the element; however, it forms the basis for optimal plastic hinge integration methods.

Places `N` Gauss-Radau integration points along the element with a point constrained to be at ndI. The location and weight of each integration point are tabulated in references on numerical analysis. The force-deformation response at each integration point is defined by the section. The order of accuracy for Gauss-Radau integration is 2N-2.

Arguments and examples see *Lobatto*.

Trapezoidal

beamIntegration(*'Trapezoidal'*, *tag*, *secTag*, *N*)

Create a Trapezoidal beamIntegration object.

Arguments and examples see *Lobatto*.

CompositeSimpson

beamIntegration(*'CompositeSimpson'*, *tag*, *secTag*, *N*)

Create a CompositeSimpson beamIntegration object.

Arguments and examples see *Lobatto*.

UserDefined

beamIntegration(*'UserDefined'*, *tag*, *N*, **secTags*, **locs*, **wts*)

Create a UserDefined beamIntegration object. This option allows user-specified locations and weights of the integration points.

`tag` (int)	tag of the beam integration
`N` (int)	number of integration points along the element.
`secTags` (list (int))	A list previous-defined section objects.
`locs` (list (float))	Locations of integration points along the element.
`wts` (list (float))	weights of integration points.

```
locs = [0.1, 0.3, 0.5, 0.7, 0.9]
wts = [0.2, 0.15, 0.3, 0.15, 0.2]
secs = [1, 2, 2, 2, 1]
beamIntegration('UserDefined',1,len(secs),*secs,*locs,*wts)
```

Places `N` integration points along the element, which are defined in `locs` on the natural domain [0, 1]. The weight of each integration point is defined in the `wts` also on the [0, 1] domain. The force-deformation response at each integration point is defined by the `secs`. The `locs`, `wts`, and `secs` should be of length `N`. In general, there is no accuracy for this approach to numerical integration.

FixedLocation

beamIntegration(*'FixedLocation'*, *tag*, *N*, **secTags*, **locs*)

Create a FixedLocation beamIntegration object. This option allows user-specified locations of the integration points. The associated integration weights are computed by the method of undetermined coefficients (Vandermonde system)

$$\sum_{i=1}^{N} x_i^{j-1} w_i = \int_0^1 x^{j-1} dx = \frac{1}{j}, \qquad (j = 1, ..., N)$$

Note that *NewtonCotes* integration is recovered when the integration point locations are equally spaced.

`tag` (int)	tag of the beam integration
`N` (int)	number of integration points along the element.
`secTags` (list (int))	A list previous-defined section objects.
`locs` (list (float))	Locations of integration points along the element.

Places `N` integration points along the element, whose locations are defined in `locs`. on the natural domain [0, 1]. The force-deformation response at each integration point is defined by the `secs`. Both the `locs` and `secs` should be of length `N`. The order of accuracy for Fixed Location integration is N-1.

LowOrder

beamIntegration (*'LowOrder'*, *tag*, *N*, **secTags*, **locs*, **wts*)

Create a LowOrder beamIntegration object. This option is a generalization of the *FixedLocation* and *UserDefined* integration approaches and is useful for moving load analysis (Kidarsa, Scott and Higgins 2008). The locations of the integration points are user defined, while a selected number of weights are specified and the remaining weights are computed by the method of undetermined coefficients.

$$\sum_{i=1}^{N_f} x_{fi}^{j-1} w_{fi} = \frac{1}{j} - \sum_{i=1}^{N_c} x_{ci}^{j-1} w_{ci}$$

Note that *FixedLocation* integration is recovered when `Nc` is zero.

`tag` (int)	tag of the beam integration
`N` (int)	number of integration points along the element.
`secTags` (list (int))	A list previous-defined section objects.
`locs` (list (float))	Locations of integration points along the element.
`wts` (list (float))	weights of integration points.

```
locs = [0.0, 0.2, 0.5, 0.8, 1.0]
wts = [0.2, 0.2]
secs = [1, 2, 2, 2, 1]
beamIntegration('LowOrder',1,len(secs),*secs,*locs,*wts)
```

Places `N` integration points along the element, which are defined in `locs`. on the natural domain [0, 1]. The force-deformation response at each integration point is defined by the `secs`. Both the `locs` and `secs` should be of length `N`. The `wts` at user-selected integration points are specified on [0, 1], which can be of length `Nc` equals `0` up to `N`. These specified weights are assigned to the first `Nc` entries in the `locs` and `secs`, respectively. The order of accuracy for Low Order integration is N-Nc-1.

Note: `Nc` is determined from the length of the `wts` list. Accordingly, *FixedLocation* integration is recovered when `wts` is an empty list and *UserDefined* integration is recovered when the `wts` and `locs` lists are of equal length.

MidDistance

beamIntegration (*'MidDistance'*, *tag*, *N*, **secTags*, **locs*)

Create a MidDistance beamIntegration object. This option allows user-specified locations of the integration points. The associated integration weights are determined from the midpoints between adjacent integration point locations. $w_i = (x_{i+1} - x_{i-1})/2$ for $i = 2 \ldots N-1$, $w_1 = (x_1 + x_2)/2$, and $w_N = 1 - (x_{N-1} + x_N)/2$.

`tag` (int)	tag of the beam integration
`N` (int)	number of integration points along the element.
`secTags` (list (int))	A list previous-defined section objects.
`locs` (list (float))	Locations of integration points along the element.

```
locs = [0.0, 0.2, 0.5, 0.8, 1.0]
secs = [1,2,2,2,1]
beamIntegration('MidDistance',1,len(secs),*secs,*locs)
```

Places `N` integration points along the element, whose locations are defined in `locs` on the natural domain [0, 1]. The force-deformation response at each integration point is defined by the `secs`. Both the `locs` and `secs` should be of length N. This integration rule can only integrate constant functions exactly since the sum of the integration weights is one.

For the `locs` shown above, the associated integration weights will be `[0.15, 0.2, 0.3, 0.2, 0.15]`.

Plastic Hinge Integration Methods. Plastic hinge integration methods confine material yielding to regions of the element of specified length while the remainder of the element is linear elastic. A summary of plastic hinge integration methods is found in (Scott and Fenves 2006).

UserHinge

beamIntegration (*'UserHinge'*, *tag*, *secE*, *npL*, **secsL*, **locsL*, **wtsL*, *npR*, **secsR*, **locsR*, **wtsR*)
Create a UserHinge beamIntegration object.

`tag` (int)	tag of the beam integration
`secE` (int)	A previous-defined section objects for non-hinge area.
`npL` (int)	number of integration points along the left hinge.
`secsL` (list (int))	A list of previous-defined section objects for left hinge area.
`locsL` (list (float))	A list of locations of integration points for left hinge area.
`wtsL` (list (float))	A list of weights of integration points for left hinge area.
`npR` (int)	number of integration points along the right hinge.
`secsR` (list (int))	A list of previous-defined section objects for right hinge area.
`locsR` (list (float))	A list of locations of integration points for right hinge area.
`wtsR` (list (float))	A list of weights of integration points for right hinge area.

```
tag = 1
secE = 5

npL = 2
secsL = [1,2]
locsL = [0.1,0.2]
wtsL = [0.5,0.5]

npR = 2
secsR = [3,4]
locsR = [0.8,0.9]
wtsR = [0.5,0.5]

beamIntegration('UserHinge',tag,secE,npL,*secsL,*locsL,*wtsL,npR,*secsR,*locsR,
↪*wtsR)
```

HingeMidpoint

beamIntegration (*'HingeMidpoint'*, *tag*, *secI*, *lpI*, *secJ*, *lpJ*, *secE*)
Create a HingeMidpoint beamIntegration object. Midpoint integration over each hinge region is the most accurate one-point integration rule; however, it does not place integration points at the element ends and there is a small integration error for linear curvature distributions along the element.

`tag` (int)	tag of the beam integration.
`secI` (int)	A previous-defined section object for hinge at I.
`lpI` (float)	The plastic hinge length at I.
`secJ` (int)	A previous-defined section object for hinge at J.
`lpJ` (float)	The plastic hinge length at J.
`secE` (int)	A previous-defined section object for the element interior.

The plastic hinge length at end I (J) is equal to `lpI` (`lpJ`) and the associated force deformation response is defined by the `secI` (`secJ`). The force deformation response of the element interior is defined by the `secE`. Typically, the interior section is linear-elastic, but this is not necessary.

```
lpI = 0.1
lpJ = 0.2
beamIntegration('HingeMidpoint',secI,lpI,secJ,lpJ,secE)
```

HingeRadau

beamIntegration(*'HingeRadau'*, *tag*, *secI*, *lpI*, *secJ*, *lpJ*, *secE*)

Create a HingeRadau beamIntegration object. Two-point Gauss-Radau integration over each hinge region places an integration point at the element ends and at 2/3 the hinge length inside the element. This approach represents linear curvature distributions exactly; however, the characteristic length for softening plastic hinges is not equal to the assumed palstic hinge length.

Arguments and examples see *HingeMidpoint*.

HingeRadauTwo

beamIntegration(*'HingeRadauTwo'*, *tag*, *secI*, *lpI*, *secJ*, *lpJ*, *secE*)

Create a HingeRadauTwo beamIntegration object. Modified two-point Gauss-Radau integration over each hinge region places an integration point at the element ends and at 8/3 the hinge length inside the element. This approach represents linear curvature distributions exactly and the characteristic length for softening plastic hinges is equal to the assumed plastic hinge length.

Arguments and examples see *HingeMidpoint*.

HingeEndpoint

beamhingeEndpoint(*tag*, *secI*, *lpI*, *secJ*, *lpJ*, *secE*)

Create a HingeEndpoint beamIntegration object. Endpoint integration over each hinge region moves the integration points to the element ends; however, there is a large integration error for linear curvature distributions along the element.

Arguments and examples see *HingeMidpoint*.

1.4.13 uniaxialMaterial commands

uniaxialMaterial(*matType*, *matTag*, **matArgs*)

This command is used to construct a UniaxialMaterial object which represents uniaxial stress-strain (or force-deformation) relationships.

`matType` (str)	material type
`matTag` (int)	material tag.
`matArgs` (list)	a list of material arguments, must be preceded with `*`.

For example,

```
matType = 'Steel01'
matTag = 1
matArgs = [Fy, E0, b]
uniaxialMaterial(matType, matTag, *matArgs)
```

The following contain information about available `matType`:

Steel01

uniaxialMaterial(*'Steel01'*, *matTag*, *Fy*, *E0*, *b*, *a1*, *a2*, *a3*, *a4*)

This command is used to construct a uniaxial bilinear steel material object with kinematic hardening and optional isotropic hardening described by a non-linear evolution equation (REF: Fedeas).

`matTag` (int)	integer tag identifying material
`Fy` (float)	yield strength
`E0` (float)	initial elastic tangent
`b` (float)	strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
`a1` (float)	isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a_2 * (F_y/E_0)$ (optional)
`a2` (float)	isotropic hardening parameter (see explanation under `a1`). (optional).
`a3` (float)	isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a_4 * (F_y/E_0)$. (optional)
`a4` (float)	isotropic hardening parameter (see explanation under `a3`). (optional)

Note: If strain-hardening ratio is zero and you do not expect softening of your system use BandSPD solver.

Steel02

uniaxialMaterial(*'Steel02'*, *matTag*, *Fy*, *E0*, *b*, **params*, *a1=a2*Fy/E0*, *a2=1.0*, *a3=a4*Fy/E0*, *a4=1.0*, *sigInit=0.0*)

This command is used to construct a uniaxial Giuffre-Menegotto-Pinto steel material object with isotropic strain hardening.

matTag (int)	integer tag identifying material
Fy (float)	yield strength
E0 (float)	initial elastic tangent
b (float)	strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
params (list (float))	parameters to control the transition from elastic to plastic branches. params=[R0,cR1,cR2]. Recommended values: R0=between 10 and 20, cR1=0.925, cR2=0.15
a1 (float)	isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a_2 * (F_y/E_0)$ (optional)
a2 (float)	isotropic hardening parameter (see explanation under a1). (optional).
a3 (float)	isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a_4 * (F_y/E_0)$. (optional)
a4 (float)	isotropic hardening parameter (see explanation under a3). (optional)

sigInit (float)	Initial Stress Value (optional, default: 0.0) the strain is calculated from epsP=sigInit/E
	`if (sigInit!= 0.0) {`
	`  double epsInit = sigInit/E;`
	`  eps = trialStrain+epsInit;`
	`} else {`
	`  eps = trialStrain;`
	`}`

See also:

Steel02

Steel4

uniaxialMaterial (*'Steel4'*, *matTag*, *Fy*, *E0*, *'-asym'*, *'-kin'*, *b_k*, *R_0*, *r_1*, *r_2*, *b_kc*, *R_0c*, *r_1c*, *r_2c*, *'-iso'*, *b_i*, *rho_i*, *b_l*, *R_i*, *l_yp*, *b_ic*, *rho_ic*, *b_lc*, *R_ic*, *'-ult'*, *f_u*, *R_u*, *f_uc*, *R_uc*, *'-init'*, *sig_init*, *'-mem'*, *cycNum*)

This command is used to construct a general uniaxial material with combined kinematic and isotropic hardening and optional non-symmetric behavior.

matTag (int)	integer tag identifying material
Fy (float)	yield strength
E0 (float)	initial elastic tangent
'-kin' (str)	apply kinematic hardening
b_k (float)	hardening ratio (E_k/E_0)
R_0, r_1, r_2 (float)	control the exponential transition from linear elastic to hardening asymptote recommended values: `R_0 = 20, r_1 = 0.90, r_2 = 0.15`
'-iso' (str)	apply isotropic hardening
b_i (float)	initial hardening ratio (E_i/E_0)
b_l (float)	saturated hardening ratio (E_is/E_0)
rho_i (float)	specifies the position of the intersection point between initial and saturated hardening asymptotes
R_i (float)	control the exponential transition from initial to saturated asymptote
l_yp (float)	length of the yield plateau in eps_y0 = f_y / E_0 units
'-ult' (str)	apply an ultimate strength limit
f_u (float)	ultimate strength
R_u (float)	control the exponential transition from kinematic hardening to perfectly plastic asymptote
'-asym' (str)	assume non-symmetric behavior
'-init' (str)	apply initial stress
sig_init (float)	initial stress value
'-mem' (str)	configure the load history memory
cycNum (float)	expected number of half-cycles during the loading process Efficiency of the material can be slightly increased by correctly setting this value. The default value is `cycNum = 50` Load history memory can be turned off by setting `cycNum = 0`.

See also:

Steel4

Hysteretic

uniaxialMaterial('Hysteretic', *matTag*, **p1*, **p2*, **p3=p2*, **n1*, **n2*, **n3=n2*, *pinchX*, *pinchY*, *damage1*, *damage2*, *beta*)

This command is used to construct a uniaxial bilinear hysteretic material object with pinching of force and deformation, damage due to ductility and energy, and degraded unloading stiffness based on ductility.

`matTag` (int)	integer tag identifying material
`p1` (list (float))	`p1=[s1p, e1p]`, stress and strain (or force & deformation) at first point of the envelope in the positive direction
`p2` (list (float))	`p2=[s2p, e2p]`, stress and strain (or force & deformation) at second point of the envelope in the positive direction
`p3` (list (float))	`p3=[s3p, e3p]`, stress and strain (or force & deformation) at third point of the envelope in the positive direction
`n1` (list (float))	`n1=[s1n, e1n]`, stress and strain (or force & deformation) at first point of the envelope in the negative direction
`n2` (list (float))	`n2=[s2n, e2n]`, stress and strain (or force & deformation) at second point of the envelope in the negative direction
`n3` (list (float))	`n3=[s3n, e3n]`, stress and strain (or force & deformation) at third point of the envelope in the negative direction
`pinchx` (float)	pinching factor for strain (or deformation) during reloading
`pinchy` (float)	pinching factor for stress (or force) during reloading
`damage1` (float)	damage due to ductility: D1(mu-1)
`damage2` (float)	damage due to energy: D2(Eii/Eult)
`beta` (float)	power used to determine the degraded unloading stiffness based on ductility, mu-beta (optional, default=0.0)

See also:

Steel4

ReinforcingSteel

uniaxialMaterial('ReinforcingSteel', *matTag*, *fy*, *fu*, *Es*, *Esh*, *esh*, *eult*, *'-GABuck'*, *lsr*, *beta*, *r*, *gama*, *'-DMBuck'*, *lsr*, *alpha=1.0*, *'-CMFatigue'*, *Cf*, *alpha*, *Cd*, *'-IsoHard'*, *a1=4.3*, *limit=1.0*, *'-MPCurveParams'*, *R1=0.333*, *R2=18.0*, *R3=4.0*)

This command is used to construct a ReinforcingSteel uniaxial material object. This object is intended to be used in a reinforced concrete fiber section as the steel reinforcing material.

`matTag` (int)	integer tag identifying material
`fy` (float)	Yield stress in tension
`fu` (float)	Ultimate stress in tension
`Es` (float)	Initial elastic tangent
`Esh` (float)	Tangent at initial strain hardening
`esh` (float)	Strain corresponding to initial strain hardening
`eult` (float)	Strain at peak stress
`'-GABuck'` (str)	Buckling Model Based on Gomes and Appleton (1997)
`lsr` (float)	Slenderness Ratio
`beta` (float)	Amplification factor for the buckled stress strain curve.
`r` (float)	Buckling reduction factor r can be a real number between [0.0 and 1.0] r=1.0 full reduction (no buckling) r=0.0 no reduction 0.0<r<1.0 linear interpolation between buckled and unbuckled curves
`gamma` (float)	Buckling constant
`'-DMBuck'` (str)	Buckling model based on Dhakal and Maekawa (2002)
`lsr` (float)	Slenderness Ratio
`alpha` (float)	Adjustment Constant usually between 0.75 and 1.0 Default: alpha=1.0, this parameter is optional.
`'-CMFatigue'` (str)	Coffin-Manson Fatigue and Strength Reduction
`Cf` (float)	Coffin-Manson constant C
`alpha` (float)	Coffin-Manson constant a
`Cd` (float)	Cyclic strength reduction constant
`'-IsoHard;` (str)	Isotropic Hardening / Diminishing Yield Plateau
`a1` (float)	Hardening constant (default = 4.3)
`limit` (float)	Limit for the reduction of the yield plateau. % of original plateau length to remain (0.01 < limit < 1.0) Limit =1.0, then no reduction takes place (default =0.01)
`'-MPCurveParams'` (str)	Menegotto and Pinto Curve Parameters
`R1` (float)	(default = 0.333)
`R2` (float)	(default = 18)
`R3` (float)	(default = 4)

See also:

Notes

Dodd_Restrepo

uniaxialMaterial (*'Dodd_Restrepo'*, *matTag*, *Fy*, *Fsu*, *ESH*, *ESU*, *Youngs*, *ESHI*, *FSHI*, *OmegaFac=1.0*)

This command is used to construct a Dodd-Restrepo steel material

matTag (int)	integer tag identifying material
Fy (float)	Yield strength
Fsu (float)	Ultimate tensile strength (UTS)
ESH (float)	Tensile strain at initiation of strain hardening
ESU (float)	Tensile strain at the UTS
Youngs (float)	Modulus of elasticity
ESHI (float)	Tensile strain for a point on strain hardening curve, recommended range of values for ESHI: [(ESU + 5*ESH)/6, (ESU + 3*ESH)/4]
FSHI (float)	Tensile stress at point on strain hardening curve corresponding to ESHI
OmegaFac (float)	Roundedness factor for Bauschinger curve in cycle reversals from the strain hardening curve. Range: [0.75, 1.15]. Largest value tends to near a bilinear Bauschinger curve. Default = 1.0.

See also:

Notes

RambergOsgoodSteel

uniaxialMaterial(*'RambergOsgoodSteel'*, *matTag*, *fy*, *E0*, *a*, *n*)

This command is used to construct a Ramberg–Osgood steel material object.

matTag (int)	integer tag identifying material
fy (float)	Yield strength
E0 (float)	initial elastic tangent
a (float)	"yield offset" and the Commonly used value for a is 0.002
n (float)	Parameters to control the transition from elastic to plastic branches. And controls the hardening of the material by increasing the "n" hardening ratio will be decreased. Commonly used values for n are ~5 or greater.

See also:

Notes

SteelMPF

uniaxialMaterial(*'SteelMPF'*, *matTag*, *fyp*, *fyn*, *E0*, *bp*, *bn*, *R0*, *cR1*, *cR2*, *a1=0.0*, *a2=1.0*, *a3=0.0*, *a4=1.0*)

This command is used to construct a uniaxialMaterial SteelMPF (Kolozvari et al., 2015), which represents the well-known uniaxial constitutive nonlinear hysteretic material model for steel proposed by Menegotto and Pinto (1973), and extended by Filippou et al. (1983) to include isotropic strain hardening effects.

`matTag` (int)	integer tag identifying material
`fyp` (float)	Yield strength in tension (positive loading direction)
`fyn` (float)	Yield strength in compression (negative loading direction)
`E0` (float)	Initial tangent modulus
`bp` (float)	Strain hardening ratio in tension (positive loading direction)
`bn` (float)	Strain hardening ratio in compression (negative loading direction)
`R0` (float)	Initial value of the curvature parameter R (R0 = 20 recommended)
`cR1` (float)	Curvature degradation parameter (a1 = 0.925 recommended)
`cR2` (float)	Curvature degradation parameter (a2 = 0.15 or 0.0015 recommended)
`a1` (float)	Isotropic hardening in compression parameter (optional, default = 0.0). Shifts compression yield envelope by a proportion of compressive yield strength after a maximum plastic tensile strain of a2(fyp/E0)
`a2` (float)	Isotropic hardening in compression parameter (optional, default = 1.0).
`a3` (float)	Isotropic hardening in tension parameter (optional, default = 0.0). Shifts tension yield envelope by a proportion of tensile yield strength after a maximum plastic compressive strain of a3(fyn/E0).
`a4` (float)	Isotropic hardening in tension parameter (optional, default = 1.0). See explanation of a3.

See also:

Notes

Concrete01

`uniaxialMaterial`(*'Concrete01'*, *matTag*, *fpc*, *epsc0*, *fpcu*, *epsU*)

This command is used to construct a uniaxial Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength. (REF: Fedeas).

`matTag` (int)	integer tag identifying material
`fpc` (float)	concrete compressive strength at 28 days (compression is negative)
`epsc0` (float)	concrete strain at maximum strength
`fpcu` (float)	concrete crushing strength
`epsU` (float)	concrete strain at crushing strength

Note:

1. Compressive concrete parameters should be input as negative values (if input as positive, they will be converted to negative internally).
2. The initial slope for this model is (2*fpc/epsc0)

See also:

Notes

Concrete02

uniaxialMaterial(*'Concrete02'*, *matTag*, *fpc*, *epsc0*, *fpcu*, *epsU*, *lambda*, *ft*, *Ets*)

This command is used to construct a uniaxial Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength. (REF: Fedeas).

`matTag` (int)	integer tag identifying material
`fpc` (float)	concrete compressive strength at 28 days (compression is negative)
`epsc0` (float)	concrete strain at maximum strength
`fpcu` (float)	concrete crushing strength
`epsU` (float)	concrete strain at crushing strength
`lambda` (float)	ratio between unloading slope at $epscu and initial slope
`ft` (float)	tensile strength
`Ets` (float)	tension softening stiffness (absolute value) (slope of the linear tension softening branch)

Note:

1. Compressive concrete parameters should be input as negative values (if input as positive, they will be converted to negative internally).
2. The initial slope for this model is (2*fpc/epsc0)

See also:

Notes

Concrete04

uniaxialMaterial(*'Concrete04'*, *matTag*, *fc*, *ec*, *ecu*, *Ec*, *fct*, *et*, *beta*)

This command is used to construct a uniaxial Popovics concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and tensile strength with exponential decay.

`matTag` (int)	integer tag identifying material
`fc` (float)	floating point values defining concrete compressive strength at 28 days (compression is negative)
`ec` (float)	floating point values defining concrete strain at maximum strength
`ecu` (float)	floating point values defining concrete strain at crushing strength
`Ec` (float)	floating point values defining initial stiffness
`fct` (float)	floating point value defining the maximum tensile strength of concrete (optional)
`et` (float)	floating point value defining ultimate tensile strain of concrete (optional)
`beta` (float)	loating point value defining the exponential curve parameter to define the residual stress (as a factor of ft) at etu

Note:

1. Compressive concrete parameters should be input as negative values.

2. The envelope of the compressive stress-strain response is defined using the model proposed by Popovics (1973). If the user defines $Ec = 57000 * sqrt(|fcc|)$ (in psi)' then the envelope curve is identical to proposed by Mander et al. (1988).
3. Model Characteristic: For loading in compression, the envelope to the stress-strain curve follows the model proposed by Popovics (1973) until the concrete crushing strength is achieved and also for strains beyond that corresponding to the crushing strength. For unloading and reloading in compression, the Karsan-Jirsa model (1969) is used to determine the slope of the curve. For tensile loading, an exponential curve is used to define the envelope to the stress-strain curve. For unloading and reloading in tensile, the secant stiffness is used to define the path.

See also:

Notes

Concrete06

uniaxialMaterial (*'Concrete06'*, *matTag*, *fc*, *e0*, *n*, *k*, *alpha1*, *fcr*, *ecr*, *b*, *alpha2*)

This command is used to construct a uniaxial concrete material object with tensile strength, nonlinear tension stiffening and compressive behavior based on Thorenfeldt curve.

`matTag` (int)	integer tag identifying material
`fc` (float)	concrete compressive strength (compression is negative)
`e0` (float)	strain at compressive strength
`n` (float)	compressive shape factor
`k` (float)	post-peak compressive shape factor
`alpha1` (float)	α_1 parameter for compressive plastic strain definition
`fcr` (float)	tensile strength
`ecr` (float)	tensile strain at peak stress (fcr)
`b` (float)	exponent of the tension stiffening curve
`alpha2` (float)	α_2 parameter for tensile plastic strain definition

Note:

1. Compressive concrete parameters should be input as negative values.

See also:

Notes

Concrete07

uniaxialMaterial (*'Concrete07'*, *matTag*, *fc*, *ec*, *Ec*, *ft*, *et*, *xp*, *xn*, *r*)

Concrete07 is an implementation of Chang & Mander's 1994 concrete model with simplified unloading and reloading curves. Additionally the tension envelope shift with respect to the origin proposed by Chang and Mander has been removed. The model requires eight input parameters to define the monotonic envelope of confined and unconfined concrete in the following form:

matTag (int)	integer tag identifying material
fc (float)	concrete compressive strength (compression is negative)
ec (float)	concrete strain at maximum compressive strength
Ec (float)	Initial Elastic modulus of the concrete
ft (float)	tensile strength of concrete (tension is positive)
et (float)	tensile strain at max tensile strength of concrete
xp (float)	Non-dimensional term that defines the strain at which the straight line descent begins in tension
xn (float)	Non-dimensional term that defines the strain at which the straight line descent begins in compression
r (float)	Parameter that controls the nonlinear descending branch

See also:

Notes

Concrete01WithSITC

uniaxialMaterial(*'Concrete01WithSITC'*, *matTag*, *fpc*, *epsc0*, *fpcu*, *epsU*, *endStrainSITC=0.01*)

This command is used to construct a modified uniaxial Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength. The modification is to model the effect of Stuff In The Cracks (SITC).

matTag (int)	integer tag identifying material
fpc (float)	concrete compressive strength at 28 days (compression is negative)
epsc0 (float)	concrete strain at maximum strength
fpcu (float)	concrete crushing strength
epsU (float)	concrete strain at crushing strength
endStrainSITC (float)	optional, default = 0.03

Note:

1. Compressive concrete parameters should be input as negative values (if input as positive, they will be converted to negative internally).
2. The initial slope for this model is (2*fpc/epsc0)

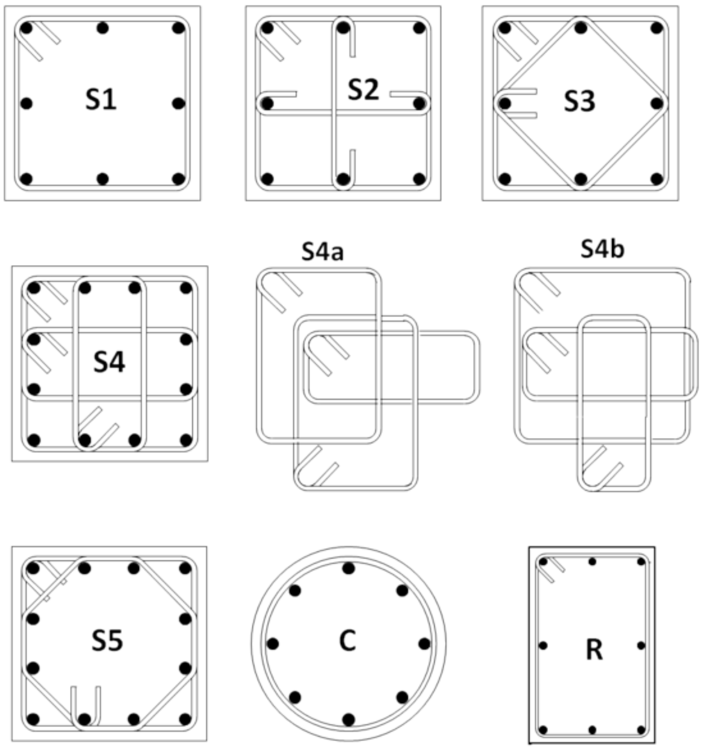
See also:

Notes

ConfinedConcrete01

uniaxialMaterial(*'ConfinedConcrete01'*, *matTag*, *secType*, *fpc*, *Ec*, *'-epscu'*, *epscu*, *'-gamma'*, *gamma*, *'-nu'*, *nu*, *'-varub'*, *'-varnoub'*, *L1*, *L2*, *L3*, *phis*, *S*, *fyh*, *Es0*, *haRatio*, *mu*, *phiLon*, *'-internal'*, **intArgs*, *'-wrap'*, **wrapArgs*, *'-gravel'*, *'-silica'*, *'-tol'*, *tol*, *'-maxNumIter'*, *maxNumIter*, *'-epscuLimit'*, *epscuLimit*, *'-stRatio'*, *stRatio*)

`matTag` (int)	integer tag identifying material
`secType` (str)	tag for the transverse reinforcement configuration. see image below. • `'S1'` square section with S1 type of transverse reinforcement with or without external FRP wrapping • `'S2'` square section with S2 type of transverse reinforcement with or without external FRP wrapping • `'S3'` square section with S3 type of transverse reinforcement with or without external FRP wrapping • `'S4a'` square section with S4a type of transverse reinforcement with or without external FRP wrapping • `'S4b'` square section with S4b type of transverse reinforcement with or without external FRP wrapping • `'S5'` square section with S5 type of transverse reinforcement with or without external FRP wrapping • `'C'` circular section with or without external FRP wrapping • `'R'` rectangular section with or without external FRP wrapping.
`fpc` (float)	unconfined cylindrical strength of concrete specimen.
`Ec` (float)	initial elastic modulus of unconfined concrete.
`epscu` (float)	confined concrete ultimate strain. (optional)
`gamma` (float)	the ratio of the strength corresponding to ultimate strain to the peak strength of the confined concrete stress-strain curve. If `gamma` cannot be achieved in the range [0, epscuLimit] then epscuLimit (optional, default: 0.05) will be assumed as ultimate strain.
`nu` (float)	Poisson's Ratio.
`'-varub'` (float)	Poisson's ratio is defined as a function of axial strain by means of the expression proposed by Braga et al. (2006) with the upper bound equal to 0.5
`'-varnoub'` (float)	Poisson's ratio is defined as a function of axial strain by means of the expression proposed by Braga et al. (2006) without any upper bound.
`L1` (float)	length/diameter of square/circular core section measured respect to the hoop center line.
`L2` (float)	additional dimensions when multiple hoops are being used.
`L3` (float)	additional dimensions when multiple hoops are being used.
`phis` (float)	hoop diameter. If section arrangement has multiple hoops it refers to the external hoop.
`S` (float)	hoop spacing.
`fyh` (float)	yielding strength of the hoop steel.
`Es0` (float)	elastic modulus of the hoop steel.
`haRatio` (float)	hardening ratio of the hoop steel.
`mu` (float)	ductility factor of the hoop steel.
`phiLon` (float)	diameter of longitudinal bars.
`intArgs` (list (float))	`intArgs= [phisi, Si, fyhi, Es0i, haRatioi, mui]` optional parameters for defining the internal transverse reinforcement. If they



See also:

Notes

ConcreteD

uniaxialMaterial('ConcreteD', *matTag*, *fc*, *epsc*, *ft*, *epst*, *Ec*, *alphac*, *alphat*, *cesp=0.25*, *etap=1.15*)

This command is used to construct a concrete material based on the Chinese design code.

`matTag` (int)	integer tag identifying material
`fc` (float)	concrete compressive strength
`epsc` (float)	concrete strain at corresponding to compressive strength
`ft` (float)	concrete tensile strength
`epst` (float)	concrete strain at corresponding to tensile strength
`Ec` (float)	concrete initial Elastic modulus
`alphac` (float)	compressive descending parameter
`alphat` (float)	tensile descending parameter
`cesp` (float)	plastic parameter, recommended values: 0.2~0.3
`etap` (float)	plastic parameter, recommended values: 1.0~1.3

Note:

1. Concrete compressive strength and the corresponding strain should be input as negative values.
2. The value `fc/epsc` and `ft/epst` should be smaller than `Ec`.

See also:

Notes

FRPConfinedConcrete

uniaxialMaterial (*'FRPConfinedConcrete'*, *matTag*, *fpc1*, *fpc2*, *epsc0*, *D*, *c*, *Ej*, *Sj*, *tj*, *eju*, *S*, *fyh*, *dlong*, *dtrans*, *Es*, *vo*, *k*)

This command is used to construct a uniaxial Megalooikonomou-Monti-Santini concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength.

`matTag` (int)	integer tag identifying material
`fpc1` (float)	concrete core compressive strength.
`fpc2` (float)	concrete cover compressive strength.
`epsc0` (float)	strain corresponding to unconfined concrete strength.
`D` (float)	diameter of the circular section.
`c` (float)	dimension of concrete cover (until the outer edge of steel stirrups)
`Ej` (float)	elastic modulus of the fiber reinforced polymer (FRP) jacket.
`Sj` (float)	clear spacing of the FRP strips - zero if FRP jacket is continuous.
`tj` (float)	total thickness of the FRP jacket.
`eju` (float)	rupture strain of the FRP jacket from tensile coupons.
`S` (float)	spacing of the steel spiral/stirrups.
`fyh` (float)	yielding strength of the steel spiral/stirrups.
`dlong` (float)	diameter of the longitudinal bars of the circular section.
`dtrans` (float)	diameter of the steel spiral/stirrups.
`Es` (float)	elastic modulus of steel.
`vo` (float)	initial Poisson's coefficient for concrete.
`k` (float)	reduction factor for the rupture strain of the FRP jacket, recommended values 0.5-0.8.

Note:

1. IMPORTANT: The units of the input parameters should be in MPa, N, mm.
2. Concrete compressive strengths and the corresponding strain should be input as positive values.
3. When rupture of FRP jacket occurs due to dilation of concrete (lateral concrete strain exceeding reduced rupture strain of FRP jacket), the analysis is not terminated. Only a message "FRP Rupture" is plotted on the screen.

See also:

Notes

ConcreteCM

uniaxialMaterial (*'ConcreteCM'*, *matTag*, *fpcc*, *epcc*, *Ec*, *rc*, *xcrn*, *ft*, *et*, *rt*, *xcrp*, *'-GapClose'*, *gap=0*)

This command is used to construct a uniaxialMaterial ConcreteCM (Kolozvari et al., 2015), which is a uniaxial hysteretic constitutive model for concrete developed by Chang and Mander (1994).

matTag (int)	integer tag identifying material
fpcc (float)	Compressive strength (f_c')
epcc (float)	Strain at compressive strength (ϵ_c')
Ec (float)	Initial tangent modulus (E_c)
rc (float)	Shape parameter in Tsai's equation defined for compression (r_c)
xcrn (float)	Non-dimensional critical strain on compression envelope (ϵ_{cr}^{-}, where the envelope curve starts following a straight line)
ft (float)	Tensile strength (f_t)
et (float)	Strain at tensile strength (ϵ_t)
rt (float)	Shape parameter in Tsai's equation defined for tension (r_t)
xcrp (float)	Non-dimensional critical strain on tension envelope (ϵ_{cr}^{+}, where the envelope curve starts following a straight line – large value [e.g., 10000] recommended when tension stiffening is considered)
gap (float)	gap = 0, less gradual gap closure (default); gap = 1, more gradual gap closure

See also:

Notes

Elastic Uniaxial Material

uniaxialMaterial(*'Elastic'*, *matTag*, *E*, *eta=0.0*, *Eneg=E*)
This command is used to construct an elastic uniaxial material object.

matTag (int)	integer tag identifying material
E (float)	tangent
eta (float)	damping tangent (optional, default=0.0)
Eneg (float)	tangent in compression (optional, default=E)

See also:

Notes

Elastic-Perfectly Plastic Material

uniaxialMaterial(*'ElasticPP'*, *matTag*, *E*, *epsyP*, *epsyN=epsyP*, *eps0=0.0*)
This command is used to construct an elastic perfectly-plastic uniaxial material object.

`matTag` (int)	integer tag identifying material
`E` (float)	tangent
`epsyP` (float)	strain or deformation at which material reaches plastic state in tension
`epsyN` (float)	strain or deformation at which material reaches plastic state in compression. (optional, default is tension value)
`eps0` (float)	initial strain (optional, default: zero)

See also:

Notes

Elastic-Perfectly Plastic Gap Material

`uniaxialMaterial`(*'ElasticPPGap'*, *matTag*, *E*, *Fy*, *gap*, *eta=0.0*, *damage='noDamage'*)

This command is used to construct an elastic perfectly-plastic gap uniaxial material object.

`matTag` (int)	integer tag identifying material
`E` (float)	tangent
`Fy` (float)	stress or force at which material reaches plastic state
`gap` (float)	initial gap (strain or deformation)
`eta` (float)	hardening ratio (=Eh/E), which can be negative
`damage` (str)	an optional string to specify whether to accumulate damage or not in the material. With the default string, `'noDamage'` the gap material will re-center on load reversal. If the string `'damage'` is provided this recentering will not occur and gap will grow.

See also:

Notes

Elastic-No Tension Material

`uniaxialMaterial`(*'ENT'*, *matTag*, *E*)

This command is used to construct a uniaxial elastic-no tension material object.

`matTag` (int)	integer tag identifying material
`E` (float)	tangent

See also:

Notes

Parallel Material

uniaxialMaterial(*'Parallel'*, *matTag*, **tags*, *'-factor'*, **facts*)

This command is used to construct a parallel material object made up of an arbitrary number of previously-constructed UniaxialMaterial objects.

`matTag` (int)	integer tag identifying material
`tags` (list (int))	identification tags of materials making up the material model
`facts` (list (float))	factors to create a linear combination of the specified materials. Factors can be negative to subtract one material from an other. (optional, default = 1.0)

See also:

Notes

Series Material

uniaxialMaterial(*'Series'*, *matTag*, **tags*)

This command is used to construct a series material object made up of an arbitrary number of previously-constructed UniaxialMaterial objects.

`matTag` (int)	integer tag identifying material
`tags` (list (int))	identification tags of materials making up the material model

See also:

Notes

PySimple1 Material

uniaxialMaterial(*'PySimple1'*, *matTag*)

This command is used to construct a PySimple1 uniaxial material object.

`matTag` (int)	integer tag identifying material
`soilType` (int)	soilType = 1 Backbone of p-y curve approximates Matlock (1970) soft clay relation. soilType = 2 Backbone of p-y curve approximates API (1993) sand relation.
`pult` (float)	Ultimate capacity of the p-y material. Note that "p" or "pult" are distributed loads [force per length of pile] in common design equations, but are both loads for this uniaxialMaterial [i.e., distributed load times the tributary length of the pile].
`Y50` (float)	Displacement at which 50% of pult is mobilized in monotonic loading.
`Cd` (float)	Variable that sets the drag resistance within a fully-mobilized gap as Cd*pult.
`c` (float)	The viscous damping term (dashpot) on the far-field (elastic) component of the displacement rate (velocity). (optional Default = 0.0). Nonzero c values are used to represent radiation damping effects

See also:

Notes

TzSimple1 Material

uniaxialMaterial (*'TzSimple1'*, *matTag*, *tzType*, *tult*, *z50*, *c=0.0*)
This command is used to construct a TzSimple1 uniaxial material object.

`matTag` (int)	integer tag identifying material
`soilType` (int)	soilType = 1 Backbone of t-z curve approximates Reese and O'Neill (1987). soilType = 2 Backbone of t-z curve approximates Mosher (1984) relation.
`tult` (float)	Ultimate capacity of the t-z material. SEE NOTE 1.
`z50` (float)	Displacement at which 50% of tult is mobilized in monotonic loading.
`c` (float)	The viscous damping term (dashpot) on the far-field (elastic) component of the displacement rate (velocity). (optional Default = 0.0). See NOTE 2.

Note:

1. The argument tult is the ultimate capacity of the t-z material. Note that "t" or "tult" are shear stresses [force per unit area of pile surface] in common design equations, but are both loads for this uniaxialMaterial [i.e., shear stress times the tributary area of the pile].
2. Nonzero c values are used to represent radiation damping effects

See also:

Notes

QzSimple1 Material

uniaxialMaterial (*'QzSimple1'*, *matTag*, *qzType*, *qult*, *Z50*, *suction=0.0*, *cd=0.0*)
This command is used to construct a QzSimple1 uniaxial material object.

`matTag` (int)	integer tag identifying material
`qzType` (int)	qzType = 1 Backbone of q-z curve approximates Reese and O'Neill's (1987) relation for drilled shafts in clay. qzType = 2 Backbone of q-z curve approximates Vijayvergiya's (1977) relation for piles in sand.
`qult` (float)	Ultimate capacity of the q-z material. SEE NOTE 1.
`Z50` (float)	Displacement at which 50% of qult is mobilized in monotonic loading. SEE NOTE 2.
`suction` (float)	Uplift resistance is equal to suction*qult. Default = 0.0. The value of suction must be 0.0 to 0.1.*
`c` (float)	The viscous damping term (dashpot) on the far-field (elastic) component of the displacement rate (velocity). Default = 0.0. Nonzero c values are used to represent radiation damping effects.*

Note:

1. `qult`: Ultimate capacity of the q-z material. Note that `q1` or `qult` are stresses [force per unit area of pile tip] in common design equations, but are both loads for this uniaxialMaterial [i.e., stress times tip area].

2. `Y50`: Displacement at which 50% of pult is mobilized in monotonic loading. Note that Vijayvergiya's relation (qzType=2) refers to a "critical" displacement (zcrit) at which qult is fully mobilized, and that the corresponding z50 would be 0. 125zcrit.
3. optional args `suction` and `c` must either both be omitted or both provided.

See also:

Notes

PyLiq1 Material

uniaxialMaterial(*'PyLiq1'*, *matTag*, *soilType*, *pult*, *Y50*, *Cd*, *c*, *pRes*, *ele1*, *ele2*)

uniaxialMaterial(*'PyLiq1'*, *matTag*, *soilType*, *pult*, *Y50*, *Cd*, *c*, *pRes*, *'-timeSeries'*, *tag*)

This command constructs a uniaxial p-y material that incorporates liquefaction effects. This p y material is used with a zeroLength element to connect a pile (beam-column element) to a 2 D plane-strain FE mesh or displacement boundary condition. The p-y material obtains the average mean effective stress (which decreases with increasing excess pore pressure) either from two specified soil elements, or from a time series. Currently, the implementation requires that the specified soil elements consist of FluidSolidPorousMaterials in FourNodeQuad elements, or PressureDependMultiYield or PressureDependMultiYield02 materials in FourNodeQuadUP or NineFourQuadUP elements. There are two possible forms:

`matTag` (int)	integer tag identifying material
`soilType` (int)	soilType = 1 Backbone of p-y curve approximates Matlock (1970) soft clay relation. soilType = 2 Backbone of p-y curve approximates API (1993) sand relation.
`pult` (float)	Ultimate capacity of the p-y material. Note that "p" or "pult" are distributed loads [force per length of pile] in common design equations, but are both loads for this uniaxialMaterial [i.e., distributed load times the tributary length of the pile].
`Y50` (float)	Displacement at which 50% of pult is mobilized in monotonic loading.
`Cd` (float)	Variable that sets the drag resistance within a fully-mobilized gap as Cd*pult.
`c` (float)	The viscous damping term (dashpot) on the far-field (elastic) component of the displacement rate (velocity). (optional Default = 0.0). Nonzero c values are used to represent radiation damping effects
`pRes` (float)	sets the minimum (or residual) peak resistance that the material retains as the adjacent solid soil elements liquefy
`ele1` `ele2` (float)	are the eleTag (element numbers) for the two solid elements from which PyLiq1 will obtain mean effective stresses and excess pore pressures
`seriesTag` (float)	Alternatively, mean effective stress can be supplied by a time series by specifying the text string `'-timeSeries'` and the tag of the series `seriesTag`.

See also:

Notes

TzLiq1 Material

uniaxialMaterial(*'TzLiq1'*, *matTag*, *tzType*, *tult*, *z50*, *c*, *ele1*, *ele2*)

uniaxialMaterial(*'TzLiq1'*, *matTag*, *tzType*, *tult*, *z50*, *c*, *'-timeSeries'*, *seriesTag*)

The command constructs a uniaxial t-z material that incorporates liquefaction effects. This t z material is used with a zeroLength element to connect a pile (beam-column element) to a 2 D plane-strain FE mesh. The t-z material obtains the average mean effective stress (which decreases with increasing excess pore pressure) from two specified soil elements. Currently, the implementation requires that the specified soil elements consist of FluidSolidPorousMaterials in FourNodeQuad elements.

`matTag` (int)	integer tag identifying material
`soilType` (int)	soilType = 1 Backbone of t-z curve approximates Reese and O'Neill (1987). soilType = 2 Backbone of t-z curve approximates Mosher (1984) relation.
`tult` (float)	Ultimate capacity of the t-z material. SEE NOTE 1.
`Z50` (float)	Displacement at which 50% of tult is mobilized in monotonic loading.
`c` (float)	The viscous damping term (dashpot) on the far-field (elastic) component of the displacement rate (velocity).
`ele1` `ele2` (float)	are the eleTag (element numbers) for the two solid elements from which PyLiq1 will obtain mean effective stresses and excess pore pressures
`seriesTag` (float)	Alternatively, mean effective stress can be supplied by a time series by specifying the text string `'-timeSeries'` and the tag of the seriesm `seriesTag`.

Note:

1. The argument `tult` is the ultimate capacity of the t-z material. Note that "t" or "tult" are shear stresses [force per unit area of pile surface] in common design equations, but are both loads for this uniaxialMaterial [i.e., shear stress times the tributary area of the pile].
2. Nonzero c values are used to represent radiation damping effects
3. To model the effects of liquefaction with `TzLiq1`, it is necessary to use the material stage updating command:

See also:

Notes

Hardening Material

uniaxialMaterial(*'Hardening'*, *matTag*, *E*, *sigmaY*, *H_iso*, *H_kin*, *eta=0.0*)

This command is used to construct a uniaxial material object with combined linear kinematic and isotropic hardening. The model includes optional visco-plasticity using a Perzyna formulation.

`matTag` (int)	integer tag identifying material
`E` (float)	tangent stiffness
`sigmaY` (float)	yield stress or force
`H_iso` (float)	isotropic hardening Modulus
`H_kin` (float)	kinematic hardening Modulus
`eta` (float)	visco-plastic coefficient (optional, default=0.0)

See also:

Notes

CastFuse Material

uniaxialMaterial('Cast', *matTag*, *n*, *bo*, *h*, *fy*, *E*, *L*, *b*, *Ro*, *cR1*, *cR2*, *a1=s2*Pp/Kp*, *a2=1.0*, *a3=a4*Pp/Kp*, *a4=1.0*)

This command is used to construct a parallel material object made up of an arbitrary number of previously-constructed UniaxialMaterial objects.

`matTag` (int)	integer tag identifying material
`n` (int)	Number of yield fingers of the CSF-brace
`bo` (float)	Width of an individual yielding finger at its base of the CSF-brace
`h` (float)	Thickness of an individual yielding finger
`fy` (float)	Yield strength of the steel material of the yielding finger
`E` (float)	Modulus of elasticity of the steel material of the yielding finger
`L` (float)	Height of an individual yielding finger
`b` (float)	Strain hardening ratio
`Ro` (float)	Parameter that controls the Bauschinger effect. Recommended Values for $Ro=between 10 to 30
`cR1` (float)	Parameter that controls the Bauschinger effect. Recommended Value cR1=0.925
`cR2` (float)	Parameter that controls the Bauschinger effect. Recommended Value cR2=0.150
`a1` (float)	isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic deformation of a2*(Pp/Kp)
`a2` (float)	isotropic hardening parameter (see explanation under a1). (optional default = 1.0)
`a3` (float)	isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic deformation of a4*(Pp/Kp)
`a4` (float)	isotropic hardening parameter (see explanation under a3). (optional default = 1.0)

Gray et al. [1] showed that the monotonic backbone curve of a CSF-brace with known properties (`n`, `bo`, `h`, `L`, `fy`, `E`) after yielding can be expressed as a close-form solution that is given by, $P = P_p/\cos(2d/L)$, in which d is the axial deformation of the brace at increment i and P_p is the yield strength of the CSF-brace and is given by the following expression

$P_p = nb_o h^2 f_y/4L$

The elastic stiffness of the CSF-brace is given by,

$K_p = nb_o E h^3 f_y/6L^3$

See also:

Notes

ViscousDamper Material

uniaxialMaterial('ViscousDamper', *matTag*, *K*, *Cd*, *alpha*, *LGap=0.0*, *NM=1*, *RelTol=1e-6*, *AbsTol=1e-10*, *MaxHalf=15*)

This command is used to construct a ViscousDamper material, which represents the Maxwell Model (linear spring and nonlinear dashpot in series). The ViscousDamper material simulates the hysteretic response of nonlinear viscous dampers. An adaptive iterative algorithm has been implemented and validated to solve numerically the constitutive equations within a nonlinear viscous damper with a high-precision accuracy.

`matTag` (int)	integer tag identifying material
`K` (float)	Elastic stiffness of linear spring to model the axial flexibility of a viscous damper (e.g. combined stiffness of the supporting brace and internal damper portion)
`Cd` (float)	Damping coefficient
`alpha` (float)	Velocity exponent
`LGap` (float)	Gap length to simulate the gap length due to the pin tolerance
`NM` (int)	Employed adaptive numerical algorithm (default value NM = 1; • `1` = Dormand-Prince54, • `2` = 6th order Adams-Bashforth-Moulton, • `3` = modified Rosenbrock Triple)
`RelTol` (float)	Tolerance for absolute relative error control of the adaptive iterative algorithm (default value 10^-6)
`AbsTol` (float)	Tolerance for absolute error control of adaptive iterative algorithm (default value 10^-10)
`MaxHalf` (int)	Maximum number of sub-step iterations within an integration step (default value 15)

See also:

Notes

BilinearOilDamper Material

uniaxialMaterial(*'BilinearOilDamper'*, *matTag*, *K*, *Cd*, *Fr=1.0*, *p=1.0*, *LGap=0.0*, *NM=1*, *RelTol=1e-6*, *AbsTol=1e-10*, *MaxHalf=15*)

This command is used to construct a BilinearOilDamper material, which simulates the hysteretic response of bilinear oil dampers with relief valve. Two adaptive iterative algorithms have been implemented and validated to solve numerically the constitutive equations within a bilinear oil damper with a high-precision accuracy.

matTag (int)	integer tag identifying material
K (float)	Elastic stiffness of linear spring to model the axial flexibility of a viscous damper (e.g. combined stiffness of the supporting brace and internal damper portion)
Cd (float)	Damping coefficient
Fr (float)	Damper relief load (default=1.0, Damper property)
p (float)	Post-relief viscous damping coefficient ratio (default=1.0, linear oil damper)
LGap (float)	Gap length to simulate the gap length due to the pin tolerance
NM (int)	Employed adaptive numerical algorithm (default value NM = 1; • 1 = Dormand-Prince54, • 2 = 6th order Adams-Bashforth-Moulton, • 3 = modified Rosenbrock Triple)
RelTol (float)	Tolerance for absolute relative error control of the adaptive iterative algorithm (default value 10^-6)
AbsTol (float)	Tolerance for absolute error control of adaptive iterative algorithm (default value 10^-10)
MaxHalf (int)	Maximum number of sub-step iterations within an integration step (default value 15)

See also:

Notes

Modified Ibarra-Medina-Krawinkler Deterioration Model with Bilinear Hysteretic Response (Bilin Material)

uniaxialMaterial('Bilin', *matTag*, *K0*, *as_Plus*, *as_Neg*, *My_Plus*, *My_Neg*, *Lamda_S*, *Lamda_C*, *Lamda_A*, *Lamda_K*, *c_S*, *c_C*, *c_A*, *c_K*, *theta_p_Plus*, *theta_p_Neg*, *theta_pc_Plus*, *theta_pc_Neg*, *Res_Pos*, *Res_Neg*, *theta_u_Plus*, *theta_u_Neg*, *D_Plus*, *D_Neg*, *nFactor=0.0*)

This command is used to construct a bilin material. The bilin material simulates the modified Ibarra-Krawinkler deterioration model with bilinear hysteretic response. Note that the hysteretic response of this material has been calibrated with respect to more than 350 experimental data of steel beam-to-column connections and multivariate regression formulas are provided to estimate the deterioration parameters of the model for different connection types. These relationships were developed by Lignos and Krawinkler (2009, 2011) and have been adopted by PEER/ATC (2010). The input parameters for this component model can be computed interactively from this link. **Use the module Component Model.**

matTag (int)	integer tag identifying material
K0 (float)	elastic stiffness
as_Plus (float)	strain hardening ratio for positive loading direction
as_Neg (float)	strain hardening ratio for negative loading direction
My_Plus (float)	effective yield strength for positive loading direction
My_Neg (float)	effective yield strength for negative loading direction (negative value)
Lamda_S (float)	Cyclic deterioration parameter for strength deterioration [E_t=Lamda_S*M_y; set Lamda_S = 0 to disable this mode of deterioration]
Lamda_C (float)	Cyclic deterioration parameter for post-capping strength deterioration [E_t=Lamda_C*M_y; set Lamda_C = 0 to disable this mode of deterioration]
Lamda_A (float)	Cyclic deterioration parameter for acceleration reloading stiffness deterioration (is not a deterioration mode for a component with Bilinear hysteretic response) [Input value is required, but not used; set Lamda_A = 0].
Lamda_K (float)	Cyclic deterioration parameter for unloading stiffness deterioration [E_t=Lamda_K*M_y; set Lamda_k = 0 to disable this mode of deterioration]
c_S (float)	rate of strength deterioration. The default value is 1.0.
c_C (float)	rate of post-capping strength deterioration. The default value is 1.0.
c_A (float)	rate of accelerated reloading deterioration. The default value is 1.0.
c_K (float)	rate of unloading stiffness deterioration. The default value is 1.0.
theta_p_Plus (float)	pre-capping rotation for positive loading direction (often noted as plastic rotation capacity)
theta_p_Neg (float)	pre-capping rotation for negative loading direction (often noted as plastic rotation capacity) (positive value)
theta_pc_Plus (float)	post-capping rotation for positive loading direction
theta_pc_Neg (float)	post-capping rotation for negative loading direction (positive value)
Res_Pos (float)	residual strength ratio for positive loading direction
Res_Neg (float)	residual strength ratio for negative loading direction (positive value)
theta_u_Plus (float)	ultimate rotation capacity for positive loading direction
theta_u_Neg (float)	ultimate rotation capacity for negative loading direction (positive value)
D_Plus (float)	rate of cyclic deterioration in the positive loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.
D_Neg (float)	rate of cyclic deterioration in the negative loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.
nFactor (float)	elastic stiffness amplification factor, mainly for use with concentrated plastic hinge elements (optional, default = 0).

See also:

Notes

Modified Ibarra-Medina-Krawinkler Deterioration Model with Peak-Oriented Hysteretic Response (ModIMKPeakOriented Material)

uniaxialMaterial(*'ModIMKPeakOriented'*, *matTag*, *K0*, *as_Plus*, *as_Neg*, *My_Plus*, *My_Neg*, *Lamda_S*, *Lamda_C*, *Lamda_A*, *Lamda_K*, *c_S*, *c_C*, *c_A*, *c_K*, *theta_p_Plus*, *theta_p_Neg*, *theta_pc_Plus*, *theta_pc_Neg*, *Res_Pos*, *Res_Neg*, *theta_u_Plus*, *theta_u_Neg*, *D_Plus*, *D_Neg*)

This command is used to construct a ModIMKPeakOriented material. This material simulates the modified Ibarra-Medina-Krawinkler deterioration model with peak-oriented hysteretic response. Note that the hysteretic response of this material has been calibrated with respect to 200 experimental data of RC beams in order to estimate the deterioration parameters of the model. This information was developed by Lignos and Krawinkler (2012). NOTE: before you use this material make sure that you have downloaded the latest OpenSees version. A youtube video presents a summary of this model including the way to be used within openSees youtube link.

matTag (int)	integer tag identifying material
K0 (float)	elastic stiffness
as_Plus (float)	strain hardening ratio for positive loading direction
as_Neg (float)	strain hardening ratio for negative loading direction
My_Plus (float)	effective yield strength for positive loading direction
My_Neg (float)	effective yield strength for negative loading direction (negative value)
Lamda_S (float)	Cyclic deterioration parameter for strength deterioration [E_t=Lamda_S*M_y, see Lignos and Krawinkler (2011); set Lamda_S = 0 to disable this mode of deterioration]
Lamda_C (float)	Cyclic deterioration parameter for post-capping strength deterioration [E_t=Lamda_C*M_y, see Lignos and Krawinkler (2011); set Lamda_C = 0 to disable this mode of deterioration]
Lamda_A (float)	Cyclic deterioration parameter for accelerated reloading stiffness deterioration [E_t=Lamda_A*M_y, see Lignos and Krawinkler (2011); set Lamda_A = 0 to disable this mode of deterioration]
Lamda_K (float)	Cyclic deterioration parameter for unloading stiffness deterioration [E_t=Lamda_K*M_y, see Lignos and Krawinkler (2011); set Lamda_K = 0 to disable this mode of deterioration]
c_S (float)	rate of strength deterioration. The default value is 1.0.
c_C (float)	rate of post-capping strength deterioration. The default value is 1.0.
c_A (float)	rate of accelerated reloading deterioration. The default value is 1.0.
c_K (float)	rate of unloading stiffness deterioration. The default value is 1.0.
theta_p_Plus (float)	pre-capping rotation for positive loading direction (often noted as plastic rotation capacity)
theta_p_Neg (float)	pre-capping rotation for negative loading direction (often noted as plastic rotation capacity) (must be defined as a positive value)
theta_pc_Plus (float)	post-capping rotation for positive loading direction
theta_pc_Neg (float)	post-capping rotation for negative loading direction (must be defined as a positive value)
Res_Pos (float)	residual strength ratio for positive loading direction
Res_Neg (float)	residual strength ratio for negative loading direction (must be defined as a positive value)
theta_u_Plus (float)	ultimate rotation capacity for positive loading direction
theta_u_Neg (float)	ultimate rotation capacity for negative loading direction (must be defined as a positive value)
D_Plus (float)	rate of cyclic deterioration in the positive loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.
D_Neg (float)	rate of cyclic deterioration in the negative loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.

See also:

Notes

Modified Ibarra-Medina-Krawinkler Deterioration Model with Pinched Hysteretic Response (ModIMKPinching Material)

uniaxialMaterial(*'ModIMKPinching'*, *matTag*, *K0*, *as_Plus*, *as_Neg*, *My_Plus*, *My_Neg*, *FprPos*, *FprNeg*, *A_pinch*, *Lamda_S*, *Lamda_C*, *Lamda_A*, *Lamda_K*, *c_S*, *c_C*, *c_A*, *c_K*, *theta_p_Plus*, *theta_p_Neg*, *theta_pc_Plus*, *theta_pc_Neg*, *Res_Pos*, *Res_Neg*, *theta_u_Plus*, *theta_u_Neg*, *D_Plus*, *D_Neg*)

This command is used to construct a ModIMKPinching material. This material simulates the modified Ibarra-Medina-Krawinkler deterioration model with pinching hysteretic response. NOTE: **before you use this material make sure that you have downloaded the latest OpenSees version**. A youtube video presents a summary of this model including the way to be used within openSees youtube link.

matTag (int)	integer tag identifying material
K0 (float)	elastic stiffness
as_Plus (float)	strain hardening ratio for positive loading direction
as_Neg (float)	strain hardening ratio for negative loading direction
My_Plus (float)	effective yield strength for positive loading direction
My_Neg (float)	effective yield strength for negative loading direction (Must be defined as a negative value)
FprPos (float)	Ratio of the force at which reloading begins to force corresponding to the maximum historic deformation demand (positive loading direction)
FprNeg (float)	Ratio of the force at which reloading begins to force corresponding to the absolute maximum historic deformation demand (negative loading direction)
A_Pinch (float)	Ratio of reloading stiffness
Lamda_S (float)	Cyclic deterioration parameter for strength deterioration [E_t=Lamda_S*M_y, see Lignos and Krawinkler (2011); set Lamda_S = 0 to disable this mode of deterioration]
Lamda_C (float)	Cyclic deterioration parameter for post-capping strength deterioration [E_t=Lamda_C*M_y, see Lignos and Krawinkler (2011); set Lamda_C = 0 to disable this mode of deterioration]
Lamda_A (float)	Cyclic deterioration parameter for accelerated reloading stiffness deterioration [E_t=Lamda_A*M_y, see Lignos and Krawinkler (2011); set Lamda_A = 0 to disable this mode of deterioration]
Lamda_K (float)	Cyclic deterioration parameter for unloading stiffness deterioration [E_t=Lamda_K*M_y, see Lignos and Krawinkler (2011); set Lamda_K = 0 to disable this mode of deterioration]
c_S (float)	rate of strength deterioration. The default value is 1.0.
c_C (float)	rate of post-capping strength deterioration. The default value is 1.0.
c_A (float)	rate of accelerated reloading deterioration. The default value is 1.0.
c_K (float)	rate of unloading stiffness deterioration. The default value is 1.0.
theta_p_Plus (float)	pre-capping rotation for positive loading direction (often noted as plastic rotation capacity)
theta_p_Neg (float)	pre-capping rotation for negative loading direction (often noted as plastic rotation capacity) (must be defined as a positive value)
theta_pc_Plus (float)	post-capping rotation for positive loading direction
theta_pc_Neg (float)	post-capping rotation for negative loading direction (must be defined as a positive value)
Res_Pos (float)	residual strength ratio for positive loading direction
Res_Neg (float)	residual strength ratio for negative loading direction (must be defined as a positive value)
theta_u_Plus (float)	ultimate rotation capacity for positive loading direction
theta_u_Neg (float)	ultimate rotation capacity for negative loading direction (must be defined as a positive value)
D_Plus (float)	rate of cyclic deterioration in the positive loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.
D_Neg (float)	rate of cyclic deterioration in the negative loading direction (this parameter is used to create assymetric hysteretic behavior for the case of a composite beam). For symmetric hysteretic response use 1.0.

See also:

Notes

SAWS Material

uniaxialMaterial('SAWS', *matTag*, *F0*, *FI*, *DU*, *S0*, *R1*, *R2*, *R3*, *R4*, *alph*, *beta*)

This file contains the class definition for SAWSMaterial. SAWSMaterial provides the implementation of a one-dimensional hysteretic model develeped as part of the CUREe Caltech wood frame project.

`matTag` (int)	integer tag identifying material
`F0` (float)	Intercept strength of the shear wall spring element for the asymtotic line to the envelope curve F0 > FI > 0
`FI` (float)	Intercept strength of the spring element for the pinching branch of the hysteretic curve. (FI > 0).
`DU` (float)	Spring element displacement at ultimate load. (DU > 0).
`S0` (float)	Initial stiffness of the shear wall spring element (S0 > 0).
`R1` (float)	Stiffness ratio of the asymptotic line to the spring element envelope curve. The slope of this line is R1 S0. (0 < R1 < 1.0).
`R2` (float)	Stiffness ratio of the descending branch of the spring element envelope curve. The slope of this line is R2 S0. (R2 < 0).
`R3` (float)	Stiffness ratio of the unloading branch off the spring element envelope curve. The slope of this line is R3 S0. (R3 1).
`R4` (float)	Stiffness ratio of the pinching branch for the spring element. The slope of this line is R4 S0. (R4 > 0).
`alpha` (float)	Stiffness degradation parameter for the shear wall spring element. (ALPHA > 0).
`beta` (float)	Stiffness degradation parameter for the spring element. (BETA > 0).

See also:

Notes

BarSlip Material

uniaxialMaterial('BarSlip', *matTag*, *fc*, *fy*, *Es*, *fu*, *Eh*, *db*, *ld*, *nb*, *depth*, *height*, *ancLratio=1.0*, *bsFlag*, *type*, *damage='Damage'*, *unit='psi'*)

This command is used to construct a uniaxial material that simulates the bar force versus slip response of a reinforcing bar anchored in a beam-column joint. The model exhibits degradation under cyclic loading. Cyclic degradation of strength and stiffness occurs in three ways: unloading stiffness degradation, reloading stiffness degradation, strength degradation.

`matTag` (int)	integer tag identifying material
`fc` (float)	positive floating point value defining the compressive strength of the concrete in which the reinforcing bar is anchored
`fy` (float)	positive floating point value defining the yield strength of the reinforcing steel
`Es` (float)	floating point value defining the modulus of elasticity of the reinforcing steel
`fu` (float)	positive floating point value defining the ultimate strength of the reinforcing steel
`Eh` (float)	floating point value defining the hardening modulus of the reinforcing steel
`ld` (float)	floating point value defining the development length of the reinforcing steel
`db` (float)	point value defining the diameter of reinforcing steel
`nb` (float)	an integer defining the number of anchored bars
`depth` (float)	floating point value defining the dimension of the member (beam or column) perpendicular to the dimension of the plane of the paper
`height` (float)	floating point value defining the height of the flexural member, perpendicular to direction in which the reinforcing steel is placed, but in the plane of the paper
`ancLratio` (float)	floating point value defining the ratio of anchorage length used for the reinforcing bar to the dimension of the joint in the direction of the reinforcing bar (optional, default: 1.0)
`bsFlag` (str)	string indicating relative bond strength for the anchored reinforcing bar (options: `'Strong'` or `'Weak'`)
`type` (str)	string indicating where the reinforcing bar is placed. (options: `'beamtop'`, `'beambot'` or `'column'`)
`damage` (str)	string indicating type of damage:whether there is full damage in the material or no damage (optional, options: `'Damage'`, `'NoDamage'` ; default: `'Damage'`)
`unit` (str)	string indicating the type of unit system used (optional, options: `'psi'`, `'MPa'`, `'Pa'`, `'psf'`, `'ksi'`, `'ksf'`) (default: `'psi'` / `'MPa'`)

See also:

Notes

Bond SP01 - - Strain Penetration Model for Fully Anchored Steel Reinforcing Bars

uniaxialMaterial(*'Bond_SP01'*, *matTag*, *Fy*, *Sy*, *Fu*, *Su*, *b*, *R*)

This command is used to construct a uniaxial material object for capturing strain penetration effects at the column-to-footing, column-to-bridge bent caps, and wall-to-footing intersections. In these cases, the bond slip associated with strain penetration typically occurs along a portion of the anchorage length. This model can also be applied to the beam end regions, where the strain penetration may include slippage of the bar along the entire anchorage length, but the model parameters should be chosen appropriately.

This model is for fully anchored steel reinforcement bars that experience bond slip along a portion of the anchorage length due to strain penetration effects, which are usually the case for column and wall longitudinal bars anchored into footings or bridge joints

`matTag` (int)	integer tag identifying material
`Fy` (float)	Yield strength of the reinforcement steel
`Sy` (float)	Rebar slip at member interface under yield stress. (see NOTES below)
`Fu` (float)	Ultimate strength of the reinforcement steel
`Su` (float)	Rebar slip at the loaded end at the bar fracture strength
`b` (float)	Initial hardening ratio in the monotonic slip vs. bar stress response (0.3~0.5)
`R` (float)	Pinching factor for the cyclic slip vs. bar response (0.5~1.0)

See also:

Notes

Fatigue Material

uniaxialMaterial (*'Fatigue'*, *matTag*, *tag*, *'-E0'*, *E0=0.191*, *'-m'*, *m=-0.458*, *'-min'*, *min=-1e16*, *'-max'*, *max=1e16*)

The fatigue material uses a modified rainflow cycle counting algorithm to accumulate damage in a material using Miner's Rule. Element stress/strain relationships become zero when fatigue life is exhausted.

`matTag` (int)	integer tag identifying material
`tag` (float)	Unique material object integer tag for the material that is being wrapped
`E0` (float)	Value of strain at which one cycle will cause failure (default 0.191)
`m` (float)	Slope of Coffin-Manson curve in log-log space (default -0.458)
`min` (float)	Global minimum value for strain or deformation (default -1e16)
`max` (float)	Global maximum value for strain or deformation (default 1e16)

See also:

Notes

Impact Material

uniaxialMaterial (*'ImpactMaterial'*, *matTag*, *K1*, *K2*, *sigy*, *gap*)

This command is used to construct an impact material object

`matTag` (int)	integer tag identifying material
`K1` (float)	initial stiffness
`K2` (float)	secondary stiffness
`sigy` (float)	yield displacement
`gap` (float)	initial gap

See also:

Notes

Hyperbolic Gap Material

uniaxialMaterial (*'HyperbolicGapMaterial'*, *matTag*, *Kmax*, *Kur*, *Rf*, *Fult*, *gap*)

This command is used to construct a hyperbolic gap material object.

`matTag` (int)	integer tag identifying material
`Kmax` (float)	initial stiffness
`Kur` (float)	unloading/reloading stiffness
`Rf` (float)	failure ratio
`Fult` (float)	ultimate (maximum) passive resistance
`gap` (float)	initial gap

Note:

1. This material is implemented as a compression-only gap material. `Fult` and `gap` should be input as negative values.

2. Recomended Values:
   - `Kmax` = 20300 kN/m of abutment width
   - `Kcur` = `Kmax`
   - `Rf` = 0.7
   - `Fult` = -326 kN per meter of abutment width
   - `gap` = -2.54 cm

See also:

Notes

Limit State Material

uniaxialMaterial(*'LimitState'*, *matTag*, *s1p*, *e1p*, *s2p*, *e2p*, *s3p*, *e3p*, *s1n*, *e1n*, *s2n*, *e2n*, *s3n*, *e3n*, *pinchX*, *pinchY*, *damage1*, *damage2*, *beta*, *curveTag*, *curveType*)

This command is used to construct a uniaxial hysteretic material object with pinching of force and deformation, damage due to ductility and energy, and degraded unloading stiffness based on ductility. Failure of the material is defined by the associated Limit Curve.

`matTag` (int)	integer tag identifying material
`s1p` `e1p` (float)	stress and strain (or force & deformation) at first point of the envelope in the positive direction
`s2p` `e2p` (float)	stress and strain (or force & deformation) at second point of the envelope in the positive direction
`s3p` `e3p` (float)	stress and strain (or force & deformation) at third point of the envelope in the positive direction
`s1n` `e1n` (float)	stress and strain (or force & deformation) at first point of the envelope in the negative direction
`s2n` `e2n` (float)	stress and strain (or force & deformation) at second point of the envelope in the negative direction
`s3n` `e3n` (float)	stress and strain (or force & deformation) at third point of the envelope in the negative direction
`pinchX` (float)	pinching factor for strain (or deformation) during reloading
`pinchY` (float)	pinching factor for stress (or force) during reloading
`damage1` (float)	damage due to ductility: D1(m-1)
`damage2` (float)	damage due to energy: D2(Ei/Eult)
`beta` (float)	power used to determine the degraded unloading stiffness based on ductility, m-b (optional, default=0.0)
`curveTag` (int)	an integer tag for the Limit Curve defining the limit surface
`curveType` (int)	an integer defining the type of LimitCurve (0 = no curve, 1 = axial curve, all other curves can be any other integer)

Note:

- negative backbone points should be entered as negative numeric values

---

See also:

Notes

MinMax Material

uniaxialMaterial(*'MinMax'*, *matTag*, *otherTag*, *'-min'*, *minStrain=1e-16*, *'-max'*, *maxStrain=1e16*)

This command is used to construct a MinMax material object. This stress-strain behaviour for this material is provided by another material. If however the strain ever falls below or above certain threshold values, the other material is assumed to have failed. From that point on, values of 0.0 are returned for the tangent and stress.

`matTag` (int)	integer tag identifying material
`otherTag` (float)	tag of the other material
`minStrain` (float)	minimum value of strain. optional default = -1.0e16.
`maxStrain` (float)	max value of strain. optional default = 1.0e16.

See also:

Notes

ElasticBilin Material

uniaxialMaterial(*'ElasticBilin'*, *matTag*, *EP1*, *EP2*, *epsP2*, *EN1=EP1*, *EN2=EP2*, *epsN2=-epsP2*)

This command is used to construct an elastic bilinear uniaxial material object. Unlike all other bilinear materials, the unloading curve follows the loading curve exactly.

`matTag` (int)	integer tag identifying material
`EP1` (float)	tangent in tension for stains: 0 <= strains <= `epsP2`
`EP2` (float)	tangent when material in tension with strains > `epsP2`
`epsP2` (float)	strain at which material changes tangent in tension.
`EN1` (float)	optional, default = `EP1`. tangent in compression for stains: 0 < strains <= `epsN2`
`EN2` (float)	optional, default = `EP2`. tangent in compression with strains < `epsN2`
`epsN2` (float)	optional, default = `-epsP2`. strain at which material changes tangent in compression.

Note: `eps0` can not be controlled. It is always zero.

See also:

Notes

ElasticMultiLinear Material

uniaxialMaterial(*'ElasticMultiLinear'*, *matTag*, *eta=0.0*, *'-strain'*, **strainPoints*, *'-stress'*, **stressPoints*)

This command is used to construct a multi-linear elastic uniaxial material object. The nonlinear stress-strain relationship is given by a multi-linear curve that is define by a set of points. The behavior is nonlinear but it is elastic. This means that the material loads and unloads along the same curve, and no energy is dissipated. The slope given by the last two specified points on the positive strain axis is extrapolated to infinite positive strain.

Similarly, the slope given by the last two specified points on the negative strain axis is extrapolated to infinite negative strain. The number of provided strain points needs to be equal to the number of provided stress points.

`matTag` (int)	integer tag identifying material
`eta` (float)	damping tangent (optional, default=0.0)
`strainPoints` (list (float))	list of strain points along stress-strain curve
`stressPoints` (list (float))	list of stress points along stress-strain curve

See also:

Notes

MultiLinear

`uniaxialMaterial`(*'MultiLinear'*, *matTag*, **pts*)

This command is used to construct a uniaxial multilinear material object.

`matTag` (int)	integer tag identifying material
`pts` (list (float))	a list of strain and stress points `pts = [strain1, stress1, strain2, stress2, ..., ]`

See also:

Notes

Initial Strain Material

`uniaxialMaterial`(*'InitStrainMaterial'*, *matTag*, *otherTag*, *initStrain*)

This command is used to construct an Initial Strain material object. The stress-strain behaviour for this material is defined by another material. Initial Strain Material enables definition of initial strains for the material under consideration. The stress that corresponds to the initial strain will be calculated from the other material.

`matTag` (int)	integer tag identifying material
`otherTag` (int)	tag of the other material
`initStrain` (float)	initial strain

See also:

Notes

Initial Stress Material

`uniaxialMaterial`(*'InitStressMaterial'*, *matTag*, *otherTag*, *initStress*)

This command is used to construct an Initial Stress material object. The stress-strain behaviour for this material is defined by another material. Initial Stress Material enables definition of initial stress for the material under consideration. The strian that corresponds to the initial stress will be calculated from the other material.

`matTag` (int)	integer tag identifying material
`otherTag` (float)	tag of the other material
`initStress` (float)	initial stress

See also:

Notes

PathIndependent Material

uniaxialMaterial(*'PathIndependent'*, *matTag*, *tag*)
This command is to create a PathIndependent material

`matTag` (int)	integer tag identifying material
`tag` (int)	a pre-defined material

Pinching4 Material

uniaxialMaterial(*'Pinching4'*, *matTag*, *ePf1*, *ePd1*, *ePf2*, *ePd2*, *ePf3*, *ePd3*, *ePf4*, *ePd4*[, *eNf1*, *eNd1*, *eNf2*, *eNd2*, *eNf3*, *eNd3*, *eNf4*, *eNd4*], *rDispP*, *rForceP*, *uForceP*[, *rDispN*, *rForceN*, *uForceN*], *gK1*, *gK2*, *gK3*, *gK4*, *gKLim*, *gD1*, *gD2*, *gD3*, *gD4*, *gDLim*, *gF1*, *gF2*, *gF3*, *gF4*, *gFLim*, *gE*, *dmgType*)
This command is used to construct a uniaxial material that represents a 'pinched' load-deformation response and exhibits degradation under cyclic loading. Cyclic degradation of strength and stiffness occurs in three ways: unloading stiffness degradation, reloading stiffness degradation, strength degradation.

matTag (int)	integer tag identifying material
ePf1 ePf2 ePf3 ePf4 (float)	floating point values defining force points on the positive response envelope
ePd1 ePd2 ePd3 ePd4 (float)	floating point values defining deformation points on the positive response envelope
eNf1 eNf2 eNf3 eNf4 (float)	floating point values defining force points on the negative response envelope
eNd1 eNd2 eNd3 eNd4 (float)	floating point values defining deformation points on the negative response envelope
rDispP (float)	floating point value defining the ratio of the deformation at which reloading occurs to the maximum historic deformation demand
fFoceP (float)	floating point value defining the ratio of the force at which reloading begins to force corresponding to the maximum historic deformation demand
uForceP (float)	floating point value defining the ratio of strength developed upon unloading from negative load to the maximum strength developed under monotonic loading
rDispN (float)	floating point value defining the ratio of the deformation at which reloading occurs to the minimum historic deformation demand
fFoceN (float)	floating point value defining the ratio of the force at which reloading begins to force corresponding to the minimum historic deformation demand
uForceN (float)	floating point value defining the ratio of strength developed upon unloading from negative load to the minimum strength developed under monotonic loading
gK1 gK2 gK3 gK4 gKLim (float)	floating point values controlling cyclic degradation model for unloading stiffness degradation
gD1 gD2 gD3 gD4 gDLim (float)	floating point values controlling cyclic degradation model for reloading stiffness degradation
gF1 gF2 gF3 gF4 gFLim (float)	floating point values controlling cyclic degradation model for strength degradation
gE (float)	floating point value used to define maximum energy dissipation under cyclic loading. Total energy dissipation capacity is defined as this factor multiplied by the energy dissipated under monotonic loading.
dmgType (str)	string to indicate type of damage (option: 'cycle', 'energy')

See also:

Notes

Engineered Cementitious Composites Material

uniaxialMaterial(*'ECC01'*, *matTag*, *sigt0*, *epst0*, *sigt1*, *epst1*, *epst2*, *sigc0*, *epsc0*, *epsc1*, *alphaT1*, *alphaT2*, *alphaC*, *alphaCU*, *betaT*, *betaC*)

This command is used to construct a uniaxial Engineered Cementitious Composites (ECC)material object based on the ECC material model of Han, et al. (see references). Reloading in tension and compression is linear.

matTag (int)	integer tag identifying material
sigt0 (float)	tensile cracking stress
epst0 (float)	strain at tensile cracking stress
sigt1 (float)	peak tensile stress
epst1 (float)	strain at peak tensile stress
sigt2 (float)	ultimate tensile strain
sigc0 (float)	compressive strength (see NOTES)
epsc0 (float)	strain at compressive strength (see NOTES)
epsc1 (float)	ultimate compressive strain (see NOTES)
alphaT1 (float)	exponent of the unloading curve in tensile strain hardening region
alphaT2 (float)	exponent of the unloading curve in tensile softening region
alphaC (float)	exponent of the unloading curve in the compressive softening
alphaCU (float)	exponent of the compressive softening curve (use 1 for linear softening)
betaT (float)	parameter to determine permanent strain in tension
betaC (float)	parameter to determine permanent strain in compression

See also:

Notes

SelfCentering Material

uniaxialMaterial(*'SelfCentering'*, *matTag*, *k1*, *k2*, *sigAct*, *beta*[, *epsSlip*, *epsBear*, *rBear*])

This command is used to construct a uniaxial self-centering (flag-shaped) material object with optional non-recoverable slip behaviour and an optional stiffness increase at high strains (bearing behaviour).

matTag (int)	integer tag identifying material
k1 (float)	Initial Stiffness
k2 (float)	Post-Activation Stiffness (0< k2``< ``k1)
sigAct (float)	Forward Activation Stress/Force
beta (float)	Ratio of Forward to Reverse Activation Stress/Force
epsSlip (float)	slip Strain/Deformation (if epsSlip = 0, there will be no slippage)
epsBear (float)	Bearing Strain/Deformation (if epsBear = 0, there will be no bearing)
rBear (float)	Ratio of Bearing Stiffness to Initial Stiffness k1

See also:

Notes

Viscous Material

uniaxialMaterial(*'Viscous'*, *matTag*)

This command is used to construct a uniaxial viscous material object. stress =C(strain-rate)^alpha

matTag (int)	integer tag identifying material
C (float)	damping coeficient
alpha (float)	power factor (=1 means linear damping)

Note:

1. This material can only be assigned to truss and zeroLength elements.
2. This material can not be combined in parallel/series with other materials. When defined in parallel with other materials it is ignored.

See also:

Notes

BoucWen Material

uniaxialMaterial (*'BoucWen'*, *matTag*, *alpha*, *ko*, *n*, *gamma*, *beta*, *Ao*, *deltaA*, *deltaNu*, *deltaEta*)

This command is used to construct a uniaxial Bouc-Wen smooth hysteretic material object. This material model is an extension of the original Bouc-Wen model that includes stiffness and strength degradation (Baber and Noori (1985)).

`matTag` (int)	integer tag identifying material
`alpha` (float)	ratio of post-yield stiffness to the initial elastic stiffenss (0< α <1)
`ko` (float)	initial elastic stiffness
`n` (float)	parameter that controls transition from linear to nonlinear range (as n increases the transition becomes sharper; n is usually grater or equal to 1)
`gamma` `beta` (float)	parameters that control shape of hysteresis loop; depending on the values of γ and β softening, hardening or quasi-linearity can be simulated (look at the NOTES)
`Ao` `deltaA` (float)	parameters that control tangent stiffness
`deltaNu` `deltaEta` (float)	parameters that control material degradation

See also:

Notes

BWBN Material

uniaxialMaterial (*'BWBN'*, *matTag*, *alpha*, *ko*, *n*, *gamma*, *beta*, *Ao*, *q*, *zetas*, *p*, *Shi*, *deltaShi*, *lambda*, *tol*, *maxIter*)

This command is used to construct a uniaxial Bouc-Wen pinching hysteretic material object. This material model is an extension of the original Bouc-Wen model that includes pinching (Baber and Noori (1986) and Foliente (1995)).

`matTag` (int)	integer tag identifying material
`alpha` (float)	ratio of post-yield stiffness to the initial elastic stiffenss (0< α <1)
`ko` (float)	initial elastic stiffness
`n` (float)	parameter that controls transition from linear to nonlinear range (as n increases the transition becomes sharper; n is usually grater or equal to 1)
`gamma` `beta` (float)	parameters that control shape of hysteresis loop; depending on the values of γ and β softening, hardening or quasi-linearity can be simulated (look at the BoucWen Material)
`Ao` (float)	parameter that controls tangent stiffness
`q` `zetas` `p` `Shi` `deltaShi` `lambda` (float)	parameters that control pinching
`tol` (float)	tolerance
`maxIter` (float)	maximum iterations

See also:

Notes

KikuchiAikenHDR Material

`uniaxialMaterial`(*'KikuchiAikenHDR'*, *matTag*, *tp*, *ar*, *hr*[, *'-coGHU'*, *cg*, *ch*, *cu*][, *'-coMSS'*, *rs*, *rf*])

This command is used to construct a uniaxial KikuchiAikenHDR material object. This material model produces nonlinear hysteretic curves of high damping rubber bearings (HDRs).

`matTag` (int)	integer tag identifying material
`tp` (str)	rubber type (see note 1)
`ar` (float)	area of rubber [unit: m^2] (see note 2)
`hr` (float)	total thickness of rubber [unit: m] (see note 2)
`cg` `ch` `cu` (float)	correction coefficients for equivalent shear modulus (`cg`), equivalent viscous daming ratio (`ch`), ratio of shear force at zero displacement (`cu`).
`rs` `rf` (float)	reduction rate for stiffness (`rs`) and force (`rf`) (see note 3)

Note:

1. Following rubber types for `tp` are available:
    - `'X0.6'` Bridgestone X0.6, standard compressive stress, up to 400% shear strain
    - `'X0.6-0MPa'` Bridgestone X0.6, zero compressive stress, up to 400% shear strain
    - `'X0.4'` Bridgestone X0.4, standard compressive stress, up to 400% shear strain
    - `'X0.4-0MPa'` Bridgestone X0.4, zero compressive stress, up to 400% shear strain
    - `'X0.3'` Bridgestone X0.3, standard compressive stress, up to 400% shear strain
    - `'X0.3-0MPa'` Bridgestone X0.3, zero compressive stress, up to 400% shear strain
2. This material uses SI unit in calculation formula. `ar` and `hr` must be converted into [m^2] and [m], respectively.

3. `rs` and `rf` areavailable if this material is applied to multipleShearSpring (MSS) element. Recommended values are `rs` = $\frac{1}{\sum_{i=0}^{n-1}\sin(\pi*i/n)^2}$ and `rf` = $\frac{1}{\sum_{i=0}^{n-1}\sin(\pi*i/n)}$, where n is the number of springs in the MSS. For example, when n=8, `rs` =0.2500, `rf` =0.1989.

See also:

Notes

KikuchiAikenLRB Material

uniaxialMaterial(*'KikuchiAikenLRB'*, *matTag*, *type*, *ar*, *hr*, *gr*, *ap*, *tp*, *alph*, *beta*[, *'-T'*, *temp*][, *'-coKQ'*, *rk*, *rq*][, *'-coMSS'*, *rs*, *rf*])

This command is used to construct a uniaxial KikuchiAikenLRB material object. This material model produces nonlinear hysteretic curves of lead-rubber bearings.

`matTag` (int)	integer tag identifying material
`type` (int)	rubber type (see note 1)
`ar` (float)	area of rubber [unit: m^2]
`hr` (float)	total thickness of rubber [unit: m]
`gr` (float)	shear modulus of rubber [unit: N/m^2]
`ap` (float)	area of lead plug [unit: m^2]
`tp` (float)	yield stress of lead plug [unit: N/m^2]
`alph` (float)	shear modulus of lead plug [unit: N/m^2]
`beta` (float)	ratio of initial stiffness to yielding stiffness
`temp` (float)	temperature [unit: °C]
`rk` `rq` (float)	reduction rate for yielding stiffness (`rk`) and force at zero displacement (`rq`)
`rs` `rf` (float)	reduction rate for stiffness (`rs`) and force (`rf`) (see note 3)

Note:

1. Following rubber types for `type` are available:
   - `1` lead-rubber bearing, up to 400% shear strain [Kikuchi et al., 2010 & 2012]
2. This material uses SI unit in calculation formula. Input arguments must be converted into [m], [m^2], [N/m^2].
3. `rs` and `rf` are available if this material is applied to multipleShearSpring (MSS) element. Recommended values are `rs` = $\frac{1}{\sum_{i=0}^{n-1}\sin(\pi*i/n)^2}$ and `rf` = $\frac{1}{\sum_{i=0} n-1\sin(\pi*i/n)}$, where n is the number of springs in the MSS. For example, when n=8, `rs` = 0.2500 and `rf` = 0.1989.

See also:

Notes

AxialSp Material

uniaxialMaterial(*'AxialSp'*, *matTag*, *sce*, *fty*, *fcy*[, *bte*, *bty*, *bcy*, *fcr*])

This command is used to construct a uniaxial AxialSp material object. This material model produces axial stress-strain curve of elastomeric bearings.

matTag (int)	integer tag identifying material
sce (float)	compressive modulus
fty fcy (float)	yield stress under tension (fty) and compression (fcy) (see note 1)
bte bty bcy (float)	reduction rate for tensile elastic range (bte), tensile yielding (bty) and compressive yielding (bcy) (see note 1)
fcr (float)	target point stress (see note 1)

Note:

1. Input parameters are required to satisfy followings.

   fcy < 0.0 < fty

   0.0 <= bty < bte <= 1.0

   0.0 <= bcy <= 1.0

   fcy <= fcr <= 0.0

See also:

Notes

AxialSpHD Material

uniaxialMaterial (*'AxialSpHD'*, *matTag*, *sce*, *fty*, *fcy*[, *bte*, *bty*, *bth*, *bcy*, *fcr*, *ath*])

This command is used to construct a uniaxial AxialSpHD material object. This material model produces axial stress-strain curve of elastomeric bearings including hardening behavior.

matTag (int)	integer tag identifying material
sce (float)	compressive modulus
fty1 fcy (float)	yield stress under tension (fty) and compression (fcy) (see note 1)
bte bty bth bcy (float)	reduction rate for tensile elastic range (bte), tensile yielding (bty), tensile hardening (bth) and compressive yielding (bcy) (see note 1)
fcr (float)	target point stress (see note 1)

Note:

1. Input parameters are required to satisfy followings.

   fcy < 0.0 < fty

   0.0 <= bty < bth < bte <= 1.0

   0.0 <= bcy <= 1.0

   fcy <= fcr <= 0.0

   1.0 <= ath

See also:

Notes

Pinching Limit State Material

This command is used to construct a uniaxial material that simulates a pinched load-deformation response and exhibits degradation under cyclic loading. This material works with the RotationShearCurve limit surface that can monitor a key deformation and/or a key force in an associated frame element and trigger a degrading behavior in this material when a limiting value of the deformation and/or force are reached. The material can be used in two modes: 1) direct input mode, where pinching and damage parameters are directly input; and 2) calibrated mode for shear-critical concrete columns, where only key column properties are input for model to fully define pinching and damage parameters.

uniaxialMaterial(*'PinchingLimitStateMaterial'*, *matTag*, *nodeT*, *nodeB*, *driftAxis*, *Kelas*, *crvTyp*, *crvTag*, *YpinchUPN*, *YpinchRPN*, *XpinchRPN*, *YpinchUNP*, *YpinchRNP*, *XpinchRNP*, *dmgStrsLimE*, *dmgDispMax*, *dmgE1*, *dmgE2*, *dmgE3*, *dmgE4*, *dmgELim*, *dmgR1*, *dmgR2*, *dmgR3*, *dmgR4*, *dmgRLim*, *dmgRCyc*, *dmgS1*, *dmgS2*, *dmgS3*, *dmgS4*, *dmgSLim*, *dmgSCyc*)

MODE 1: Direct Input

matTag (int)	integer tag identifying material
nodeT (int)	integer node tag to define the first node at the extreme end of the associated flexural frame member (L3 or D5 in Figure)
nodeB (int)	integer node tag to define the last node at the extreme end of the associated flexural frame member (L2 or D2 in Figure)
driftAxis (int)	Integer to indicate the drift axis in which lateral-strength degradation will occur. This axis should be orthogonal to the axis of measured rotation (see `rotAxis` in Rotation Shear Curve definition) `driftAxis` = 1 – Drift along the x-axis `driftAxis` = 2 – Drift along the y-axis `driftAxis` = 3 – Drift along the z-axis
Kelas (float)	floating point value to define the initial material elastic stiffness (Kelastic); Kelas > 0
crvTyp (int)	integer flag to indicate the type of limit curve associated with this material. `crvTyp` = 0 – No limit curve `crvTyp` = 1 – axial limit curve `crvTyp` = 2 – RotationShearCurve
crvTag (int)	integer tag for the unique limit curve object associated with this material
YpinchUPN (float)	floating point unloading force pinching factor for loading in the negative direction. **Note: This value must be between zero and unity**
YpinchRPN (float)	floating point reloading force pinching factor for loading in the negative direction. **Note: This value must be between negative one and unity**
XpinchRPN (float)	floating point reloading displacement pinching factor for loading in the negative direction. **Note: This value must be between negative one and unity**
YpinchUNP (float)	floating point unloading force pinching factor for loading in the positive direction. **Note: This value must be between zero and unity**
YpinchRNP (float)	floating point reloading force pinching factor for loading in the positive direction. **Note: This value must be between negative one and unity**
XpinchRNP (float)	floating point reloading displacement pinching factor for loading in the positive direction. **Note: This value must be between negative one and unity**
dmgStrsLimE (float)	floating point force limit for elastic stiffness damage (typically defined as the lowest of shear strength or shear at flexrual yielding). This value is used to compute the maximum deformation at flexural yield (δmax Eq. 1) and using the initial elastic stiffness (Kelastic) the monotonic energy (Emono Eq. 1) to yield. Input 1 if this type of damage is not required and set `dmgE1`, `dmgE2`, `dmgE3`, `dmgE4`, and `dmgELim` to zero
dmgDispMax (float)	floating point for ultimate drift at failure (δmax Eq. 1) and is used for strength and stiffness damage. This value is used to compute the monotonic energy at axial failure (Emono Eq. 2) by computing the area under the backbone in the positive loading direction up to δmax. Input 1 if this type of damage is not required and set `dmgR1`, `dmgR2`, `dmgR3`, `dmgR4`, and `dmgRLim` to zero for reloading stiffness damage. Similarly set `dmgS1`, `dmgS2`, `dmgS3`, `dmgS4`, and `dmgSLim` to zero if reloading strength damage is not required
dmgE1 dmgE2 (float)	
dmgE3 dmgE4 (float)	floating point elastic stiffness damage factors $\alpha 1$,$\alpha 2$,$\alpha 3$,$\alpha 4$ shown in Eq. 1
dmgELim (float)	floating point elastic stiffness damage limit Dlim shown in Eq. 1; **Note: This value must be between zero and unity**
dmgR1 dmgR2	
dmgR3 dmgR4 (float)	floating point reloading stiffness damage factors $\alpha 1$,$\alpha 2$,$\alpha 3$,$\alpha 4$ shown in Eq. 1
dmgRLim (float)	floating point reloading stiffness damage limit Dlim shown in Eq. 1; **Note: This value must be between zero and unity**
dmgRCyc (float)	floating point cyclic reloading stiffness damage index; **Note: This value must be between zero and unity**
dmgS1 dmgS2	
dmgS3	floating point backbone strength damage factors $\alpha 1$,$\alpha 2$,$\alpha 3$,$\alpha 4$ shown in Eq. 1

uniaxialMaterial(*'PinchingLimitStateMaterial'*, *matTag*, *dnodeT*, *nodeB*, *driftAxis*, *Kelas*, *crvTyp*, *crvTag*, *eleTag*, *b*, *d*, *h*, *a*, *st*, *As*, *Acc*, *ld*, *db*, *rhot*, *fc*, *fy*, *fyt*)

MODE 2: Calibrated Model for Shear-Critical Concrete Columns

`matTag` (int)	integer tag identifying material
`nodeT` (int)	integer node tag to define the first node at the extreme end of the associated flexural frame member (L3 or D5 in Figure)
`nodeB` (int)	integer node tag to define the last node at the extreme end of the associated flexural frame member (L2 or D2 in Figure)
`driftAxis` (int)	integer to indicate the drift axis in which lateral-strength degradation will occur. This axis should be orthogonal to the axis of measured rotation (see `rotAxis`` in Rotation Shear Curve definition) `driftAxis` = 1 – Drift along the x-axis `driftAxis` = 2 – Drift along the y-axis `driftAxis` = 3 – Drift along the z-axis
`Kelas` (float)	floating point value to define the shear stiffness (Kelastic) of the shear spring prior to shear failure `Kelas` = -4 – Shear stiffness calculated assuming double curvature and shear springs at both column element ends `Kelas` = -3 – Shear stiffness calculated assuming double curvature and a shear spring at one column element end `Kelas` = -2 – Shear stiffness calculated assuming single curvature and shear springs at both column element ends `Kelas` = -1 – Shear stiffness calculated assuming single curvature and a shear spring at one column element end `Kelas` > 0 – Shear stiffness is the input value Note: integer inputs allow the model to know whether column height equals the shear span (cantelever) or twice the shear span (double curvature). For columns in frames, input the value for the case that best approximates column end conditions or manually input shear stiffness (typically double curvature better estimates framed column behavior)
`crvTag` (int)	integer tag for the unique limit curve object associated with this material
`eleTag` (int)	integer element tag to define the associated beam-column element used to extract axial load
`b` (float)	floating point column width (inches)
`d` (float)	floating point column depth (inches)
`h` (float)	floating point column height (inches)
`a` (float)	floating point shear span length (inches)
`st` (float)	floating point transverse reinforcement spacing (inches) along column height
`As` (float)	floating point total area (inches squared) of longitudinal steel bars in section
`Acc` (float)	floating point gross confined concrete area (inches squared) bounded by the transverse reinforcement in column section
`ld` (float)	floating point development length (inches) of longitudinal bars using ACI 318-11 Eq. 12-1 and Eq. 12-2
`db` (float)	floating point diameter (inches) of longitudinal bars in column section
`rhot` (float)	floating point transverse reinforcement ratio (Ast/st.db)
`f'c` (float)	floating point concrete compressive strength (ksi)
`fy` (float)	floating point longitudinal steel yield strength (ksi)
`fyt` (float)	floating point transverse steel yield strength (ksi)

See also:

Notes

CFSWSWP Wood-Sheathed Cold-Formed Steel Shear Wall Panel

uniaxialMaterial('CFSWSWP', *matTag*, *height*, *width*, *fut*, *tf*, *Ife*, *Ifi*, *ts*, *np*, *ds*, *Vs*, *sc*, *nc*, *type*, *openingArea*, *openingLength*)

This command is used to construct a uniaxialMaterial model that simulates the hysteresis response (Shear strength-Lateral displacement) of a wood-sheathed cold-formed steel shear wall panel (CFS-SWP). The hysteresis model has smooth curves and takes into account the strength and stiffness degradation, as well as pinching effect.

This uniaxialMaterial gives results in Newton and Meter units, for strength and displacement, respectively.

`matTag` (int)	integer tag identifying material
`height` (float)	SWP's height (mm)
`width` (float)	SWP's width (mm)
`fuf` (float)	Tensile strength of framing members (MPa)
`tf` (float)	Framing thickness (mm)
`Ife` (float)	Moment of inertia of the double end-stud (mm4)
`Ifi` (float)	Moment of inertia of the intermediate stud (mm4)
`ts` (float)	Sheathing thickness (mm)
`np` (float)	Sheathing number (one or two sides sheathed)
`ds` (float)	Screws diameter (mm)
`Vs` (float)	Screws shear strength (N)
`sc` (float)	Screw spacing on the SWP perimeter (mm)
`nc` (float)	Total number of screws located on the SWP perimeter
`type` (int)	Integer identifier used to define wood sheathing type (DFP=1, OSB=2, CSP=3)
`openingArea` (float)	Total area of openings (mm^2)
`openingLength` (float)	Cumulative length of openings (mm)

See also:

Notes

CFSSSWP Steel-Sheathed Cold-formed Steel Shear Wall Panel

uniaxialMaterial('CFSSSWP', *matTag*, *height*, *width*, *fuf*, *fyf*, *tf*, *Af*, *fus*, *fys*, *ts*, *np*, *ds*, *Vs*, *sc*, *dt*, *openingArea*, *openingLength*)

This command is used to construct a uniaxialMaterial model that simulates the hysteresis response (Shear strength-lateral Displacement) of a Steel-Sheathed Cold-Formed Steel Shear Wall Panel (CFS-SWP). The hysteresis model has smooth curves and takes into account the strength and stiffness degradation, as well as pinching effect.

This uniaxialMaterial gives results in Newton and Meter units, for strength and displacement, respectively.

matTag (int)	integer tag identifying material
height (float)	SWP's height (mm)
width (float)	SWP's width (mm)
fuf (float)	Tensile strength of framing members (MPa)
fyf (float)	Yield strength of framing members (MPa)
tf (float)	Framing thickness (mm)
Af (float)	Framing cross section area (mm^2)
fus (float)	Tensile strength of steel sheet sheathing (MPa)
fys (float)	Yield strength of steel sheet sheathing (MPa)
ts (float)	Sheathing thickness (mm)
np (float)	Sheathing number (one or two sides sheathed)
ds (float)	Screws diameter (mm)
Vs (float)	Screws shear strength (N)
sc (float)	Screw spacing on the SWP perimeter (mm)
dt (float)	Anchor bolt's diameter (mm)
openingArea (float)	Total area of openings (mm^2)
openingLength (float)	Cumulative length of openings (mm)

See also:

Notes

1.4.14 nDMaterial commands

nDMaterial (*matType*, *matTag*, **matArgs*)

This command is used to construct an NDMaterial object which represents the stress-strain relationship at the gauss-point of a continuum element.

matType (str)	material type
matTag (int)	material tag.
matArgs (list)	a list of material arguments, must be preceded with *.

For example,

```
matType = 'ElasticIsotropic'
matTag = 1
matArgs = [E, v]
nDMaterial(matType, matTag, *matArgs)
```

The following contain information about available matType:

ElasticIsotropic

nDMaterial (*'ElasticIsotropic'*, *matTag*, *E*, *v*, *rho=0.0*)

This command is used to construct an ElasticIsotropic material object.

matTag (int)	integer tag identifying material
E (float)	elastic modulus
v (float)	Poisson's ratio
rho (float)	mass density (optional)

The material formulations for the ElasticIsotropic object are:

- 'ThreeDimensional'
- 'PlaneStrain'
- 'Plane Stress'
- 'AxiSymmetric'
- 'PlateFiber'

ElasticOrthotropic

nDMaterial (*'ElasticOrthotropic'*, *matTag*, *Ex*, *Ey*, *Ez*, *vxy*, *vyz*, *vzx*, *Gxy*, *Gyz*, *Gzx*, *rho=0.0*)
This command is used to construct an ElasticOrthotropic material object.

`matTag` (int)	integer tag identifying material
`Ex` (float)	elastic modulus in x direction
`Ey` (float)	elastic modulus in y direction
`Ez` (float)	elastic modulus in z direction
`vxy` (float)	Poisson's ratios in x and y plane
`vyz` (float)	Poisson's ratios in y and z plane
`vzx` (float)	Poisson's ratios in z and x plane
`Gxy` (float)	shear modulii in x and y plane
`Gyz` (float)	shear modulii in y and z plane
`Gzx` (float)	shear modulii in z and x plane
`rho` (float)	mass density (optional)

The material formulations for the ElasticOrthotropic object are:

- 'ThreeDimensional'
- 'PlaneStrain'
- 'Plane Stress'
- 'AxiSymmetric'
- 'BeamFiber'
- 'PlateFiber'

J2Plasticity

nDMaterial (*'J2Plasticity'*, *matTag*, *K*, *G*, *sig0*, *sigInf*, *delta*, *H*)
This command is used to construct an multi dimensional material object that has a von Mises (J2) yield criterium and isotropic hardening.

`matTag` (int)	integer tag identifying material
`K` (float)	bulk modulus
`G` (float)	shear modulus
`sig0` (float)	initial yield stress
`sigInf` (float)	final saturation yield stress
`delta` (float)	exponential hardening parameter
`H` (float)	linear hardening parameter

The material formulations for the J2Plasticity object are:

- 'ThreeDimensional'
- 'PlaneStrain'
- 'Plane Stress'
- 'AxiSymmetric'
- 'PlateFiber'

J2 isotropic hardening material class

Elastic Model

$$\sigma = K * trace(\epsilon_e) + (2 * G) * dev(\epsilon_e)$$

Yield Function

$$\phi(\sigma, q) = ||dev(\sigma)|| - \sqrt(\tfrac{2}{3} * q(x_i))$$

Saturation Isotropic Hardening with linear term

$$q(x_i) = \sigma_0 + (\sigma_\infty - \sigma_0) * exp(-delta * \xi) + H * \xi$$

Flow Rules

$$\dot{\epsilon_p} = \gamma * \frac{\partial \phi}{\partial \sigma}$$
$$\dot{\xi} = -\gamma * \frac{\partial \phi}{\partial q}$$

Linear Viscosity

$$\gamma = \frac{\phi}{\eta}(if \phi > 0)$$

Backward Euler Integration Routine Yield condition enforced at time n+1

set η = 0 for rate independent case

DrukerPrager

nDMaterial (*'DrukerPrager'*, *matTag*, *K*, *G*, *sigmaY*, *rho*, *rhoBar*, *Kinf*, *Ko*, *delta1*, *delta2*, *H*, *theta*, *density*, *atmPressure=101e3*)

This command is used to construct an multi dimensional material object that has a Drucker-Prager yield criterium.

`matTag` (int)	integer tag identifying material
`K` (float)	bulk modulus
`G` (float)	shear modulus
`sigmaY` (float)	yield stress
`rho` (float)	frictional strength parameter
`rhoBar` (float)	controls evolution of plastic volume change, $0 \leq rhoBar \leq rho$.
`Kinf` (float)	nonlinear isotropic strain hardening parameter, $Kinf \geq 0$.
`Ko` (float)	nonlinear isotropic strain hardening parameter, $Ko \geq 0$.
`delta1` (float)	nonlinear isotropic strain hardening parameter, $delta1 \geq 0$.
`delta2` (float)	tension softening parameter, $delta2 \geq 0$.
`H` (float)	linear hardening parameter, $H \geq 0$.
`theta` (float)	controls relative proportions of isotropic and kinematic hardening, $0 \leq theta \leq 1$.
`density` (float)	mass density of the material
`atmPressure` (float)	optional atmospheric pressure for update of elastic bulk and shear moduli

The material formulations for the DrukerPrager object are:

- 'ThreeDimensional'
- 'PlaneStrain'

See theory.

Damage2p

nDMaterial (*'Damage2p'*, *matTag*, *fcc*, *'-fct'*, *fct*, *'-E'*, *E*, *'-ni'*, *ni*, *'-Gt'*, *Gt*, *'-Gc'*, *Gc*, *'-rho_bar'*, *rho_bar*, *'-H'*, *H*, *'-theta'*, *theta*, *'-tangent'*, *tangent*)

This command is used to construct a three-dimensional material object that has a Drucker-Prager plasticity model coupled with a two-parameter damage model.

matTag (int)	integer tag identifying material
fcc (float)	concrete compressive strength, negative real value (positive input is changed in sign automatically)
fct (float)	optional concrete tensile strength, positive real value (for concrete like materials is less than fcc), $0.1 * abs(fcc)$ = $4750 * sqrt(abs(fcc))\ if\ abs(fcc) < 2000$ because fcc is assumed in MPa (see ACI 318)
E (float)	optional Young modulus, $57000 * sqrt(abs(fcc))$ if $abs(fcc) > 2000$ because fcc is assumed in psi (see ACI 318)
ni (float)	optional Poisson coefficient, 0.15 (from comparison with tests by Kupfer Hilsdorf Rusch 1969)
Gt (float)	optional tension fracture energy density, positive real value (integral of the stress-strain envelope in tension), $1840 * fct * fct/E$ (from comparison with tests by Gopalaratnam and Shah 1985)
Gc (float)	optional compression fracture energy density, positive real value (integral of the stress-strain envelope after the peak in compression), :math:6250*fcc*fcc/E‘ (from comparison with tests by Karsan and Jirsa 1969)
rho_bar (float)	optional parameter of plastic volume change, positive real value $0 = rhoBar < sqrt(2/3)$, 0.2 (from comparison with tests by Kupfer Hilsdorf Rusch 1969)
H (float)	optional linear hardening parameter for plasticity, positive real value (usually less than E), $0.25 * E$ (from comparison with tests by Karsan and Jirsa 1969 and Gopalaratnam and Shah 1985)
theta (float)	optional ratio between isotropic and kinematic hardening, positive real value $0 = theta = 1$ (with: 0 hardening kinematic only and 1 hardening isotropic only, 0.5 (from comparison with tests by Karsan and Jirsa 1969 and Gopalaratnam and Shah 1985)
tangent (float)	optional integer to choose the computational stiffness matrix, 0: computational tangent; 1: damaged secant stiffness (hint: in case of strong nonlinearities use it with Krylov-Newton algorithm)

The material formulations for the Damage2p object are:

- 'ThreeDimensional'
- 'PlaneStrain'
- 'Plane Stress'
- 'AxiSymmetric'
- 'PlateFiber'

See also here

PlaneStress

nDMaterial('PlaneStress', *matTag*, *threeDtag*)

This command is used to construct a plane-stress material wrapper which converts any three-dimensional material into a plane stress material via static condensation.

`matTag` (int)	integer tag identifying material
`threeDtag` (int)	tag of perviously defined 3d ndMaterial material

The material formulations for the PlaneStress object are:

- `'Plane Stress'`

PlaneStrain

nDMaterial('PlaneStrain', *matTag*, *threeDtag*)

This command is used to construct a plane-stress material wrapper which converts any three-dimensional material into a plane strain material by imposing plain strain conditions on the three-dimensional material.

`matTag` (int)	integer tag identifying material
`threeDtag` (int)	integer tag of previously defined 3d ndMaterial material

The material formulations for the PlaneStrain object are:

- `'PlaneStrain'`

MultiaxialCyclicPlasticity

nDMaterial('MultiaxialCyclicPlasticity', *matTag*, *rho*, *K*, *G*, *Su*, *Ho*, *h*, *m*, *beta*, *KCoeff*)

This command is used to construct an multiaxial Cyclic Plasticity model for clays

`matTag` (int)	integer tag identifying material
`rho` (float)	density
`K` (float)	buck modulus
`G` (float)	maximum (small strain) shear modulus
`Su` (float)	undrained shear strength, size of bounding surface $R = \sqrt{8/3} * Su$
`Ho` (float)	linear kinematic hardening modulus of bounding surface
`h` (float)	hardening parameter
`m` (float)	hardening parameter
`beta` (float)	integration parameter, usually beta=0.5
`KCoeff` (float)	coefficient of earth pressure, K0

BoundingCamClay

nDMaterial('BoundingCamClay', *matTag*, *massDensity*, *C*, *bulkMod*, *OCR*, *mu_o*, *alpha*, *lambda*, *h*, *m*)

This command is used to construct a multi-dimensional bounding surface Cam Clay material object after Borja et al. (2001).

`matTag` (int)	integer tag identifying material
`massDensity` (float)	mass density
`C` (float)	ellipsoidal axis ratio (defines shape of ellipsoidal loading/bounding surfaces)
`bulkMod` (float)	initial bulk modulus
`OCR` (float)	overconsolidation ratio
`mu_o` (float)	initial shear modulus
`alpha` (float)	pressure-dependency parameter for modulii (greater than or equal to zero)
`lambda` (float)	soil compressibility index for virgin loading
`h` (float)	hardening parameter for plastic response inside of bounding surface (if h = 0, no hardening)
`m` (float)	hardening parameter (exponent) for plastic response inside of bounding surface (if m = 0, only linear hardening)

The material formulations for the BoundingCamClay object are:

- `'ThreeDimensional'`
- `'PlaneStrain'`

See also for information

PlateFiber

nDMaterial(*'PlateFiber'*, *matTag*, *threeDTag*)

This command is used to construct a plate-fiber material wrapper which converts any three-dimensional material into a plate fiber material (by static condensation) appropriate for shell analysis.

`matTag` (int)	integer tag identifying material
`threeDTag` (float)	material tag for a previously-defined three-dimensional material

FSAM

nDMaterial(*'FSAM'*, *matTag*, *rho*, *sX*, *sY*, *conc*, *rouX*, *rouY*, *nu*, *alfadow*)

This command is used to construct a nDMaterial FSAM (Fixed-Strut-Angle-Model, Figure 1, Kolozvari et al., 2015), which is a plane-stress constitutive model for simulating the behavior of RC panel elements under generalized, in-plane, reversed-cyclic loading conditions (Ulugtekin, 2010; Orakcal et al., 2012). In the FSAM constitutive model, the strain fields acting on concrete and reinforcing steel components of a RC panel are assumed to be equal to each other, implying perfect bond assumption between concrete and reinforcing steel bars. While the reinforcing steel bars develop uniaxial stresses under strains in their longitudinal direction, the behavior of concrete is defined using stress–strain relationships in biaxial directions, the orientation of which is governed by the state of cracking in concrete. Although the concrete stress–strain relationship used in the FSAM is fundamentally uniaxial in nature, it also incorporates biaxial softening effects including compression softening and biaxial damage. For transfer of shear stresses across the cracks, a friction-based elasto-plastic shear aggregate interlock model is adopted, together with a linear elastic model for representing dowel action on the reinforcing steel bars (Kolozvari, 2013). Note that FSAM constitutive model is implemented to be used with Shear-Flexure Interaction model for RC walls (SFI_MVLEM), but it could be also used elsewhere.

`matTag` (int)	integer tag identifying material
`rho` (float)	Material density
`sX` (float)	Tag of uniaxialMaterial simulating horizontal (x) reinforcement
`sY` (float)	Tag of uniaxialMaterial simulating vertical (y) reinforcement
`conc` (float)	Tag of uniaxialMaterial simulating concrete, shall be used with uniaxialMaterial ConcreteCM
`rouX` (float)	Reinforcing ratio in horizontal (x) direction ($rouX =_{s,x} / A_{gross,x}$)
`rouY` (float)	Reinforcing ratio in vertical (x) direction ($rouY =_{s,y} / A_{gross,y}$)
`nu` (float)	Concrete friction coefficient ($0.0 < \nu < 1.5$)
`alfadow` (float)	Stiffness coefficient of reinforcement dowel action ($0.0 < alfadow < 0.05$)

See also here

References:

1. Kolozvari K., Orakcal K., and Wallace J. W. (2015). "Shear-Flexure Interaction Modeling of reinforced Concrete Structural Walls and Columns under Reversed Cyclic Loading", Pacific Earthquake Engineering Research Center, University of California, Berkeley, PEER Report No. 2015/12
2. Kolozvari K. (2013). "Analytical Modeling of Cyclic Shear-Flexure Interaction in Reinforced Concrete Structural Walls", PhD Dissertation, University of California, Los Angeles.
3. Orakcal K., Massone L.M., and Ulugtekin D. (2012). "Constitutive Modeling of Reinforced Concrete Panel Behavior under Cyclic Loading", Proceedings, 15th World Conference on Earthquake Engineering, Lisbon, Portugal.
4. Ulugtekin D. (2010). "Analytical Modeling of Reinforced Concrete Panel Elements under Reversed Cyclic Loadings", M.S. Thesis, Bogazici University, Istanbul, Turkey.

ManzariDafalias

nDMaterial(*'ManzariDafalias'*, *matTag*, *G0*, *nu*, *e_init*, *Mc*, *c*, *lambda_c*, *e0*, *ksi*, *P_atm*, *m*, *h0*, *ch*, *nb*, *A0*, *nd*, *z_max*, *cz*, *Den*)

This command is used to construct a multi-dimensional Manzari-Dafalias(2004) material.

`matTag` (int)	integer tag identifying material
`G0` (float)	shear modulus constant
`nu` (float)	poisson ratio
`e_init` (float)	initial void ratio
`Mc` (float)	critical state stress ratio
`c` (float)	ratio of critical state stress ratio in extension and compression
`lambda_c` (float)	critical state line constant
`e0` (float)	critical void ratio at p = 0
`ksi` (float)	critical state line constant
`P_atm` (float)	atmospheric pressure
`m` (float)	yield surface constant (radius of yield surface in stress ratio space)
`h0` (float)	constant parameter
`ch` (float)	constant parameter
`nb` (float)	bounding surface parameter, $nb \geq 0$
`A0` (float)	dilatancy parameter
`nd` (float)	dilatancy surface parameter $nd \geq 0$
`z_max` (float)	fabric-dilatancy tensor parameter
`cz` (float)	fabric-dilatancy tensor parameter
`Den` (float)	mass density of the material

The material formulations for the ManzariDafalias object are:

- `'ThreeDimensional'`
- `'PlaneStrain'`

See also here

References

Dafalias YF, Manzari MT. "Simple plasticity sand model accounting for fabric change effects". Journal of Engineering Mechanics 2004

PM4Sand

`nDMaterial`(*'PM4Sand', matTag, Dr, G0, hpo, Den, patm, h0, emax, emin, nb, nd, Ado, zmax, cz, ce, phic, nu, cgd, cdr, ckaf, Q, R, m, Fsed_min, p_sedo*)

This command is used to construct a 2-dimensional PM4Sand material.

matTag (int)	integer tag identifying material
Dr (float)	Relative density, in fraction
G0 (float)	Shear modulus constant
hpo (float)	Contraction rate parameter
Den (float)	Mass density of the material
P_atm (float)	Optional, Atmospheric pressure
h0 (float)	Optional, Variable that adjusts the ratio of plastic modulus to elastic modulus
emax (float)	Optional, Maximum and minimum void ratios
emin (float)	Optional, Maximum and minimum void ratios
nb (float)	Optional, Bounding surface parameter, $nb \geq 0$
nd (float)	Optional, Dilatancy surface parameter $nd \geq 0$
Ado (float)	Optional, Dilatancy parameter, will be computed at the time of initialization if input value is negative
z_max (float)	Optional, Fabric-dilatancy tensor parameter
cz (float)	Optional, Fabric-dilatancy tensor parameter
ce (float)	Optional, Variable that adjusts the rate of strain accumulation in cyclic loading
phic (float)	Optional, Critical state effective friction angle
nu (float)	Optional, Poisson's ratio
cgd (float)	Optional, Variable that adjusts degradation of elastic modulus with accumulation of fabric
cdr (float)	Optional, Variable that controls the rotated dilatancy surface
ckaf (float)	Optional, Variable that controls the effect that sustained static shear stresses have on plastic modulus
Q (float)	Optional, Critical state line parameter
R (float)	Optional, Critical state line parameter
m (float)	Optional, Yield surface constant (radius of yield surface in stress ratio space)
Fsed_min (float)	Optional, Variable that controls the minimum value the reduction factor of the elastic moduli can get during reconsolidation
p_sedo (float)	Optional, Mean effective stress up to which reconsolidation strains are enhanced

The material formulations for the PM4Sand object are:

- `'PlaneStrain'`

See als here

References

R.W.Boulanger, K.Ziotopoulou. "PM4Sand(Version 3.1): A Sand Plasticity Model for Earthquake Engineering Applications". Report No. UCD/CGM-17/01 2017

StressDensityModel

nDMaterial (*'StressDensityModel', matTag, mDen, eNot, A, n, nu, a1, b1, a2, b2, a3, b3, fd, muNot, muCyc, sc, M, patm, ssl1, ssl2, ssl3, ssl4, ssl5, ssl6, ssl7, ssl8, ssl9, ssl10, hsl, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10*)

This command is used to construct a multi-dimensional stress density material object for modeling sand behaviour following the work of Cubrinovski and Ishihara (1998a,b).

matTag (int)	integer tag identifying material

Continued on next page

Table 1 – continued from previous page

mDen (float)	mass density
eNot (float)	initial void ratio
A (float)	constant for elastic shear modulus
n (float)	pressure dependency exponent for elastic shear modulus
nu (float)	Poisson's ratio
a1 (float)	peak stress ratio coefficient ($etaMax = a1 + b1 * Is$)
b1 (float)	peak stress ratio coefficient ($etaMax = a1 + b1 * Is$)
a2 (float)	max shear modulus coefficient ($Gn_max = a2 + b2 * Is$)
b2 (float)	max shear modulus coefficient ($Gn_max = a2 + b2 * Is$)
a3 (float)	min shear modulus coefficient ($Gn_min = a3 + b3 * Is$)
b3 (float)	min shear modulus coefficient ($Gn_min = a3 + b3 * Is$)
fd (float)	degradation constant
muNot (float)	dilatancy coefficient (monotonic loading)
muCyc (float)	dilatancy coefficient (cyclic loading)
sc (float)	dilatancy strain
M (float)	critical state stress ratio
patm (float)	atmospheric pressure (in appropriate units)
ssl1 (float)	void ratio of quasi steady state (QSS-line) at pressure p1 (default = 0.877)
ssl2 (float)	void ratio of quasi steady state (QSS-line) at pressure p2 (default = 0.877)
ssl3 (float)	void ratio of quasi steady state (QSS-line) at pressure p3 (default = 0.873)
ssl4 (float)	void ratio of quasi steady state (QSS-line) at pressure p4 (default = 0.870)
ssl5 (float)	void ratio of quasi steady state (QSS-line) at pressure p5 (default = 0.860)
ssl6 (float)	void ratio of quasi steady state (QSS-line) at pressure p6 (default = 0.850)
ssl7 (float)	void ratio of quasi steady state (QSS-line) at pressure p7 (default = 0.833)
ssl8 (float)	void ratio of quasi steady state (QSS-line) at pressure p8 (default = 0.833)
ssl9 (float)	void ratio of quasi steady state (QSS-line) at pressure p9 (default = 0.833)
ssl10 (float)	void ratio of quasi steady state (QSS-line) at pressure p10 (default = 0.833)
hsl (float)	void ratio of upper reference state (UR-line) for all pressures (default = 0.895)
p1 (float)	pressure corresponding to ssl1 (default = 1.0 kPa)
p2 (float)	pressure corresponding to ssl1 (default = 10.0 kPa)
p3 (float)	pressure corresponding to ssl1 (default = 30.0 kPa)
p4 (float)	pressure corresponding to ssl1 (default = 50.0 kPa)
p5 (float)	pressure corresponding to ssl1 (default = 100.0 kPa)
p6 (float)	pressure corresponding to ssl1 (default = 200.0 kPa)
p7 (float)	pressure corresponding to ssl1 (default = 400.0 kPa)
p8 (float)	pressure corresponding to ssl1 (default = 400.0 kPa)
p9 (float)	pressure corresponding to ssl1 (default = 400.0 kPa)
p10 (float)	pressure corresponding to ssl1 (default = 400.0 kPa)

The material formulations for the StressDensityModel object are:

- `'ThreeDimensional'`
- `'PlaneStrain'`

References

Cubrinovski, M. and Ishihara K. (1998a) 'Modelling of sand behaviour based on state concept,' Soils and Foundations, 38(3), 115-127.

Cubrinovski, M. and Ishihara K. (1998b) 'State concept and modified elastoplasticity for sand modelling,' Soils and Foundations, 38(4), 213-225.

Das, S. (2014) Three Dimensional Formulation for the Stress-Strain-Dilatancy Elasto-Plastic Constitutive Model for Sand Under Cyclic Behaviour, Master's Thesis, University of Canterbury.

AcousticMedium

nDMaterial (*'AcousticMedium'*, *matTag*, *K*, *rho*)

This command is used to construct an acoustic medium NDMaterial object.

`matTag` (int)	integer tag identifying material
`K` (float)	bulk module of the acoustic medium
`rho` (float)	mass density of the acoustic medium

CycLiqCP

nDMaterial (*'CycLiqCP'*, *matTag*, *G0*, *kappa*, *h*, *Mfc*, *dre1*, *Mdc*, *dre2*, *rdr*, *alpha*, *dir*, *ein*, *rho*)

This command is used to construct a multi-dimensional material object that that follows the constitutive behavior of a cyclic elastoplasticity model for large post- liquefaction deformation.

CycLiqCP material is a cyclic elastoplasticity model for large post-liquefaction deformation, and is implemented using a cutting plane algorithm. The model is capable of reproducing small to large deformation in the pre- to post-liquefaction regime. The elastic moduli of the model are pressure dependent. The plasticity in the model is developed within the framework of bounding surface plasticity, with special consideration to the formulation of reversible and irreversible dilatancy.

The model does not take into consideration of the state of sand, and requires different parameters for sand under different densities and confining pressures. The surfaces (i.e. failure and maximum pre-stress) are considered as circles in the pi plane.

The model has been validated against VELACS centrifuge model tests and has used on numerous simulations of liquefaction related problems.

When this material is employed in regular solid elements (e.g., FourNodeQuad, Brick), it simulates drained soil response. When solid-fluid coupled elements (u-p elements and SSP u-p elements) are used, the model is able to simulate undrained and partially drained behavior of soil.

`matTag` (int)	integer tag identifying material
`G0` (float)	A constant related to elastic shear modulus
`kappa` (float)	bulk modulus
`h` (float)	Model parameter for plastic modulus
`Mfc` (float)	Stress ratio at failure in triaxial compression
`dre1` (float)	Coefficient for reversible dilatancy generation
`Mdc` (float)	Stress ratio at which the reversible dilatancy sign changes
`dre2` (float)	Coefficient for reversible dilatancy release
`rdr` (float)	Reference shear strain length
`alpha` (float)	Parameter controlling the decrease rate of irreversible dilatancy
`dir` (float)	Coefficient for irreversible dilatancy potential
`ein` (float)	Initial void ratio
`rho` (float)	Saturated mass density

The material formulations for the CycLiqCP object are:

- `'ThreeDimensional'`
- `'PlaneStrain'`

See also here

CycLiqCPSP

nDMaterial ('CycLiqCPSP', *matTag*, *G0*, *kappa*, *h*, *M*, *dre1*, *dre2*, *rdr*, *alpha*, *dir*, *lambdac*, *ksi*, *e0*, *np*, *nd*, *ein*, *rho*)

This command is used to construct a multi-dimensional material object that that follows the constitutive behavior of a cyclic elastoplasticity model for large post- liquefaction deformation.

CycLiqCPSP material is a constitutive model for sand with special considerations for cyclic behaviour and accumulation of large post-liquefaction shear deformation, and is implemented using a cutting plane algorithm. The model: (1) achieves the simulation of post-liquefaction shear deformation based on its physics, allowing the unified description of pre- and post-liquefaction behavior of sand; (2) directly links the cyclic mobility of sand with reversible and irreversible dilatancy, enabling the unified description of monotonic and cyclic loading; (3) introduces critical state soil mechanics concepts to achieve unified modelling of sand under different states.

The critical, maximum stress ratio and reversible dilatancy surfaces follow a rounded triangle in the pi plane similar to the Matsuoka-Nakai criterion.

When this material is employed in regular solid elements (e.g., FourNodeQuad, Brick), it simulates drained soil response. When solid-fluid coupled elements (u-p elements and SSP u-p elements) are used, the model is able to simulate undrained and partially drained behavior of soil.

`matTag` (int)	integer tag identifying material
`G0` (float)	A constant related to elastic shear modulus
`kappa` (float)	bulk modulus
`h` (float)	Model parameter for plastic modulus
`M` (float)	Critical state stress ratio
`dre1` (float)	Coefficient for reversible dilatancy generation
`dre2` (float)	Coefficient for reversible dilatancy release
`rdr` (float)	Reference shear strain length
`alpha` (float)	Parameter controlling the decrease rate of irreversible dilatancy
`dir` (float)	Coefficient for irreversible dilatancy potential
`lambdac` (float)	Critical state constant
`ksi` (float)	Critical state constant
`e0` (float)	Void ratio at pc=0
`np` (float)	Material constant for peak mobilized stress ratio
`nd` (float)	Material constant for reversible dilatancy generation stress ratio
`ein` (float)	Initial void ratio
`rho` (float)	Saturated mass density

The material formulations for the CycLiqCP object are:

- `'ThreeDimensional'`
- `'PlaneStrain'`

See also here

REFERENCES: Wang R., Zhang J.M., Wang G., 2014. A unified plasticity model for large post-liquefaction shear deformation of sand. Computers and Geotechnics. 59, 54-66.

PlaneStressUserMaterial

nDMaterial ('PlaneStressUserMaterial', *matTag*, *fc*, *ft*, *fcu*, *epsc0*, *epscu*, *epstu*, *stc*)

This command is used to create the multi-dimensional concrete material model that is based on the damage mechanism and smeared crack model.

`matTag` (int)	integer tag identifying material
`fc` (float)	concrete compressive strength at 28 days (positive)
`ft` (float)	concrete tensile strength (positive)
`fcu` (float)	concrete crushing strength (negative)
`epsc0` (float)	concrete strain at maximum strength (negative)
`epscu` (float)	concrete strain at crushing strength (negative)
`epstu` (float)	ultimate tensile strain (positive)
`stc` (float)	shear retention factor

PlateFromPlaneStress

nDMaterial('*PlateFromPlaneStress*', *matTag*, *newmatTag*, *matTag*, *OutofPlaneModulus*)

This command is used to create the multi-dimensional concrete material model that is based on the damage mechanism and smeared crack model.

`matTag` (int)	integer tag identifying material
`newmatTag` (int)	new integer tag identifying material deriving from pre-defined PlaneStressUserMaterial
`matTag` (int)	integer tag identifying PlaneStressUserMaterial
`OutofPlaneModulus` (float)	shear modulus of out plane

PlateRebar

nDMaterial('*PlateRebar*', *matTag*, *newmatTag*, *matTag*, *sita*)

This command is used to create the multi-dimensional reinforcement material.

`matTag` (int)	integer tag identifying material
`newmatTag` (int)	new integer tag identifying material deriving from pre-defined uniaxial steel material
`matTag` (int)	integer tag identifying uniaxial steel material
`sita` (float)	define the angle of steel layer, 90 (longitudinal steel), 0 (tranverse steel)

ContactMaterial2D

nDMaterial('*ContactMaterial2D*', *matTag*, *mu*, *G*, *c*, *t*)

This command is used to construct a ContactMaterial2D nDMaterial object.

`matTag` (int)	integer tag identifying material
`mu` (float)	interface frictional coefficient
`G` (float)	interface stiffness parameter
`c` (float)	interface cohesive intercept
`t` (float)	interface tensile strength

The ContactMaterial2D nDMaterial defines the constitutive behavior of a frictional interface between two bodies in contact. The interface defined by this material object allows for sticking, frictional slip, and separation between the two bodies in a two-dimensional analysis. A regularized Coulomb frictional law is assumed. Information on the theory behind this material can be found in, e.g. Wriggers (2002).

Note:

1. The ContactMaterial2D nDMaterial has been written to work with the SimpleContact2D and BeamContact2D element objects.
2. There are no valid recorder queries for this material other than those which are listed with those elements

References:

Wriggers, P. (2002). Computational Contact Mechanics. John Wilely & Sons, Ltd, West Sussex, England.

ContactMaterial3D

nDMaterial (*'ContactMaterial3D'*, *matTag*, *mu*, *G*, *c*, *t*)

This command is used to construct a ContactMaterial3D nDMaterial object.

`matTag` (int)	integer tag identifying material
`mu` (float)	interface frictional coefficient
`G` (float)	interface stiffness parameter
`c` (float)	interface cohesive intercept
`t` (float)	interface tensile strength

The ContactMaterial3D nDMaterial defines the constitutive behavior of a frictional interface between two bodies in contact. The interface defined by this material object allows for sticking, frictional slip, and separation between the two bodies in a three-dimensional analysis. A regularized Coulomb frictional law is assumed. Information on the theory behind this material can be found in, e.g. Wriggers (2002).

Note:

1. The ContactMaterial3D nDMaterial has been written to work with the SimpleContact3D and BeamContact3D element objects.
2. There are no valid recorder queries for this material other than those which are listed with those elements.

References:

Wriggers, P. (2002). Computational Contact Mechanics. John Wilely & Sons, Ltd, West Sussex, England.

InitialStateAnalysisWrapper

nDMaterial (*'InitialStateAnalysisWrapper'*, *matTag*, *nDMatTag*, *nDim*)

The InitialStateAnalysisWrapper nDMaterial allows for the use of the InitialStateAnalysis command for setting initial conditions. The InitialStateAnalysisWrapper can be used with any nDMaterial. This material wrapper allows for the development of an initial stress field while maintaining the original geometry of the problem. An example analysis is provided below to demonstrate the use of this material wrapper object.

`matTag` (int)	integer tag identifying material
`nDMatTag` (int)	the tag of the associated nDMaterial object
`nDim` (int)	number of dimensions (2 for 2D, 3 for 3D)

Note:

1. There are no valid recorder queries for the InitialStateAnalysisWrapper.
2. The InitialStateAnalysis off command removes all previously defined recorders. Two sets of recorders are needed if the results before and after this command are desired. See the example below for more.
3. The InitialStateAnalysisWrapper material is somewhat tricky to use in dynamic analysis. Sometimes setting the displacement to zero appears to be interpreted as an initial displacement in subsequent steps, resulting in undesirable vibrations.

PressureIndependMultiYield

nDMaterial (*'PressureIndependMultiYield'*, *matTag*, *nd*, *rho*, *refShearModul*, *refBulkModul*, *cohesi*, *peakShearStra*, *frictionAng=0.*, *refPress=100.*, *pressDependCoe=0.*, *noYieldSurf=20*, **yieldSurf*)

PressureIndependMultiYield material is an elastic-plastic material in which plasticity exhibits only in the deviatoric stress-strain response. The volumetric stress-strain response is linear-elastic and is independent of the deviatoric response. This material is implemented to simulate monotonic or cyclic response of materials whose shear behavior is insensitive to the confinement change. Such materials include, for example, organic soils or clay under fast (undrained) loading conditions.

`matTag` (int)	integer tag identifying material
`nd` (float)	Number of dimensions, 2 for plane-strain, and 3 for 3D analysis.
`rho` (float)	Saturated soil mass density.
`refShearModul` (float)	(G_r) Reference low-strain shear modulus, specified at a reference mean effective confining pressure refPress of p'r (see below).
`refBulkModul` (float)	(B_r) Reference bulk modulus, specified at a reference mean effective confining pressure refPress of p'r (see below).
`cohesi` (float)	(c) Apparent cohesion at zero effective confinement.
`peakShearStra` (float)	(γ_{max}) An octahedral shear strain at which the maximum shear strength is reached, specified at a reference mean effective confining pressure refPress of p'r (see below).
`frictionAng` (float)	(phi) Friction angle at peak shear strength in degrees, optional (default is 0.0).
`refPress` (float)	(p'_r) Reference mean effective confining pressure at which G_r, B_r, and γ_{max} are defined, optional (default is 100. kPa).
`pressDependCoe` (float)	(d) A positive constant defining variations of G and B as a function of instantaneous effective confinement p' (default is 0.0) $G = G_r(\frac{p'}{p'_r})^d$ $B = B_r(\frac{p'}{p'_r})^d$ If $\phi = 0$, d is reset to 0.0.
`noYieldSurf` (float)	Number of yield surfaces, optional (must be less than 40, default is 20). The surfaces are generated based on the hyperbolic relation defined in Note 2 below.
`yieldSurf` (list (float))	Instead of automatic surfaces generation (Note 2), you can define yield surfaces directly based on desired shear modulus reduction curve. To do so, add a minus sign in front of noYieldSurf, then provide noYieldSurf pairs of shear strain (r) and modulus ratio (Gs) values. For example, to define 10 surfaces: yieldSurf = [r1, Gs1, . . . , r10, Gs10]

See also notes

PressureDependMultiYield

nDMaterial(*'PressureDependMultiYield', matTag, nd, rho, refShearModul, refBulkModul, frictionAng, peakShearStra, refPress, pressDependCoe, PTAng, contrac, *dilat, *liquefac, noYieldSurf=20.0, *yieldSurf=[], e=0.6, *params=[0.9, 0.02, 0.7, 101.0], c=0.3*)

PressureDependMultiYield material is an elastic-plastic material for simulating the essential response characteristics of pressure sensitive soil materials under general loading conditions. Such characteristics include dilatancy (shear-induced volume contraction or dilation) and non-flow liquefaction (cyclic mobility), typically exhibited in sands or silts during monotonic or cyclic loading.

`matTag` (int)	integer tag identifying material
`nd` (float)	Number of dimensions, 2 for plane-strain, and 3 for 3D analysis.
`rho` (float)	Saturated soil mass density.
`refShearModul` (float)	(G_r) Reference low-strain shear modulus, specified at a reference mean effective confining pressure refPress of p'r (see below).
`refBulkModul` (float)	(B_r) Reference bulk modulus, specified at a reference mean effective confining pressure refPress of p'r (see below).
`frictionAng` (float)	(phi) Friction angle at peak shear strength in degrees, optional (default is 0.0).
`peakShearStra` (float)	(γ_{max}) An octahedral shear strain at which the maximum shear strength is reached, specified at a reference mean effective confining pressure refPress of p'r (see below).
`refPress` (float)	(p'_r) Reference mean effective confining pressure at which G_r, B_r, and γ_{max} are defined, optional (default is 100. kPa).
`pressDependCoe` (float)	(d) A positive constant defining variations of G and B as a function of instantaneous effective confinement p' (default is 0.0) $G = G_r(\frac{p'}{p'_r})^d$ $B = B_r(\frac{p'}{p'_r})^d$ If $\phi = 0$, d is reset to 0.0.
`PTAng` (float)	(ϕ_{PT}) Phase transformation angle, in degrees.
`contrac` (float)	A non-negative constant defining the rate of shear-induced volume decrease (contraction) or pore pressure buildup. A larger value corresponds to faster contraction rate.
`dilat` (list (float))	Non-negative constants defining the rate of shear-induced volume increase (dilation). Larger values correspond to stronger dilation rate. `dilat = [dilat1, dilat2]`.
`liquefac` (list (float))	Parameters controlling the mechanism of liquefaction-induced perfectly plastic shear strain accumulation, i.e., cyclic mobility. Set `liquefac[0] = 0` to deactivate this mechanism altogether. `liquefac[0]` defines the effective confining pressure (e.g., 10 kPa in SI units or 1.45 psi in English units) below which the mechanism is in effect. Smaller values should be assigned to denser sands. `Liquefac[1]` defines the maximum amount of perfectly plastic shear strain developed at zero effective confinement during each loading phase. Smaller values should be assigned to denser sands. `Liquefac[2]` defines the maximum amount of biased perfectly plastic shear strain γ_b accumulated at each loading phase under biased shear loading conditions, as $\gamma_b = liquefac[1] \times liquefac[2]$. Typically, $liquefac[2]$ takes a value between 0.0 and 3.0. Smaller values should be assigned to denser sands. See the references listed at the end of this chapter for more information.
`noYieldSurf` (float)	Number of yield surfaces, optional (must be less than 40, default is 20). The surfaces are generated based on the hyperbolic relation defined in Note 2 below.
`yieldSurf` (list (float))	Instead of automatic surfaces generation (Note 2), you can define yield surfaces directly based on desired shear modulus reduction curve. To do so, add a minus sign in front of noYieldSurf, then provide noYieldSurf pairs of shear strain (r) and modulus ratio (Gs) values. For example, to define 10 surfaces: yieldSurf = [r1, Gs1, . . . , r10, Gs10]
`e` (float)	Initial void ratio, optional (default is 0.6).
`params` (list (float))	`params=[cs1, cs2, cs3, pa]` defining a straight critical-state line ec in e-p' space. If cs3=0, ec = cs1-cs2 log(p'/pa) else (Li and Wang, JGGE, 124(12)), ec = cs1-cs2(p'/pa)cs3 where pa is atmospheric pressure for normalization (typically 101 kPa in SI units, or 14.65 psi in English units). All four constants are optional
`c` (float)	Numerical constant (default value = 0.3 kPa)

See also notes

PressureDependMultiYield02

nDMaterial (*'PressureDependMultiYield02', matTag, nd, rho, refShearModul, refBulkModul, frictionAng, peakShearStra, refPress, pressDependCoe, PTAng, contrac[0], contrac[2], dilat[0], dilat[2], noYieldSurf=20.0, *yieldSurf=[], contrac[1]=5.0, dilat[1]=3.0, *liquefac=[1.0,0.0],e=0.6, *params=[0.9, 0.02, 0.7, 101.0], c=0.1*)

PressureDependMultiYield02 material is modified from PressureDependMultiYield material, with:

1. additional parameters (`contrac[2]` and `dilat[2]`) to account for K_σ effect,
2. a parameter to account for the influence of previous dilation history on subsequent contraction phase (`contrac[1]`), and
3. modified logic related to permanent shear strain accumulation (`liquefac[0]` and `liquefac[1]`).

`matTag` (int)	integer tag identifying material
`contrac[2]` (float)	A non-negative constant reflecting K_σ effect.
`dilat[2]` (float)	A non-negative constant reflecting K_σ effect.
`contrac[1]` (float)	A non-negative constant reflecting dilation history on contraction tendency.
`liquefac[0]` (float)	Damage parameter to define accumulated permanent shear strain as a function of dilation history. (Redefined and different from PressureDependMultiYield material).
`liquefac[1]` (float)	Damage parameter to define biased accumulation of permanent shear strain as a function of load reversal history. (Redefined and different from PressureDependMultiYield material).
`c` (float)	Numerical constant (default value = 0.1 kPa)

See also notes

FluidSolidPorousMaterial

nDMaterial (*'FluidSolidPorousMaterial', matTag, nd, soilMatTag, combinedBulkModul, pa=101.0*)

FluidSolidPorousMaterial couples the responses of two phases: fluid and solid. The fluid phase response is only volumetric and linear elastic. The solid phase can be any NDMaterial. This material is developed to simulate the response of saturated porous media under fully undrained condition.

`matTag` (int)	integer tag identifying material
`nd` (float)	Number of dimensions, 2 for plane-strain, and 3 for 3D analysis.
`soilMatTag` (int)	The material number for the solid phase material (previously defined).
`combinBulkModul` (float)	Combined undrained bulk modulus B_c relating changes in pore pressure and volumetric strain, may be approximated by: $B_c \approx B_f/n$ where B_f is the bulk modulus of fluid phase (2.2x106 kPa (or 3.191x105 psi) for water), and n the initial porosity.
`pa` (float)	Optional atmospheric pressure for normalization (typically 101 kPa in SI units, or 14.65 psi in English units)

See also notes

1.4.15 section commands

section (*secType*, *secTag*, **secArgs*)

This command is used to construct a SectionForceDeformation object, hereto referred to as Section, which represents force-deformation (or resultant stress-strain) relationships at beam-column and plate sample points.

`secType` (str)	section type
`secTag` (int)	section tag.
`secArgs` (list)	a list of section arguments, must be preceded with `*`.

For example,

```
secType = 'Elastic'
secTag = 1
secArgs = [E, A, Iz]
section(secType, secTag, *secArgs)
```

The following contain information about available `secType`:

Elastic Section

section (*'Elastic'*, *secTag*, *E*, *A*, *Iz*, *G=0.0*, *alphaY=0.0*)

section (*'Elastic'*, *secTag*, *E*, *A*, *Iz*, *Iy*, *G*, *J*, *alphaY=0.0*, *alphaZ=0.0*)

This command allows the user to construct an ElasticSection. The inclusion of shear deformations is optional.

`secTag` (int)	unique section tag
`E` (float)	Young's Modulus
`A` (float)	cross-sectional area of section
`Iz` (float)	second moment of area about the local z-axis
`Iy` (float)	second moment of area about the local y-axis (required for 3D analysis)
`G` (float)	Shear Modulus (optional for 2D analysis, required for 3D analysis)
`J` (float)	torsional moment of inertia of section (required for 3D analysis)
`alphaY` (float)	shear shape factor along the local y-axis (optional)
`alphaZ` (float)	shear shape factor along the local z-axis (optional)

Note: The elastic section can be used in the nonlinear beam column elements, which is useful in the initial stages of developing a complex model.

Fiber Section

section (*'Fiber'*, *secTag*, *'-GJ'*, *GJ=0.0*)

This commnand allows the user to construct a FiberSection object. Each FiberSection object is composed of Fibers, with each fiber containing a UniaxialMaterial, an area and a location (y,z).

`secTag` (int)	unique section tag
`GJ` (float)	linear-elastic torsional stiffness assigned to the section (optional)

section (*'FiberThermal'*, *secTag*, *'-GJ'*, *GJ=0.0*)
This command create a FiberSectionThermal object.

Note:

1. The commands below should be called after the section command to generate all the fibers in the section.
2. The patch and layer commands can be used to generate multiple fibers in a single command.

Fiber Command

fiber (*yloc*, *zloc*, *A*, *matTag*)
This command allows the user to construct a single fiber and add it to the enclosing FiberSection or NDFiberSection.

`yloc` (float)	y coordinate of the fiber in the section (local coordinate system)
`zloc` (float)	z coordinate of the fiber in the section (local coordinate system)
`A` (float)	cross-sectional area of fiber
`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).

Patch Command

patch (*type*, **args*)
The patch command is used to generate a number of fibers over a cross-sectional area. Currently there are three types of cross-section that fibers can be generated: quadrilateral, rectangular and circular.

patch (*'quad'*, *matTag*, *numSubdivIJ*, *numSubdivJK*, **crdsI*, **crdsJ*, **crdsK*, **crdsL*)
This is the command to generate a quadrilateral shaped patch (the geometry of the patch is defined by four vertices: I J K L. The coordinates of each of the four vertices is specified in COUNTER CLOCKWISE sequence)

`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).
`numSubdivIJ` (int)	number of subdivisions (fibers) in the IJ direction.
`numSubdivJK` (int)	number of subdivisions (fibers) in the JK direction.
`crdsI` (list (float))	y & z-coordinates of vertex I (local coordinate system)
`crdsJ` (list (float))	y & z-coordinates of vertex J (local coordinate system)
`crdsK` (list (float))	y & z-coordinates of vertex K (local coordinate system)
`crdsL` (list (float))	y & z-coordinates of vertex L (local coordinate system)

patch (*'rect'*, *matTag*, *numSubdivY*, *numSubdivZ*, **crdsI*, **crdsJ*)
This is the command to generate a rectangular patch. The geometry of the patch is defined by coordinates of

vertices: I and J. The first vertex, I, is the bottom-left point and the second vertex, J, is the top-right point, having as a reference the local y-z plane.

`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).
`numSubdivY` (int)	number of subdivisions (fibers) in local y direction.
`numSubdivZ` (int)	number of subdivisions (fibers) in local z direction.
`crdsI` (list (float))	y & z-coordinates of vertex I (local coordinate system)
`crdsJ` (list (float))	y & z-coordinates of vertex J (local coordinate system)

patch(*'circ'*, *matTag*, *numSubdivCirc*, *numSubdivRad*, **center*, **rad*, **ang*)

This is the command to generate a circular shaped patch

`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).
`numSubdivCirc` (int)	number of subdivisions (fibers) in the circumferential direction (number of wedges)
`numSubdivRad` (int)	number of subdivisions (fibers) in the radial direction (number of rings)
`center` (list (float))	y & z-coordinates of the center of the circle
`rad` (list (float))	internal & external radius
`ang` (list (float))	starting & ending-coordinates angles (degrees)

Layer Command

layer(*type*, **args*)

The layer command is used to generate a number of fibers along a line or a circular arc.

layer(*'straight'*, *matTag*, *numFiber*, *areaFiber*, **start*, **end*)

This command is used to construct a straight line of fibers

`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).
`numFiber` (int)	number of fibers along line
`areaFiber` (float)	area of each fiber
`start` (list (float))	y & z-coordinates of first fiber in line (local coordinate system)
`end` (list (float))	y & z-coordinates of last fiber in line (local coordinate system)

layer(*'circ'*, *matTag,numFiber,areaFiber,*center,radius,*ang=[0.0,360.0-360/numFiber]*)

This command is used to construct a line of fibers along a circular arc

`matTag` (int)	material tag associated with this fiber (UniaxialMaterial tag for a FiberSection and NDMaterial tag for use in an NDFiberSection).
`numFiber` (int)	number of fibers along line
`areaFiber` (float)	area of each fiber
`center` (list (float))	y & z-coordinates of center of circular arc
`radius` (float)	radius of circlular arc
`ang` (list (float))	starting and ending angle (optional)

NDFiber Section

section(*'NDFiber'*, *secTag*)

This commnand allows the user to construct an NDFiberSection object. Each NDFiberSection object is composed of NDFibers, with each fiber containing an NDMaterial, an area and a location (y,z). The NDFiberSection works for 2D and 3D frame elements and it queries the NDMaterial of each fiber for its axial and shear stresses. In 2D, stress components 11 and 12 are obtained from each fiber in order to provide stress resultants for axial force, bending moment, and shear (N, Mz, and Vy). Stress components 11, 12, and 13 lead to all six stress resultants in 3D (N, Mz, Vy, My, Vz, and T).

The NDFiberSection works with any NDMaterial via wrapper classes that perform static condensation of the stress vector down to the 11, 12, and 13 components, or via concrete NDMaterial subclasses that implement the appropriate fiber stress conditions.

`secTag` (int)	unique section tag

Note:

1. The commands below should be called after the section command to generate all the fibers in the section.
2. The patch and layer commands can be used to generate multiple fibers in a single command.

1. `fiber()`
2. `patch()`
3. `layer()`

Wide Flange Section

section(*'WFSection2d'*, *secTag*, *matTag*, *d*, *tw*, *bf*, *tf*, *Nfw*, *Nff*)

This command allows the user to construct a WFSection2d object, which is an encapsulated fiber representation of a wide flange steel section appropriate for plane frame analysis.

`secTag` (int)	unique section tag
`matTag` (int)	tag of uniaxialMaterial assigned to each fiber
`d` (float)	section depth
`tw` (float)	web thickness
`bf` (float)	flange width
`tf` (float)	flange thickness
`Nfw` (float)	number of fibers in the web
`Nff` (float)	number of fibers in each flange

Note: The section dimensions `d`, `tw`, `bf`, and `tf` can be found in the AISC steel manual.

RC Section

section (*'RCSection2d'*, *secTag*, *coreTag*, *coverTag*, *steelTag*, *d*, *b*, *cover*, *Atop*, *Abot*, *Aside*, *Nfcore*, *Nfcover*, *Nfs*)

This command allows the user to construct an RCSection2d object, which is an encapsulated fiber representation of a rectangular reinforced concrete section with core and confined regions of concrete and single top and bottom layers of reinforcement appropriate for plane frame analysis.

`secTag` (int)	unique section tag
`coreTag` (int)	tag of uniaxialMaterial assigned to each fiber in the core region
`coverTag` (int)	tag of uniaxialMaterial assigned to each fiber in the cover region
`steelTag` (int)	tag of uniaxialMaterial assigned to each reinforcing bar
`d` (float)	section depth
`b` (float)	section width
`cover` (float)	cover depth (assumed uniform around perimeter)
`Atop` (float)	area of reinforcing bars in top layer
`Abot` (float)	area of reinforcing bars in bottom layer
`Aside` (float)	area of reinforcing bars on intermediate layers
`Nfcore` (float)	number of fibers through the core depth
`Nfcover` (float)	number of fibers through the cover depth
`Nfs` (float)	number of bars on the top and bottom rows of reinforcement (Nfs-2 bars will be placed on the side rows)

Note: For more general reinforced concrete section definitions, use the Fiber Section command.

Parallel Section

section (*'Parallel'*, *secTag*, **tags*)

Connect sections in parallel.

`secTag` (int)	unique section tag
`tags` (list (int))	tags of of predefined sections.

Section Aggregator

section (*'Aggregator'*, *secTag*, **mats*, *'-section'*, *sectionTag*)

This command is used to construct a SectionAggregator object which aggregates groups previously-defined UniaxialMaterial objects into a single section force-deformation model. Each UniaxialMaterial object represents the section force-deformation response for a particular section degree-of-freedom (dof). There is no interaction between responses in different dof directions. The aggregation can include one previously defined section.

`secTag` (int)	unique section tag
`mats` (list)	list of tags and dofs of previously-defined UniaxialMaterial objects, `mats = [matTag1,dof1, matTag2,dof2,...]` the force-deformation quantity to be modeled by this section object. One of the following section dof may be used: • `'P'` Axial force-deformation • `'Mz'` Moment-curvature about section local z-axis • `'Vy'` Shear force-deformation along section local y-axis • `'My'` Moment-curvature about section local y-axis • `'Vz'` Shear force-deformation along section local z-axis • `'T'` Torsion Force-Deformation
`sectionTag` (int)	tag of previously-defined Section object to which the UniaxialMaterial objects are aggregated as additional force-deformation relationships (optional)

Uniaxial Section

section (*'Uniaxial'*, *secTag*, *matTag*, *quantity*)

This command is used to construct a UniaxialSection object which uses a previously-defined UniaxialMaterial object to represent a single section force-deformation response quantity.

`secTag` (int)	unique section tag
`matTag` (int)	tag of uniaxial material
`quantity` (str)	the force-deformation quantity to be modeled by this section object. One of the following section dof may be used: • `'P'` Axial force-deformation • `'Mz'` Moment-curvature about section local z-axis • `'Vy'` Shear force-deformation along section local y-axis • `'My'` Moment-curvature about section local y-axis • `'Vz'` Shear force-deformation along section local z-axis • `'T'` Torsion Force-Deformation

Elastic Membrane Plate Section

section (*'ElasticMembranePlateSection'*, *secTag*, *E*, *nu*, *h*, *rho*)

This command allows the user to construct an ElasticMembranePlateSection object, which is an isotropic section appropriate for plate and shell analysis.

`secTag` (int)	unique section tag
`E` (float)	Young's Modulus
`nu` (float)	Poisson's Ratio
`h` (float)	depth of section
`rho` (float)	mass density

Plate Fiber Section

section (*'PlateFiber'*, *secTag*, *matTag*, *h*)

This command allows the user to construct a MembranePlateFiberSection object, which is a section that numerically integrates through the plate thickness with "fibers" and is appropriate for plate and shell analysis.

`secTag` (int)	unique section tag
`matTag` (int)	nDMaterial tag to be assigned to each fiber
`h` (float)	plate thickness

Bidirectional Section

section (*'Bidirectional'*, *secTag*, *E*, *Fy*, *Hiso*, *Hkin*, *code1='Vy'*, *code2='P'*)

This command allows the user to construct a Bidirectional section, which is a stress-resultant plasticity model of two coupled forces. The yield surface is circular and there is combined isotropic and kinematic hardening.

`secTag` (int)	unique section tag
`E` (float)	elastic modulus
`Fy` (float)	yield force
`Hiso` (float)	isotropic hardening modulus
`Hkin` (float)	kinematic hardening modulus
`code1` (str)	section force code for direction 1 (optional)
`code2` (str)	section force code for direction 2 (optional) One of the following section code may be used: • `'P'` Axial force-deformation • `'Mz'` Moment-curvature about section local z-axis • `'Vy'` Shear force-deformation along section local y-axis • `'My'` Moment-curvature about section local y-axis • `'Vz'` Shear force-deformation along section local z-axis • `'T'` Torsion Force-Deformation

Isolator2spring Section

section (*'Iso2spring'*, *matTag*, *tol*, *k1*, *Fyo*, *k2o*, *kvo*, *hb*, *PE*, *Po=0.0*)

This command is used to construct an Isolator2spring section object, which represents the buckling behavior of an elastomeric bearing for two-dimensional analysis in the lateral and vertical plane. An Isolator2spring section represents the resultant force-deformation behavior of the bearing, and should be used with a zeroLengthSection element. The bearing should be constrained against rotation.

`secTag` (int)	unique section tag
`tol` (float)	tolerance for convergence of the element state. Suggested value: E-12 to E-10. OpenSees will warn if convergence is not achieved, however this usually does not prevent global convergence.
`k1` (float)	initial stiffness for lateral force-deformation
`Fyo` (float)	nominal yield strength for lateral force-deformation
`k2o` (float)	nominal postyield stiffness for lateral force-deformation
`kvo` (float)	nominal stiffness in the vertical direction
`hb` (float)	total height of elastomeric bearing
`PE` (float)	Euler Buckling load for the bearing
`Po` (float)	axial load at which nominal yield strength is achieved (optional)

LayeredShell

nDMaterial (*'LayeredShell'*, *sectionTag*, *nLayers *mats*)

This command will create the section of the multi-layer shell element, including the multi-dimensional concrete, reinforcement material and the corresponding thickness.

`sectionTag` (int)	unique tag among sections
`nLayers` (int)	total numbers of layers
`mats` (list)	a list of material tags and thickenss, `[[mat1,thk1], ..., [mat2,thk2]]`

1.4.16 frictionModel commands

frictionModel (*frnType*, *frnTag*, **frnArgs*)

The frictionModel command is used to construct a friction model object, which specifies the behavior of the coefficient of friction in terms of the absolute sliding velocity and the pressure on the contact area. The command has at least one argument, the friction model type.

`frnType` (str)	frictionModel type
`frnTag` (int)	frictionModel tag.
`frnArgs` (list)	a list of frictionModel arguments, must be preceded with `*`.

For example,

```
frnType = 'Coulomb'
frnTag = 1
frnArgs = [mu]
frictionModel(frnType, frnTag, *frnArgs)
```

The following contain information about available `frnType`:

Coulomb

frictionModel (*'Coulomb'*, *frnTag*, *mu*)

This command is used to construct a Coulomb friction model object. Coulomb's Law of Friction states that kinetic friction is independent of the sliding velocity.

`frnTag` (int)	unique friction model tag
`mu` (float)	coefficient of friction

Velocity Dependent Friction

frictionModel (*'VelDependent'*, *frnTag*, *muSlow*, *muFast*, *transRate*)

This command is used to construct a VelDependent friction model object. It is useful for modeling the behavior of PTFE or PTFE-like materials sliding on a stainless steel surface. For a detailed presentation on the velocity dependence of such interfaces please refer to Constantinou et al. (1999).

`frnTag` (int)	unique friction model tag
`muSlow` (float)	coefficient of friction at low velocity
`muFast` (float)	coefficient of friction at high velocity
`transRate` (float)	transition rate from low to high velocity

$$\mu = \mu_{fast} - (\mu_{fast} - \mu_{slow}) \cdot e^{-transRate \cdot |v|}$$

REFERENCE:

Constantinou, M.C., Tsopelas, P., Kasalanati, A., and Wolff, E.D. (1999). "Property modification factors for seismic isolation bearings". Report MCEER-99-0012, Multidisciplinary Center for Earthquake Engineering Research, State University of New York.

Velocity and Normal Force Dependent Friction

frictionModel (*'VelNormalFrcDep'*, *frnTag*, *aSlow*, *nSlow*, *aFast*, *nFast*, *alpha0*, *alpha1*, *alpha2*, *maxMuFact*)

This command is used to construct a VelNormalFrcDep friction model object.

`frnTag` (int)	unique friction model tag
`aSlow` (float)	constant for coefficient of friction at low velocity
`nSlow` (float)	exponent for coefficient of friction at low velocity
`aFast` (float)	constant for coefficient of friction at high velocity
`nFast` (float)	exponent for coefficient of friction at high velocity
`alpha0` (float)	constant rate parameter coefficient
`alpha1` (float)	linear rate parameter coefficient
`alpha2` (float)	quadratic rate parameter coefficient
`maxMuFact` (float)	factor for determining the maximum coefficient of friction. This value prevents the friction coefficient from exceeding an unrealistic maximum value when the normal force becomes very small. The maximum friction coefficient is determined from μFast, for example $\mu \leq maxMuFac*Fast$.

Velocity and Pressure Dependent Friction

frictionModel (*'VelPressureDep'*, *frnTag*, *muSlow*, *muFast0*, *A*, *deltaMu*, *alpha*, *transRate*)

This command is used to construct a VelPressureDep friction model object.

`frnTag` (int)	unique friction model tag
`muSlow` (float)	coefficient of friction at low velocity
`muFast0` (float)	initial coefficient of friction at high velocity
`A` (float)	nominal contact area
`deltaMu` (float)	pressure parameter calibrated from experimental data
`alpha` (float)	pressure parameter calibrated from experimental data
`transRate` (float)	transition rate from low to high velocity

Multi-Linear Velocity Dependent Friction

frictionModel (*'VelDepMultiLinear'*, *frnTag*, *'-vel'*, **velocityPoints*, *'-frn'*, **frictionPoints*)

This command is used to construct a VelDepMultiLinear friction model object. The friction-velocity relationship is given by a multi-linear curve that is define by a set of points. The slope given by the last two specified points on the positive velocity axis is extrapolated to infinite positive velocities. Velocity and friction points need to be equal or larger than zero (no negative values should be defined). The number of provided velocity points needs to be equal to the number of provided friction points.

`frnTag` (int)	unique friction model tag
`velocityPoints` (list (float))	list of velocity points along friction-velocity curve
`frictionPoints` (list (float))	list of friction points along friction-velocity curve

1.4.17 geomTransf commands

geomTransf (*transfType*, *transfTag*, **transfArgs*)

The geometric-transformation command is used to construct a coordinate-transformation (CrdTransf) object, which transforms beam element stiffness and resisting force from the basic system to the global-coordinate system. The command has at least one argument, the transformation type.

`transfType` (str)	geomTransf type
`transfTag` (int)	geomTransf tag.
`transfArgs` (list)	a list of geomTransf arguments, must be preceded with `*`.

For example,

```
transfType = 'Linear'
transfTag = 1
transfArgs = []
geomTransf(transfType, transfTag, *transfArgs)
```

The following contain information about available `transfType`:

Linear Transformation

geomTransf (*'Linear'*, *transfTag*, *'-jntOffset'*, **dI*, **dJ*)

geomTransf (*'Linear'*, *transfTag*, **vecxz*, *'-jntOffset'*, **dI*, **dJ*)

This command is used to construct a linear coordinate transformation (LinearCrdTransf) object, which performs a linear geometric transformation of beam stiffness and resisting force from the basic system to the global-coordinate system.

`transfTag` (int)	integer tag identifying transformation
`vecxz` (list (float))	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. The local y-axis is defined by taking the cross product of the vecxz vector and the x-axis. These components are specified in the global-coordinate system X,Y,Z and define a vector that is in a plane parallel to the x-z plane of the local-coordinate system. These items need to be specified for the three-dimensional problem.
`dI` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node i (the number of arguments depends on the dimensions of the current model).
`dJ` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node j (the number of arguments depends on the dimensions of the current model).

PDelta Transformation

geomTransf (*'PDelta'*, *transfTag*, *'-jntOffset'*, **dI*, **dJ*)

geomTransf (*'PDelta'*, *transfTag*, **vecxz*, *'-jntOffset'*, **dI*, **dJ*)

This command is used to construct the P-Delta Coordinate Transformation (PDeltaCrdTransf) object, which performs a linear geometric transformation of beam stiffness and resisting force from the basic system to the global coordinate system, considering second-order P-Delta effects.

`transfTag` (int)	integer tag identifying transformation
`vecxz` (list (float))	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. The local y-axis is defined by taking the cross product of the vecxz vector and the x-axis. These components are specified in the global-coordinate system X,Y,Z and define a vector that is in a plane parallel to the x-z plane of the local-coordinate system. These items need to be specified for the three-dimensional problem.
`dI` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node i (the number of arguments depends on the dimensions of the current model).
`dJ` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node j (the number of arguments depends on the dimensions of the current model).

Note: P LARGE Delta effects do not include P small delta effects.

Corotational Transformation

geomTransf (*'Corotational'*, *transfTag*, *'-jntOffset'*, **dI*, **dJ*)

geomTransf (*'Corotational'*, *transfTag*, **vecxz*)

This command is used to construct the Corotational Coordinate Transformation (CorotCrdTransf) object. Corotational transformation can be used in large displacement-small strain problems.

`transfTag` (int)	integer tag identifying transformation
`vecxz` (list (float))	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. The local y-axis is defined by taking the cross product of the vecxz vector and the x-axis. These components are specified in the global-coordinate system X,Y,Z and define a vector that is in a plane parallel to the x-z plane of the local-coordinate system. These items need to be specified for the three-dimensional problem.
`dI` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node i (the number of arguments depends on the dimensions of the current model).
`dJ` (list (float))	joint offset values – offsets specified with respect to the global coordinate system for element-end node j (the number of arguments depends on the dimensions of the current model).

Note: Currently the transformation does not deal with element loads and will ignore any that are applied to the element.

1.5 Analysis Commands

In OpenSees, an analysis is an object which is composed by the aggregation of component objects. It is the component objects which define the type of analysis that is performed on the model. The component classes, as shown in the figure below, consist of the following:

1. ConstraintHandler – determines how the constraint equations are enforced in the analysis – how it handles the boundary conditions/imposed displacements
2. DOF_Numberer – determines the mapping between equation numbers and degrees-of-freedom
3. Integrator – determines the predictive step for time t+dt
4. SolutionAlgorithm – determines the sequence of steps taken to solve the non-linear equation at the current time step
5. SystemOfEqn/Solver – within the solution algorithm, it specifies how to store and solve the system of equations in the analysis
6. Convergence Test – determines when convergence has been achieved.

1.5.1 constraints commands

constraints(*constraintType*, **constraintArgs*)

This command is used to construct the ConstraintHandler object. The ConstraintHandler object determines how the constraint equations are enforced in the analysis. Constraint equations enforce a specified value for a DOF, or a relationship between DOFs.

`constraintType` (str)	constraints type
`constraintArgs` (list)	a list of constraints arguments

The following contain information about available `constraintType`:

Plain Constraints

constraints(*'Plain'*)

This command is used to construct a Plain constraint handler. A plain constraint handler can only enforce homogeneous single point constraints (fix command) and multi-point constraints constructed where the constraint matrix is equal to the identity (equalDOF command). The following is the command to construct a plain constraint handler:

Note: As mentioned, this constraint handler can only enforce homogeneous single point constraints (fix command) and multi-pont constraints where the constraint matrix is equal to the identity (equalDOF command).

Lagrange Multipliers

constraints(*'Lagrange'*, *alphaS=1.0*, *alphaM=1.0*)

This command is used to construct a LagrangeMultiplier constraint handler, which enforces the constraints by introducing Lagrange multiplies to the system of equation. The following is the command to construct a plain constraint handler:

alphaS (float)	α_S factor on single points.
alphaM (float)	α_M factor on multi-points.

Note: The Lagrange multiplier method introduces new unknowns to the system of equations. The diagonal part of the system corresponding to these new unknowns is 0.0. This ensure that the system IS NOT symmetric positive definite.

Penalty Method

constraints(*'Penalty'*, *alphaS=1.0*, *alphaM=1.0*)

This command is used to construct a Penalty constraint handler, which enforces the constraints using the penalty method. The following is the command to construct a penalty constraint handler:

alphaS (float)	α_S factor on single points.
alphaM (float)	α_M factor on multi-points.

Note: The degree to which the constraints are enforced is dependent on the penalty values chosen. Problems can arise if these values are too small (constraint not enforced strongly enough) or too large (problems associated with conditioning of the system of equations).

Transformation Method

constraints(*'Transformation'*)

This command is used to construct a transformation constraint handler, which enforces the constraints using the transformation method. The following is the command to construct a transformation constraint handler

Note:

- The single-point constraints when using the transformation method are done directly. The matrix equation is not manipulated to enforce them, rather the trial displacements are set directly at the nodes at the start of each analysis step.
- Great care must be taken when multiple constraints are being enforced as the transformation method does not follow constraints:
  1. If a node is fixed, constrain it with the fix command and not equalDOF or other type of constraint.
  2. If multiple nodes are constrained, make sure that the retained node is not constrained in any other constraint.

And remember if a node is constrained to multiple nodes in your model it probably means you have messed up.

1.5.2 numberer commands

numberer(*numbererType*, **numbererArgs*)

This command is used to construct the DOF_Numberer object. The DOF_Numberer object determines the mapping between equation numbers and degrees-of-freedom – how degrees-of-freedom are numbered.

`numbererType` (str)	numberer type
`numbererArgs` (list)	a list of numberer arguments

The following contain information about available `numbererType`:

Plain Numberer

numberer(*'Plain'*)

This command is used to construct a Plain degree-of-freedom numbering object to provide the mapping between the degrees-of-freedom at the nodes and the equation numbers. A Plain numberer just takes whatever order the domain gives it nodes and numbers them, this ordering is both dependent on node numbering and size of the model.

Note: For very small problems and for the sparse matrix solvers which provide their own numbering scheme, order is not really important so plain numberer is just fine. For large models and analysis using solver types other than the sparse solvers, the order will have a major impact on performance of the solver and the plain handler is a poor choice.

RCM Numberer

numberer(*'RCM'*)

This command is used to construct an RCM degree-of-freedom numbering object to provide the mapping between the degrees-of-freedom at the nodes and the equation numbers. An RCM numberer uses the reverse Cuthill-McKee scheme to order the matrix equations.

AMD Numberer

numberer(*'AMD'*)

This command is used to construct an AMD degree-of-freedom numbering object to provide the mapping between the degrees-of-freedom at the nodes and the equation numbers. An AMD numberer uses the approximate minimum degree scheme to order the matrix equations.

1.5.3 system commands

system(*systemType*, **systemArgs*)

This command is used to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.

`systemType` (str)	system type
`systemArgs` (list)	a list of system arguments

The following contain information about available `systemType`:

BandGeneral SOE

system(*'BandGen'*)

This command is used to construct a BandGeneralSOE linear system of equation object. As the name implies, this class is used for matrix systems which have a banded profile. The matrix is stored as shown below in a

1dimensional array of size equal to the bandwidth times the number of unknowns. When a solution is required, the Lapack routines DGBSV and SGBTRS are used.

BandSPD SOE

system(*'BandSPD'*)

This command is used to construct a BandSPDSOE linear system of equation object. As the name implies, this class is used for symmetric positive definite matrix systems which have a banded profile. The matrix is stored as shown below in a 1 dimensional array of size equal to the (bandwidth/2) times the number of unknowns. When a solution is required, the Lapack routines DPBSV and DPBTRS are used.

ProfileSPD SOE

system(*'ProfileSPD'*)

This command is used to construct a profileSPDSOE linear system of equation object. As the name implies, this class is used for symmetric positive definite matrix systems. The matrix is stored as shown below in a 1 dimensional array with only those values below the first non-zero row in any column being stored. This is sometimes also referred to as a skyline storage scheme.

SuperLU SOE

system(*'SuperLU'*)

This command is used to construct a SparseGEN linear system of equation object. As the name implies, this class is used for sparse matrix systems. The solution of the sparse matrix is carried out using SuperLU.

UmfPack SOE

system(*'UmfPack'*)

This command is used to construct a sparse system of equations which uses the UmfPack solver.

FullGeneral SOE

system(*'FullGeneral'*)

This command is used to construct a Full General linear system of equation object. As the name implies, the class utilizes NO space saving techniques to cut down on the amount of memory used. If the matrix is of size, nxn, then storage for an nxn array is sought from memory when the program runs. When a solution is required, the Lapack routines DGESV and DGETRS are used.

Note: This type of system should almost never be used! This is because it requires a lot more memory than every other solver and takes more time in the actal solving operation than any other solver. It is required if the user is interested in looking at the global system matrix.

SparseSYM SOE

system(*'SparseSYM'*)

This command is used to construct a sparse symmetric system of equations which uses a row-oriented solution method in the solution phase.

- *PFEM SOE*

1.5.4 test commands

test (*testType*, **testArgs*)

This command is used to construct the LinearSOE and LinearSolver objects to store and solve the test of equations in the analysis.

`testType` (str)	test type
`testArgs` (list)	a list of test arguments

The following contain information about available `testType`:

NormUnbalance

test (*'NormUnbalance'*, *tol*, *iter*, *pFlag=0*, *nType=2*, *maxincr=-1*)

Create a NormUnbalance test, which uses the norm of the right hand side of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)
`maxincr` (int)	Maximum times of error increasing. (optional)

When using the Penalty method additional large forces to enforce the penalty functions exist on the right hand side, making convergence using this test usually impossible (even though solution might have converged).

NormDispIncr

test (*'NormDispIncr'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a NormUnbalance test, which uses the norm of the left hand side solution vector of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

When using the Lagrange method to enforce the constraints, the Lagrange multipliers appear in the solution vector.

energyIncr

test (*'EnergyIncr'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a EnergyIncr test, which uses the dot product of the solution vector and norm of the right hand side of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

- When using the Penalty method additional large forces to enforce the penalty functions exist on the right hand side, making convergence using this test usually impossible (even though solution might have converged).
- When using the Lagrange method to enforce the constraints, the Lagrange multipliers appear in the solution vector.

RelativeNormUnbalance

test (*'RelativeNormUnbalance'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a RelativeNormUnbalance test, which uses the relative norm of the right hand side of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

- When using the Penalty method additional large forces to enforce the penalty functions exist on the right hand side, making convergence using this test usually impossible (even though solution might have converged).

RelativeNormDispIncr

test (*'RelativeNormDispIncr'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a RelativeNormDispIncr test, which uses the relative of the solution vector of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

RelativeTotalNormDispIncr

test (*'relativeTotalNormDispIncr'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a RelativeTotalNormDispIncr test, which uses the ratio of the current norm to the total norm (the sum of all the norms since last convergence) of the solution vector.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

RelativeEnergyIncr

test (*'RelativeEnergyIncr'*, *tol*, *iter*, *pFlag=0*, *nType=2*)

Create a RelativeEnergyIncr test, which uses the relative dot product of the solution vector and norm of the right hand side of the matrix equation to determine if convergence has been reached.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

FixedNumIter

test (*'FixedNumIter'*, *iter*, *pFlag=0*, *nType=2*)

Create a FixedNumIter test, that performs a fixed number of iterations without testing for convergence.

`tol` (float)	Tolerance criteria used to check for convergence.
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

NormDispAndUnbalance

`test` (*'NormDispAndUnbalance'*, *tolIncr*, *tolR*, *iter*, *pFlag=0*, *nType=2*, *maxincr=-1*)

Create a NormDispAndUnbalance test, which check if both `'NormUnbalance'` and `'NormDispIncr'` are converged.

`tolIncr` (float)	Tolerance for left hand solution increments
`tolIncr` (float)	Tolerance for right hand residual
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)
`maxincr` (int)	Maximum times of error increasing. (optional)

NormDispOrUnbalance

`test` (*'NormDispOrUnbalance'*, *tolIncr*, *tolR*, *iter*, *pFlag=0*, *nType=2*, *maxincr=-1*)

Create a NormDispOrUnbalance test, which check if both `'NormUnbalance'` and `'normDispIncr'` are converged.

`tolIncr` (float)	Tolerance for left hand solution increments
`tolIncr` (float)	Tolerance for right hand residual
`iter` (int)	Max number of iterations to check
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)
`maxincr` (int)	Maximum times of error increasing. (optional)

- *PFEM test*

1.5.5 algorithm commands

algorithm(*algoType*, **algoArgs*)

This command is used to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.

`algoType` (str)	algorithm type
`algoArgs` (list)	a list of algorithm arguments

The following contain information about available `algoType`:

Linear Algorithm

algorithm(*'Linear'*, *secant=False*, *initial=False*, *factorOnce=False*)

Create a Linear algorithm which takes one iteration to solve the system of equations.

`secant` (bool)	Flag to indicate to use secant stiffness. (optional)
`initial` (bool)	Flag to indicate to use initial stiffness. (optional)
`factorOnce` (bool)	Flag to indicate to only set up and factor matrix once. (optional)

Note: As the tangent matrix typically will not change during the analysis in case of an elastic system it is highly advantageous to use the -factorOnce option. Do not use this option if you have a nonlinear system and you want the tangent used to be actual tangent at time of the analysis step.

Newton Algorithm

algorithm(*'Newton'*, *secant=False*, *initial=False*, *initialThenCurrent=False*)

Create a Newton-Raphson algorithm. The Newton-Raphson method is the most widely used and most robust method for solving nonlinear algebraic equations.

`secant` (bool)	Flag to indicate to use secant stiffness. (optional)
`initial` (bool)	Flag to indicate to use initial stiffness.(optional)
`initialThenCurrent` (bool)	Flag to indicate to use initial stiffness on first step, then use current stiffness for subsequent steps. (optional)

Newton with Line Search

algorithm(*'NewtonLineSearch'*, *Bisection=False*, *Secant=False*, *RegulaFalsi=False*, *InitialInterpolated=False*, *tol=0.8*, *maxIter=10*, *minEta=0.1*, *maxEta=10.0*)

Create a NewtonLineSearch algorithm. Introduces line search to the Newton algorithm to solve the nonlinear residual equation.

`Bisection` (bool)	Flag to use Bisection line search. (optional)
`Secant` (bool)	Flag to use Secant line search. (optional)
`RegulaFalsi` (bool)	Flag to use RegulaFalsi line search. (optional)
`InitialInterpolated` (bool)	Flag to use InitialInterpolated line search.(optional)
`tol` (float)	Tolerance for search. (optional)
`maxIter` (float)	Max num of iterations to try. (optional)
`minEta` (float)	Min η value. (optional)
`maxEta` (float)	Max η value. (optional)

Modified Newton Algorithm

algorithm(*'ModifiedNewton'*, *secant=False*, *initial=False*)

Create a ModifiedNewton algorithm. The difference to Newton is that the tangent at the initial guess is used in the iterations, instead of the current tangent.

`secant` (bool)	Flag to indicate to use secant stiffness. (optional)
`initial` (bool)	Flag to indicate to use initial stiffness.(optional)

Krylov-Newton Algorithm

algorithm(*'KrylovNewton'*, *iterate='current'*, *increment='current'*, *maxDim=3*)

Create a KrylovNewton algorithm which uses a Krylov subspace accelerator to accelerate the convergence of the ModifiedNewton.

`iterate` (str)	Tangent to iterate on, `'current'`, `'initial'`, `'noTangent'` (optional)
`increment` (str)	Tangent to increment on, `'current'`, `'initial'`, `'noTangent'` (optional)
`maxDim` (int)	Max number of iterations until the tangent is reformed and the acceleration restarts. (optional)

SecantNewton Algorithm

algorithm(*'SecantNewton'*, *iterate='current'*, *increment='current'*, *maxDim=3*)

Create a SecantNewton algorithm which uses the two-term update to accelerate the convergence of the ModifiedNewton.

The default "cut-out" values recommended by Crisfield (R1=3.5, R2=0.3) are used.

`iterate` (str)	Tangent to iterate on, `'current'`, `'initial'`, `'noTangent'` (optional)
`increment` (str)	Tangent to increment on, `'current'`, `'initial'`, `'noTangent'` (optional)
`maxDim` (int)	Max number of iterations until the tangent is reformed and the acceleration restarts. (optional)

RaphsonNewton Algorithm

algorithm(*'RaphsonNewton'*, *iterate='current'*, *increment='current'*)

Create a RaphsonNewton algorithm which uses Raphson accelerator.

`iterate` (str)	Tangent to iterate on, `'current'`, `'initial'`, `'noTangent'` (optional)
`increment` (str)	Tangent to increment on, `'current'`, `'initial'`, `'noTangent'` (optional)

PeriodicNewton Algorithm

algorithm(*'PeriodicNewton'*, *iterate='current'*, *increment='current'*, *maxDim=3*)

Create a PeriodicNewton algorithm using periodic accelerator.

`iterate` (str)	Tangent to iterate on, `'current'`, `'initial'`, `'noTangent'` (optional)
`increment` (str)	Tangent to increment on, `'current'`, `'initial'`, `'noTangent'` (optional)
`maxDim` (int)	Max number of iterations until the tangent is reformed and the acceleration restarts. (optional)

BFGS Algorithm

algorithm(*'BFGS'*, *secant=False*, *initial=False*, *count=10*)

Create a BFGS algorithm. The BFGS method is one of the most effective matrix-update or quasi Newton methods for iteration on a nonlinear system of equations. The method computes new search directions at each iteration step based on the initial jacobian, and subsequent trial solutions. The unlike regular Newton does not require the tangent matrix be reformulated and refactored at every iteration, however unlike ModifiedNewton it does not rely on the tangent matrix from a previous iteration.

`secant` (bool)	Flag to indicate to use secant stiffness. (optional)
`initial` (bool)	Flag to indicate to use initial stiffness.(optional)
`count` (int)	Number of iterations. (optional)

Broyden Algorithm

algorithm(*'Broyden'*, *secant=False*, *initial=False*, *count=10*)

Create a Broyden algorithm for general unsymmetric systems which performs successive rank-one updates of the tangent at the first iteration of the current time step.

`secant` (bool)	Flag to indicate to use secant stiffness. (optional)
`initial` (bool)	Flag to indicate to use initial stiffness.(optional)
`count` (int)	Number of iterations. (optional)

1.5.6 integrator commands

integrator(*intType*, **intArgs*)

This command is used to construct the Integrator object. The Integrator object determines the meaning of the terms in the system of equation object Ax=B.

The Integrator object is used for the following:

- determine the predictive step for time t+dt
- specify the tangent matrix and residual vector at any iteration
- determine the corrective step based on the displacement increment dU

`intType` (str)	integrator type
`intArgs` (list)	a list of integrator arguments

The following contain information about available `intType`:

LoadControl

integrator(*'LoadControl'*, *incr*, *numIter=1*, *minIncr=incr*, *maxIncr=incr*)

Create a OpenSees LoadControl integrator object.

`incr` (float)	Load factor increment λ.
`numIter` (int)	Number of iterations the user would like to occur in the solution algorithm. (optional)
`minIncr` (float)	Min stepsize the user will allow λ_{min}. (optional)
`maxIncr` (float)	Max stepsize the user will allow λ_{max}. (optional)

1. The change in applied loads that this causes depends on the active load pattern (those load pattern not set constant) and the loads in the load pattern. If the only active load acting on the Domain are in load pattern with a Linear time series with a factor of 1.0, this integrator is the same as the classical load control method.
2. The optional arguments are supplied to speed up the step size in cases where convergence is too fast and slow down the step size in cases where convergence is too slow.

DisplacementControl

integrator(*'DisplacementControl'*, *nd*, *dof*, *incr*, *numIter=1*, *dUmin=incr*, *dUmax=incr*)

Create a DisplacementControl integrator. In an analysis step with Displacement Control we seek to determine the time step that will result in a displacement increment for a particular degree-of-freedom at a node to be a prescribed value.

`nd` (int)	tag of node whose response controls solution
`dof` (int)	Degree of freedom at the node, 1 through ndf.
`incr` (float)	First displacement increment ΔU_{dof}.
`numIter` (int)	Number of iterations the user would like to occur in the solution algorithm. (optional)
`minIncr` (float)	Min stepsize the user will allow ΔU_{min}. (optional)
`maxIncr` (float)	Max stepsize the user will allow ΔU_{max}. (optional)

Minimum Unbalanced Displacement Norm

integrator(*'MinUnbalDispNorm'*, *dlambda1*, *Jd=1*, *minLambda=dlambda1*, *maxLambda=dlambda1*, *det=False*)

Create a MinUnbalDispNorm integrator.

`dlambda1` (float)	First load increment (pseudo-time step) at the first iteration in the next invocation of the analysis command.
`Jd` (int)	Factor relating first load increment at subsequent time steps. (optional)
`minLambda` (float)	Min load increment. (optional)
`maxLambda` (float)	Max load increment. (optional)

Arc-Length Control

integrator(*'ArcLength'*, *s*, *alpha*)

Create a ArcLength integrator. In an analysis step with ArcLength we seek to determine the time step that will result in our constraint equation being satisfied.

`s` (float)	The arcLength.
`alpha` (float)	α a scaling factor on the reference loads.

Central Difference

integrator(*'CentralDifference'*)

Create a centralDifference integrator.

1. The calculation of $U_t + \Delta t$, is based on using the equilibrium equation at time t. For this reason the method is called an explicit integration method.
2. If there is no rayleigh damping and the C matrix is 0, for a diagonal mass matrix a diagonal solver may and should be used.
3. For stability, $\frac{\Delta t}{T_n} < \frac{1}{\pi}$

Newmark Method

integrator(*'Newmark'*, *gamma*, *beta*, *formD=True*)

Create a Newmark integrator.

gamma (float)	γ factor.
beta (float)	β factor.
formD (bool)	Flag to indicate if use displacement as primary variable. If not, use acceleration. (optional)

1. If the accelerations are chosen as the unknowns and β is chosen as 0, the formulation results in the fast but conditionally stable explicit Central Difference method. Otherwise the method is implicit and requires an iterative solution process.
2. Two common sets of choices are
   (a) Average Acceleration Method ($\gamma = \frac{1}{2}, \beta = \frac{1}{4}$)
   (b) Linear Acceleration Method ($\gamma = \frac{1}{2}, \beta = \frac{1}{6}$)
3. $\gamma > \frac{1}{2}$ results in numerical damping proportional to $\gamma - \frac{1}{2}$
4. The method is second order accurate if and only if $\gamma = \frac{1}{2}$
5. The method is unconditionally stable for $\beta >= \frac{\gamma}{2} >= \frac{1}{4}$

Hilber-Hughes-Taylor Method

integrator(*'HHT'*, *alpha*, *gamma=1.5-alpha*, *beta=(2-alpha)^2/4*)

Create a Hilber-Hughes-Taylor (HHT) integrator. This is an implicit method that allows for energy dissipation and second order accuracy (which is not possible with the regular Newmark object). Depending on choices of input parameters, the method can be unconditionally stable.

alpha (float)	α factor.
gamma (float)	γ factor. (optional)
beta (float)	β factor. (optional)

1. Like Mewmark and all the implicit schemes, the unconditional stability of this method applies to linear problems. There are no results showing stability of this method over the wide range of nonlinear problems that potentially exist. Experience indicates that the time step for implicit schemes in nonlinear situations can be much greater than those for explicit schemes.
2. α = 1.0 corresponds to the Newmark method.
3. α should be between 0.67 and 1.0. The smaller the α the greater the numerical damping.
4. γ and β are optional. The default values ensure the method is second order accurate and unconditionally stable when α is $\frac{2}{3} <= \alpha <= 1.0$. The defaults are:

   $\beta = \frac{(2-\alpha)^2}{4}$

   and

   $\gamma = \frac{3}{2} - \alpha$

Generalized Alpha Method

integrator(*'GeneralizedAlpha'*, *alphaM*, *alphaF*, *gamma=0.5+alphaM-alphaF*, *beta=(1+alphaM-alphaF)^2/4*)

Create a GeneralizedAlpha integrator. This is an implicit method that like the HHT method allows for high frequency energy dissipation and second order accuracy, i.e. Δt^2. Depending on choices of input parameters, the method can be unconditionally stable.

`alphaM` (float)	α_M factor.
`alphaF` (float)	α_F factor.
`gamma` (float)	γ factor. (optional)
`beta` (float)	β factor. (optional)

1. Like Newmark and all the implicit schemes, the unconditional stability of this method applies to linear problems. There are no results showing stability of this method over the wide range of nonlinear problems that potentially exist. Experience indicates that the time step for implicit schemes in nonlinear situations can be much greater than those for explicit schemes.
2. α_M = 1.0, α_F = 1.0 produces the Newmark Method.
3. α_M = 1.0 corresponds to the `integrator.HHT()` method.
4. The method is second-order accurate provided $\gamma = \frac{1}{2} + \alpha_M - \alpha_F$
5. The method is unconditionally stable provided $\alpha_M >= \alpha_F >= \frac{1}{2}, \beta >= \frac{1}{4} + \frac{1}{2}(\gamma_M - \gamma_F)$
6. γ and β are optional. The default values ensure the method is unconditionally stable, second order accurate and high frequency dissipation is maximized.

   The defaults are:

   $\gamma = \frac{1}{2} + \alpha_M - \alpha_F$

   and

   $\beta = \frac{1}{4}(1 + \alpha_M - \alpha_F)^2$

TRBDF2

integrator(*'TRBDF2'*)

Create a TRBDF2 integrator. The TRBDF2 integrator is a composite scheme that alternates between the Trapezoidal scheme and a 3 point backward Euler scheme. It does this in an attempt to conserve energy and momentum, something Newmark does not always do.

As opposed to dividing the time-step in 2 as outlined in the Bathe2007, we just switch alternate between the 2 integration strategies,i.e. the time step in our implementation is double that described in the Bathe2007.

Explicit Difference

integrator(*'ExplicitDifference'*)

Create a ExplicitDifference integrator.

1. When using Rayleigh damping, the damping ratio of high vibration modes is overrated, and the critical time step size will be much smaller. Hence Modal damping is more suitable for this method.
2. There should be no zero element on the diagonal of the mass matrix when using this method.
3. Diagonal solver should be used when lumped mass matrix is used because the equations are uncoupled.
4. For stability, $\Delta t \leq \left(\sqrt{\zeta^2 + 1} - \zeta\right) \frac{2}{\omega}$

- *PFEM integrator*

1.5.7 analysis command

analysis(*analysisType*)

This command is used to construct the Analysis object, which defines what type of analysis is to be performed.

- determine the predictive step for time t+dt
- specify the tangent matrix and residual vector at any iteration
- determine the corrective step based on the displacement increment dU

analysisType (str)	char string identifying type of analysis object to be constructed. Currently 3 valid options: 1. `'Static'` - for static analysis 2. `'Transient'` - for transient analysis constant time step 3. `'VariableTransient'` - for transient analysis with variable time step 4. `'PFEM'` - for *PFEM analysis*.

Note: If the component objects are not defined before hand, the command automatically creates default component objects and issues warning messages to this effect. The number of warning messages depends on the number of component objects that are undefined.

1.5.8 eigen command

eigen(*solver='-genBandArpack'*, *numEigenvalues*)

Eigen value analysis. Return a list of eigen values.

numEigenvalues (int)	number of eigenvalues required
solver (str)	optional string detailing type of solver: `'-genBandArpack'`, `'-symmBandLapack'`, `'-fullGenLapack'`, (optional)

Note:

1. The eigenvectors are stored at the nodes and can be printed out using a Node Recorder, the nodeEigenvector command, or the Print command.
2. The default eigensolver is able to solve only for N-1 eigenvalues, where N is the number of inertial DOFs. When running into this limitation the -fullGenLapack solver can be used instead of the default Arpack solver.

1.5.9 analyze command

analyze(*numIncr=1*, *dt=0.0*, *dtMin=0.0*, *dtMax=0.0*, *Jd=0*)

Perform the analysis. Return `0` if successful, `<0` if **NOT** successful

`numIncr` (int)	Number of analysis steps to perform. (required except for *PFEM analysis*)
`dt` (float)	Time-step increment. (required for Transient analysis and VariableTransient analysis.‘)
`dtMin` (float)	Minimum time steps. (required for VariableTransient analysis)
`dtMax` (float)	Maximum time steps (required for VariableTransient analysis)
`Jd` (float)	Number of iterations user would like performed at each step. The variable transient analysis will change current time step if last analysis step took more or less iterations than this to converge (required for VariableTransient analysis)

1.6 Output Commands

Get outputs from OpenSees. These commands don't change internal states of OpenSees.

1.6.1 basicDeformation command

`basicDeformation`(*eleTag*)
Returns the deformation of the basic system for a beam-column element.

`eleTag` (int)	element tag.

1.6.2 basicForce command

`basicForce`(*eleTag*)
Returns the forces of the basic system for a beam-column element.

`eleTag` (int)	element tag.

1.6.3 basicStiffness command

`basicStiffness`(*eleTag*)
Returns the stiffness of the basic system for a beam-column element. A list of values in row order will be returned.

`eleTag` (int)	element tag.

1.6.4 eleDynamicalForce command

`eleDynamicalForce`(*eleTag*, *dof=-1*)
Returns the elemental dynamic force.

eleTag (int)	element tag.
dof (int)	specific dof at the element, (optional), if no dof is provided, a list of values for all dofs is returned.

1.6.5 eleForce command

eleForce (*eleTag*, *dof=-1*)
Returns the elemental resisting force.

eleTag (int)	element tag.
dof (int)	specific dof at the element, (optional), if no dof is provided, a list of values for all dofs is returned.

1.6.6 eleNodes command

eleNodes (*eleTag*)
Get nodes in an element

eletag (int)	element tag.

1.6.7 eleResponse command

eleResponse (*eleTag*, **args*)
This command is used to obtain the same element quantities as those obtained from the element recorder at a particular time step.

eletag (int)	element tag.
args (list)	same arguments as those specified in element recorder. These arguments are specific to the type of element being used.

1.6.8 getEleTags command

getEleTags (*'-mesh'*, *mtag*)
Get all elements in the domain or in a mesh.

mtag (int)	mesh tag. (optional)

1.6.9 getLoadFactor command

getLoadFactor (*patternTag*)
Returns the load factor λ for the pattern

patternTag (int)	pattern tag.

1.6.10 getNodeTags command

getNodeTags (*'-mesh'*, *mtag*)
Get all nodes in the domain or in a mesh.

`mtag` (int)	mesh tag. (optional)

1.6.11 getTime command

getTime ()
Returns the current time in the domain.

1.6.12 nodeAccel command

nodeAccel (*nodeTag*, *dof=-1*)
Returns the current acceleration at a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.13 nodeBounds command

nodeBounds ()
Get the boundary of all nodes. Return a list of boundary values.

1.6.14 nodeCoord command

nodeCoord (*nodeTag*, *dim=-1*)
Returns the coordinates of a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dimension at the node (1 through ndf), (optional), if no `dim` is provided, a list of values for all dimensions is returned.

1.6.15 nodeDisp command

nodeDisp (*nodeTag*, *dof=-1*)
Returns the current displacement at a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.16 nodeEigenvector command

nodeEigenvector (*eigenvector*, *dof=-1*)

Returns the eigenvector at a specified node.

`nodeTag` (int)	node tag.
`eigenvector` (int)	mode number of eigenvector to be returned
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.17 nodeMass command

nodeMass (*nodeTag*, *dof=-1*)

Returns the masss at a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.18 nodePressure command

nodePressure (*nodeTag*)

Returns the fluid pressures at a specified node if this is a fluid node.

`nodeTag` (int)	node tag.

1.6.19 nodeReaction command

nodeReaction (*nodeTag*, *dof=-1*)

Returns the reactions at a specified node. Must call `reactions()` command before this command.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.20 nodeResponse command

nodeResponse (*nodeTag*, *dof*, *responseID*)

Returns the responses at a specified node. Must call `responses` command before this command.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof of the response
`responseID` (int)	the id of responses: • Disp = 1 • Vel = 2 • Accel = 3 • IncrDisp = 4 • IncrDeltaDisp = 5 • Reaction = 6 • Unbalance = 7 • RayleighForces = 8

1.6.21 nodeVel command

nodeVel (*nodeTag*, *dof=-1*)
Returns the current velocity at a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.22 nodeUnbalance command

nodeUnbalance (*nodeTag*, *dof=-1*)
Returns the unbalanced force at a specified node.

`nodeTag` (int)	node tag.
`dof` (int)	specific dof at the node (1 through ndf), (optional), if no `dof` is provided, a list of values for all dofs is returned.

1.6.23 numFact command

numFact ()
Return the number of factorizations.

1.6.24 numIter command

numIter ()
Return the number of iterations.

1.6.25 printA command

printA (*'-file'*, *filename*)
print the contents of a FullGeneral system that the integrator creates to the screen or a file if the `'-file'`

option is used. If using a static integrator, the resulting matrix is the stiffness matrix. If a transient integrator, it will be some combination of mass and stiffness matrices.

`filename` (str)	name of file to which output is sent, by default, print to the screen. (optional)

1.6.26 printB command

printB(*'-file'*, *filename*)
print the right hand side of a FullGeneral system that the integrator creates to the screen or a file if the `'-file'` option is used.

`filename` (str)	name of file to which output is sent, by default, print to the screen. (optional)

1.6.27 printGID command

printGID(*filename*, *'-append'*, *'-eleRange'*, *startEle*, *endEle*)
Print in GID format.

`filename` (str)	output file name.
`'-append'` (str)	append to existing file. (optional)
`startEle` (int)	start element tag. (optional)
`endEle` (int)	end element tag. (optional)

1.6.28 printModel command

printModel(*'-file'*, *filename*, *'-JSON'*, *'-node'*, *'-flag'*, *flag*, **nodes=[]*, **eles=[]*)
This command is used to print output to screen or file.

`filename` (str)	name of file to which output is sent, by default, print to the screen. (optional)
`'-JSON'` (str)	print to a JSON file. (optional)
`'-node'` (str)	print node information. (optional)
`flag` (int)	integer flag to be sent to the print() method, depending on the node and element type (optional)
`nodes` (list (int))	a list of nodes tags to be printed, default is to print all, (optional)
`eles` (list (int))	a list of element tags to be printed, default is to print all, (optional)

Note: This command was called `print` in Tcl. Since `print` is a built-in function in Python, it is renamed to `printModel`.

1.6.29 record command

record()
This command is used to cause all the recorders to do a record on the current state of the model.

Note: A record is issued after every successfull static or transient analysis step. Sometimes the user may need the record to be issued on more occasions than this, for example if the user is just looking to record the eigenvectors after an eigen command or for example the user wishes to include the state of the model at time 0.0 before any analysis has been completed.

1.6.30 recorder command

recorder (*recorderType*, **recorderArgs*)

This command is used to generate a recorder object which is to monitor what is happening during the analysis and generate output for the user.

Return:

- >0 an integer tag that can be used as a handle on the recorder for the remove recorder commmand.
- -1 recorder command failed if integer -1 returned.

`recorderType` (str)	recorder type
`recorderArgs` (list)	a list of recorder arguments

The following contain information about available `recorderType`:

node recorder command

recorder (*'Node'*, *'-file'*, *filename*, *'-xml'*, *filename*, *'-binary'*, *filename*, *'-tcp'*, *inetAddress*, *port*, *'-precision'*, *nSD=6*, *'-timeSeries'*, *tsTag*, *'-time'*, *'-dT'*, *deltaT=0.0*, *'-closeOnWrite'*, *'-node'*, **nodeTags=[]*, *'-nodeRange'*, *startNode*, *endNode*, *'-region'*, *regionTag*, *'-dof'*, **dofs=[]*, *respType*)

The Node recorder type records the response of a number of nodes at every converged step.

`filename` (str)	name of file to which output is sent. file output is either in xml format (`'-xml'` option), textual (`'-file'` option) or binary (`'-binary'` option) which must pre-exist.
`inetAddr` (str)	ip address, "xx.xx.xx.xx", of remote machine to which data is sent. (optional)
`port` (int)	port on remote machine awaiting tcp. (optional)
`nSD` (int)	number of significant digits (optional)
`'-time'` (str)	using this option places domain time in first entry of each data line, default is to have time ommitted, (optional)
`'-closeOnWrite'` (str)	using this option will instruct the recorder to invoke a close on the data handler after every timestep. If this is a file it will close the file on every step and then re-open it for the next step. Note, this greatly slows the execution time, but is useful if you need to monitor the data during the analysis. (optional)
`deltaT` (float)	time interval for recording. will record when next step is `deltaT` greater than last recorder step. (optional, default: records at every time step)
`tsTag` (int)	the tag of a previously constructed TimeSeries, results from node at each time step are added to load factor from series (optional)
`nodeTags` (list (int))	list of tags of nodes whose response is being recorded (optional)
`startNode` (int)	tag for start node whose response is being recorded (optional)
`endNode` (int)	tag for end node whose response is being recorded (optional)
`regionTag` (int)	a region tag; to specify all nodes in the previously defined region. (optional)
`dofs` (list (int))	the specified dof at the nodes whose response is requested.
`resType` (list (str))	a string indicating response required. Response types are given in table below • `'disp'` displacement • `'vel'` velocity • `'accel'` acceleration • `'incrDisp'` incremental displacement • `'reaction'` nodal reaction • `'eigen i'` eigenvector for mode i • `'rayleighForces'` damping forces

Note: Only one of `'-file'`, `'-xml'`, `'-binary'`, `'-tcp'` will be used. If multiple specified last option is used.

node envelope recorder command

recorder (*'EnvelopeNode'*, *'-file'*, *filename*, *'-xml'*, *filename*, *'-precision'*, *nSD=6*, *'-timeSeries'*, *tsTag*, *'-time'*, *'-dT'*, *deltaT=0.0*, *'-closeOnWrite'*, *'-node'*, **nodeTags=[]*, *'-nodeRange'*, *startNode*, *endNode*, *'-region'*, *regionTag*, *'-dof'*, **dofs=[]*, *respType*)

The EnvelopeNode recorder type records the min, max and absolute max of a number of nodal response quantaties.

`filename` (str)	name of file to which output is sent. file output is either in xml format (`'-xml'` option), or textual (`'-file'` option) which must pre-exist.
`nSD` (int)	number of significant digits (optional)
`'-time'` (str)	using this option places domain time in first entry of each data line, default is to have time ommitted, (optional)
`'-closeOnWrite'` (str)	using this option will instruct the recorder to invoke a close on the data handler after every timestep. If this is a file it will close the file on every step and then re-open it for the next step. Note, this greatly slows the execution time, but is useful if you need to monitor the data during the analysis. (optional)
`deltaT` (float)	time interval for recording. will record when next step is `deltaT` greater than last recorder step. (optional, default: records at every time step)
`tsTag` (int)	the tag of a previously constructed TimeSeries, results from node at each time step are added to load factor from series (optional)
`nodeTags` (list (int))	list of tags of nodes whose response is being recorded (optional)
`startNode` (int)	tag for start node whose response is being recorded (optional)
`endNode` (int)	tag for end node whose response is being recorded (optional)
`regionTag` (int)	a region tag; to specify all nodes in the previously defined region. (optional)
`dofs` (list (int))	the specified dof at the nodes whose response is requested.
`resType` (list (str))	a string indicating response required. Response types are given in table below • `'disp'` displacement • `'vel'` velocity • `'accel'` acceleration • `'incrDisp'` incremental displacement • `'reaction'` nodal reaction • `'eigen i'` eigenvector for mode i

element recorder command

recorder (*'Element'*, *'-file'*, *filename*, *'-xml'*, *filename*, *'-binary'*, *filename*, *'-precision'*, *nSD=6*, *'-timeSeries'*, *tsTag*, *'-time'*, *'-dT'*, *deltaT=0.0*, *'-closeOnWrite'*, *'-ele'*, **eleTags=[]*, *'-eleRange'*, *startEle*, *endEle*, *'-region'*, *regionTag*, **args*)

The Element recorder type records the response of a number of elements at every converged step. The response recorded is element-dependent and also depends on the arguments which are passed to the setResponse() element method.

`filename` (str)	name of file to which output is sent. file output is either in xml format (`'-xml'` option), textual (`'-file'` option) or binary (`'-binary'` option) which must pre-exist.
`nSD` (int)	number of significant digits (optional)
`'-time'` (str)	using this option places domain time in first entry of each data line, default is to have time ommitted, (optional)
`'-closeOnWrite'` (str)	using this option will instruct the recorder to invoke a close on the data handler after every timestep. If this is a file it will close the file on every step and then re-open it for the next step. Note, this greatly slows the execution time, but is useful if you need to monitor the data during the analysis. (optional)
`deltaT` (float)	time interval for recording. will record when next step is `deltaT` greater than last recorder step. (optional, default: records at every time step)
`tsTag` (int)	the tag of a previously constructed TimeSeries, results from node at each time step are added to load factor from series (optional)
`eleTags` (list (int))	list of tags of elements whose response is being recorded (optional)
`startEle` (int)	tag for start node whose response is being recorded (optional)
`endEle` (int)	tag for end node whose response is being recorded (optional)
`regionTag` (int)	a region tag; to specify all nodes in the previously defined region. (optional)
`args` (list)	arguments which are passed to the setResponse() element method, all arguments must be in string format even for double and integer numbers because internally the setResponse() element method only accepts strings.

Note: The setResponse() element method is dependent on the element type, and is described with the `element()` Command.

element envelope recorder command

recorder (*'EnvelopeElement'*, *'-file'*, *filename*, *'-xml'*, *filename*, *'-binary'*, *filename*, *'-precision'*, *nSD=6*, *'-timeSeries'*, *tsTag*, *'-time'*, *'-dT'*, *deltaT=0.0*, *'-closeOnWrite'*, *'-ele'*, **eleTags=[]*, *'-eleRange'*, *startEle*, *endEle*, *'-region'*, *regionTag*, **args*)

The Envelope Element recorder type records the response of a number of elements at every converged step. The response recorded is element-dependent and also depends on the arguments which are passed to the setResponse() element method. When the object is terminated, through the use of a wipe, exit, or remove the object will output the min, max and absolute max values on 3 seperate lines of the output file for each quantity.

filename (str)	name of file to which output is sent. file output is either in xml format ('-xml' option), textual ('-file' option) or binary ('-binary' option) which must pre-exist.
nSD (int)	number of significant digits (optional)
'-time' (str)	using this option places domain time in first entry of each data line, default is to have time ommitted, (optional)
'-closeOnWrite' (str)	using this option will instruct the recorder to invoke a close on the data handler after every timestep. If this is a file it will close the file on every step and then re-open it for the next step. Note, this greatly slows the execution time, but is useful if you need to monitor the data during the analysis. (optional)
deltaT (float)	time interval for recording. will record when next step is deltaT greater than last recorder step. (optional, default: records at every time step)
tsTag (int)	the tag of a previously constructed TimeSeries, results from node at each time step are added to load factor from series (optional)
eleTags (list (int))	list of tags of elements whose response is being recorded (optional)
startEle (int)	tag for start node whose response is being recorded (optional)
endEle (int)	tag for end node whose response is being recorded (optional)
regionTag (int)	a region tag; to specify all nodes in the previously defined region. (optional)
args (list)	arguments which are passed to the setResponse() element method

Note: The setResponse() element method is dependent on the element type, and is described with the `element()` Command.

pvd recorder command

recorder (*'PVD'*, *filename*, *'-precision'*, *precision=10*, *'-dT'*, *dT=0.0*, **res*)

Create a PVD recorder.

filename (str)	the name for filename.pvd and filename/ directory, which must pre-exist.
precision (int)	the precision of data. (optional)
dT (float)	the time interval for recording. (optional)
res (list (str))	a list of (str) of responses to be recorded, (optional) • **'disp'** – 'vel' – 'accel' – 'incrDisp' – 'reaction' – 'pressure' – 'unbalancedLoad' – 'mass' – 'eigen'

1.6.31 sectionForce command

sectionForce (*eleTag*, *secNum*, *dof*)
Returns the section force for a beam-column element.

eleTag (int)	element tag.
secNum (int)	section number, i.e. the Gauss integratio number
dof (int)	the dof of the section

1.6.32 sectionDeformation command

sectionDeformation (*eleTag*, *secNum*, *dof*)
Returns the section deformation for a beam-column element.

eleTag (int)	element tag.
secNum (int)	section number, i.e. the Gauss integratio number
dof (int)	the dof of the section

1.6.33 sectionStiffness command

sectionStiffness (*eleTag*, *secNum*, *dof*)
Returns the section stiffness matrix for a beam-column element. A list of values in the row order will be returned.

eleTag (int)	element tag.
secNum (int)	section number, i.e. the Gauss integratio number
dof (int)	the dof of the section

1.6.34 sectionFlexibility command

sectionFlexibility (*eleTag*, *secNum*, *dof*)
Returns the section flexibility matrix for a beam-column element. A list of values in the row order will be returned.

eleTag (int)	element tag.
secNum (int)	section number, i.e. the Gauss integratio number
dof (int)	the dof of the section

1.6.35 sectionLocation command

sectionLocation (*eleTag*, *secNum*)
Returns the locations of integration points of a section for a beam-column element.

eleTag (int)	element tag.
secNum (int)	section number, i.e. the Gauss integration number

1.6.36 sectionWeight command

sectionWeight (*eleTag*, *secNum*)
Returns the weights of integration points of a section for a beam-column element.

`eleTag` (int)	element tag.
`secNum` (int)	section number, i.e. the Gauss integration number

1.6.37 systemSize command

systemSize ()
Return the size of the system.

1.6.38 testIter command

testIter ()
Returns the number of iterations the convergence test took in the last analysis step

1.6.39 testNorm command

testNorm ()
Returns the norms from the convergence test for the last analysis step.

Note: The size of norms will be equal to the max number of iterations specified. The first `testIter` of these will be non-zero, the remaining ones will be zero.

1.6.40 version command

version ()
Return the current OpenSees version.

1.7 Utility Commands

These commands are used to monitor and change the state of the model.

1.7.1 convertBinaryToText command

convertBinaryToText (*inputfile*, *outputfile*)
Convert binary file to text file

`inputfile` (str)	input file name.
`outputfile` (str)	output file name.

1.7.2 convertTextToBinary command

convertTextToBinary (*inputfile*, *outputfile*)
Convert text file to binary file

inputfile (str)	input file name.
outputfile (str)	output file name.

1.7.3 database command

database (*type*, *dbName*)
Create a database.

type (str)	database type: • 'File' - outputs database into a file • 'MySQL' - creates a SQL database • 'BerkeleyDB' - creates a BerkeleyDB database
dbName (str)	database name.

1.7.4 domainChange command

domainChange ()
Mark the domain has changed manually.

1.7.5 InitialStateAnalysis command

InitialStateAnalysis (*flag*)
Set the initial state analysis to 'on' or 'off'

flag (str)	'on' or 'off'

1.7.6 loadConst command

loadConst (*'-time'*, *pseudoTime*)
This command is used to set the loads constant in the domain and to also set the time in the domain. When setting the loads constant, the procedure will invoke setLoadConst() on all LoadPattern objects which exist in the domain at the time the command is called.

pseudoTime (float)	Time domain is to be set to (optional)

Note: Load Patterns added afer this command is invoked are not set to constant.

1.7.7 modalDamping command

modalDamping (*factor*)
Set modal damping factor. The `eigen()` must be called before.

`factor` (float)	damping factor.

1.7.8 reactions command

reactions (*'-dynamic'*, *'-rayleigh'*)
Calculate the reactions. Call this command before the `nodeReaction()`.

`'-dynamic'` (str)	Include dynamic effects.
`'-rayleigh'` (str)	Include rayleigh damping.

1.7.9 remove command

remove (*type*, *tag*)
This commmand is used to remove components from the model.

`type` (str)	type of the object, `'ele'`, `'loadPattern'`, `'parameter'`, `'node'`, `'timeSeries'`, `'sp'`, `'mp'`.
`tag` (int)	tag of the object

remove (*'recorders'*)
Remove all recorder objects.

remove (*'sp'*, *nodeTag*, *dofTag*, *patternTag*)
Remove a sp object based on node

`nodeTag` (int)	node tag
`dof` (int)	dof the sp constrains
`patternTag` (int)	pattern tag, (optional)

1.7.10 reset command

reset ()
This command is used to set the state of the domain to its original state.

Note: It iterates over all components of the domain telling them to set their state back to the initial state. This is not always the same as going back to the state of the model after initial model generation, e.g. if elements have been removed.

1.7.11 restore command

restore (*commitTag*)
Restore data from database, which should be created through `database()`.

`commitTag` (int)	a tag identify the commit

1.7.12 save command

save (*commitTag*)

Save current state to database, which should be created through `database()`.

`commitTag` (int)	a tag identify the commit

1.7.13 setTime command

setTime (*pseudoTime*)

This command is used to set the time in the Domain.

`pseudoTime` (float)	Time domain to be set

1.7.14 setNodeCoord command

setNodeCoord (*nodeTag*, *dim*, *value*)

set the nodal coodinate at the specified dimension.

`nodeTag` (int)	node tag.
`dim` (int)	the dimension of the coordinate to be set.
`value` (float)	coordinate value

1.7.15 setNodeDisp command

setNodeDisp (*nodeTag*, *dim*, *value*, *'-commit'*)

set the nodal displacement at the specified dimension.

`nodeTag` (int)	node tag.
`dim` (int)	the dimension of the dispinate to be set.
`value` (float)	displacement value
`'-commit'` (str)	commit nodal state. (optional)

1.7.16 setNodeVel command

setNodeVel (*nodeTag*, *dim*, *value*, *'-commit'*)

set the nodal velocity at the specified dimension.

`nodeTag` (int)	node tag.
`dim` (int)	the dimension of the velinate to be set.
`value` (float)	velocity value
`'-commit'` (str)	commit nodal state. (optional)

1.7.17 setNodeAccel command

setNodeAccel (*nodeTag*, *dim*, *value*, *'-commit'*)
set the nodal acceleration at the specified dimension.

`nodeTag` (int)	node tag.
`dim` (int)	the dimension of the accelinate to be set.
`value` (float)	acceleration value
`'-commit'` (str)	commit nodal state. (optional)

1.7.18 setPrecision command

setPrecision (*precision*)
Set the precision for screen output.

`precision` (int)	the precision number.

1.7.19 setElementRayleighDampingFactors command

setElementRayleighDampingFactors (*eleTag*, *alphaM*, *betaK*, *betaK0*, *betaKc*)
Set the `rayleigh()` damping for an element.

`eleTag` (int)	element tag
`alphaM` (float)	factor applied to elements or nodes mass matrix
`betaK` (float)	factor applied to elements current stiffness matrix.
`betaK0` (float)	factor applied to elements initial stiffness matrix.
`betaKc` (float)	factor applied to elements committed stiffness matrix.

1.7.20 start command

start ()
Start the timer

1.7.21 stop command

stop ()
Stop the timer and print timing information.

1.7.22 stripXML command

stripXML (*inputml*, *outputdata*, *outputxml*)
Strip a xml file to a data file and a descriptive file.

`inputxml` (str)	input xml file name.
`outputdata` (str)	output data file name.
`outputxml` (str)	output xml file name.

1.7.23 updateElementDomain command

updateElementDomain()
Update elements in the domain.

1.7.24 wipe command

wipe()
This command is used to destroy all constructed objects, i.e. all components of the model, all components of the analysis and all recorders.

This command is used to start over without having to exit and restart the interpreter. It causes all elements, nodes, constraints, loads to be removed from the domain. In addition it deletes all recorders, analysis objects and all material objects created by the model builder.

1.7.25 wipeAnalysis command

wipeAnalysis()
This command is used to destroy all components of the Analysis object, i.e. any objects created with system, numberer, constraints, integrator, algorithm, and analysis commands.

1.8 FSI Commands

These commands are related to the Fluid-Structure Interaction analysis in OpenSees.

1.8.1 mesh command

mesh(*type*, *tag*, **args*)
Create a mesh object. See below for available mesh types.

line mesh

mesh(*'line'*, *tag*, *numnodes*, **ndtags*, *id*, *ndf*, *meshsize*, *eleType=''*, **eleArgs=[]*)
Create a line mesh object.

`tag` (int)	mesh tag.
`numnodes` (int)	number of nodes for defining consective lines.
`ndtags` (list (int))	the node tags
`id` (int)	mesh id. Meshes with same id are considered as same structure of fluid identity. • `id` = 0 : not in FSI • `id` > 0 : structure • `id` < 0 : fluid
`ndf` (int)	ndf for nodes to be created.
`meshsize` (float)	mesh size.
`eleType` (str)	the type of the element, (optional) • `'elasticBeamColumn'` • `'forceBeamColumn'` • `'dispBeamColumn'` if no type is given, only nodes are created
`eleArgs` (list)	a list of element arguments. (optional)

triangular mesh

mesh (*'tri'*, *tag*, *numlines*, **ltags*, *id*, *ndf*, *meshsize*, *eleType=''*, **eleArgs=[]*)

Create a triangular mesh object.

`tag` (int)	mesh tag.
`numlines` (int)	number of lines (*line mesh*) for defining a polygon.
`ltags` (list (int))	the *line mesh* tags
`id` (int)	mesh id. Meshes with same id are considered as same structure of fluid identity. • `id` = 0 : not in FSI • `id` > 0 : structure • `id` < 0 : fluid
`ndf` (int)	ndf for nodes to be created.
`meshsize` (float)	mesh size.
`eleType` (str)	the element type, (optional) • `'PFEMElement2DBubble'` • `'PFEMElement2DQuasi'` • `'tri31'` if no type is given, only nodes are created
`eleArgs` (list)	a list of element arguments. (optional)

particle mesh

mesh (*'part'*, *tag*, *type*, **pArgs*, *eleType=''*, **eleArgs=[]*, *'-vel'*, **vel0*, *'-pressure'*, *p0*)

Create a particle mesh which is used for background mesh.

tag (int)	mesh tag.
type (str)	type of the mesh, 'quad', 'tri', 'line', 'point'
pArgs (list (float))	coordinates of points defining the mesh region • 'quad' : [x1, y1, x2, y2, x3, y3, x4, y4] • 'tri' : [x1, y1, x2, y2, x3, y3] • 'line' : [x1, y1, x2, y2] • 'point' : [x1, y1]
eleType (str)	the element type, (optional) • 'PFEMElement2DBubble' • 'PFEMElement2DQuasi' • 'tri31' if no type is given, only nodes are created
eleArgs (list)	a list of element arguments. (optional)
vel0 (list (float))	a list of initial velocities. (optional)
p0 (float)	initial pressure. (optional)

1.8.2 remesh command

remesh(*alpha=-1.0*)

- $\alpha \geq 0$ for updating moving mesh.
- $\alpha < 0$ for updating background mesh.

alpha (float)	Parameter for the α method to construct a mesh from the node cloud of moving meshes. (optional) • $\alpha = 0$: no elements are created • large α : all elements in the convex hull are created • $1.0 < \alpha < 2.0$: usually gives a good shape

1.8.3 PFEM integrator

integrator(*'PFEM'*)
Create a PFEM Integrator.

1.8.4 PFEM SOE

system(*'PFEM'*, *'-compressible'*)
Create a incompressible PFEM system of equations using the Umfpack solver

-compressible	Solve using a quasi-incompressible formulation. (optional)

1.8.5 PFEM test

test (*'PFEM'*, *tolv*, *tolp*, *tolrv*, *tolrp*, *tolrelv*, *tolrelp*, *iter*, *maxincr*, *pFlag=0*, *nType=2*)
Create a PFEM test, which check both increments and residual for velocities and pressures.

`tolv` (float)	Tolerance for velocity increments
`tolp` (float)	Tolerance for pressure increments
`tolrv` (float)	Tolerance for velocity residual
`tolrp` (float)	Tolerance for pressure residual
`tolrv` (float)	Tolerance for relative velocity increments
`tolrp` (float)	Tolerance for relative pressure increments
`iter` (int)	Max number of iterations to check
`maxincr` (int)	Max times for error increasing
`pFlag` (int)	Print flag (optional): • 0 print nothing. • 1 print information on norms each time `test()` is invoked. • 2 print information on norms and number of iterations at end of successful test. • 4 at each step it will print the norms and also the ΔU and $R(U)$ vectors. • 5 if it fails to converge at end of `numIter` it will print an error message **but return a successfull test**.
`nType` (int)	Type of norm, (0 = max-norm, 1 = 1-norm, 2 = 2-norm). (optional)

1.8.6 PFEM analysis

analysis (*'PFEM'*, *dtmax*, *dtmin*, *gravity*, *ratio=0.5*)
Create a OpenSees PFEMAnalysis object.

`dtmax` (float)	Maximum time steps.
`dtmin` (float)	Mimimum time steps.
`gravity` (float)	Gravity acceleration used to move isolated particles.
`ratio` (float)	The ratio to reduce time steps if it was not converged. (optional)

1.9 Sensitivity Commands

These commands are for sensitivity analysis in OpenSees.

1.9.1 computeGradients command

computeGradients ()
This command is used to perform a sensitivity analysis. If the user wants to call this command, then the `'-computeByCommand'` should be set in the `sensitivityAlgorithm` command.

1.9.2 sensitivityAlgorithm command

sensitivityAlgorithm (*type*)

This command is used to create a sensitivity algorithm.

`type` (str)	the type of the sensitivity algorithm, • `'-compuateAtEachStep'` automatically compute at the end of each step • `'-compuateByCommand'` compute by calling `computeGradients`.

1.9.3 sensNodeDisp command

sensNodeDisp (*nodeTag, dof, paramTag*)

Returns the current displacement sensitivity to a parameter at a specified node.

`nodeTag` (int)	node tag
`dof` (int)	specific dof at the node (1 through ndf)
`paramTag` (int)	parameter tag

1.9.4 sensNodeVel command

sensNodeVel (*nodeTag, dof, paramTag*)

Returns the current velocity sensitivity to a parameter at a specified node.

`nodeTag` (int)	node tag
`dof` (int)	specific dof at the node (1 through ndf)
`paramTag` (int)	parameter tag

1.9.5 sensNodeAccel command

sensNodeAccel (*nodeTag, dof, paramTag*)

Returns the current acceleration sensitivity to a parameter at a specified node.

`nodeTag` (int)	node tag
`dof` (int)	specific dof at the node (1 through ndf)
`paramTag` (int)	parameter tag

1.9.6 sensLambda command

sensLambda (*patternTag, paramTag*)

Returns the current load factor sensitivity to a parameter in a load pattern.

`patternTag` (int)	load pattern tag
`paramTag` (int)	parameter tag

1.9.7 sensSectionForce command

sensSectionForce (*eleTag*[, *secNum*], *dof*, *paramTag*)

Returns the current section force sensitivity to a parameter at a specified element and section.

`eleTag` (int)	element tag
`secNum` (int)	section number (optional)
`dof` (int)	specific dof at the element (1 through element force ndf)
`paramTag` (int)	parameter tag

1.9.8 sensNodePressure command

sensNodePressure (*nodeTag*, *paramTag*)

Returns the current pressure sensitivity to a parameter at a specified node.

`nodeTag` (int)	node tag
`paramTag` (int)	parameter tag

1.10 Reliability Commands

These commands are for reliability analysis in OpenSees.

1.10.1 randomVariable command

randomVariable (*tag*, *dist*, *'-mean'*, *mean*, *'-stdv'*, *stdv*, *'-startPoint'*, *startPoint*, *'-parameters'*, **params*)

Create a random variable with user specified distribution

tag (int)	random variable tag
dist (str)	random variable distribution • 'normal' • 'lognormal' • 'gamma' • 'shiftedExponential' • 'shiftedRayleigh' • 'exponential' • 'rayleigh' • 'uniform' • 'beta' • 'type1LargestValue' • 'type1SmallestValue' • 'type2LargestValue' • 'type3SmallestValue' • 'chiSquare' • 'gumbel' • 'weibull' • 'laplace' • 'pareto'
mean (float)	mean value
stdv (float)	standard deviation
startPoint (float)	starting point of the distribution
params (list (int))	a list of parameter tags

1.11 Structural Examples

1.11.1 Elastic Truss Analysis

1. The source code is shown below, which can be downloaded here.
2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. Run the source code in your favorate Python program and should see Passed! in the results.

```
1  import sys
2  sys.path.append('/path/to/OpenSeesPy')
3  from opensees import *
4
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # ------------------------------
9  # Start of model generation
10 # -----------------------------
11
12 # remove existing model
13 wipe()
14
15 # set modelbuilder
16 model('basic', '-ndm', 2, '-ndf', 2)
```

(continues on next page)

(continued from previous page)

```

# create nodes
node(1, 0.0, 0.0)
node(2, 144.0,  0.0)
node(3, 168.0,  0.0)
node(4,  72.0, 96.0)

# set boundary condition
fix(1, 1, 1)
fix(2, 1, 1)
fix(3, 1, 1)

# define materials
uniaxialMaterial("Elastic", 1, 3000.0)

# define elements
element("Truss",1,1,4,10.0,1)
element("Truss",2,2,4,5.0,1)
element("Truss",3,3,4,5.0,1)

# create TimeSeries
timeSeries("Linear", 1)

# create a plain load pattern
pattern("Plain", 1, 1)

# Create the nodal load - command: load nodeID xForce yForce
load(4, 100.0, -50.0)

# ------------------------------
# Start of analysis generation
# ------------------------------

# create SOE
system("BandSPD")

# create DOF number
numberer("RCM")

# create constraint handler
constraints("Plain")

# create integrator
integrator("LoadControl", 1.0)

# create algorithm
algorithm("Linear")

# create analysis object
analysis("Static")

# perform the analysis
analyze(1)

ux = nodeDisp(4,1)
uy = nodeDisp(4,2)
if abs(ux-0.53009277713228375450)<1e-12 and abs(uy+0.17789363846931768864)<1e-12:
```

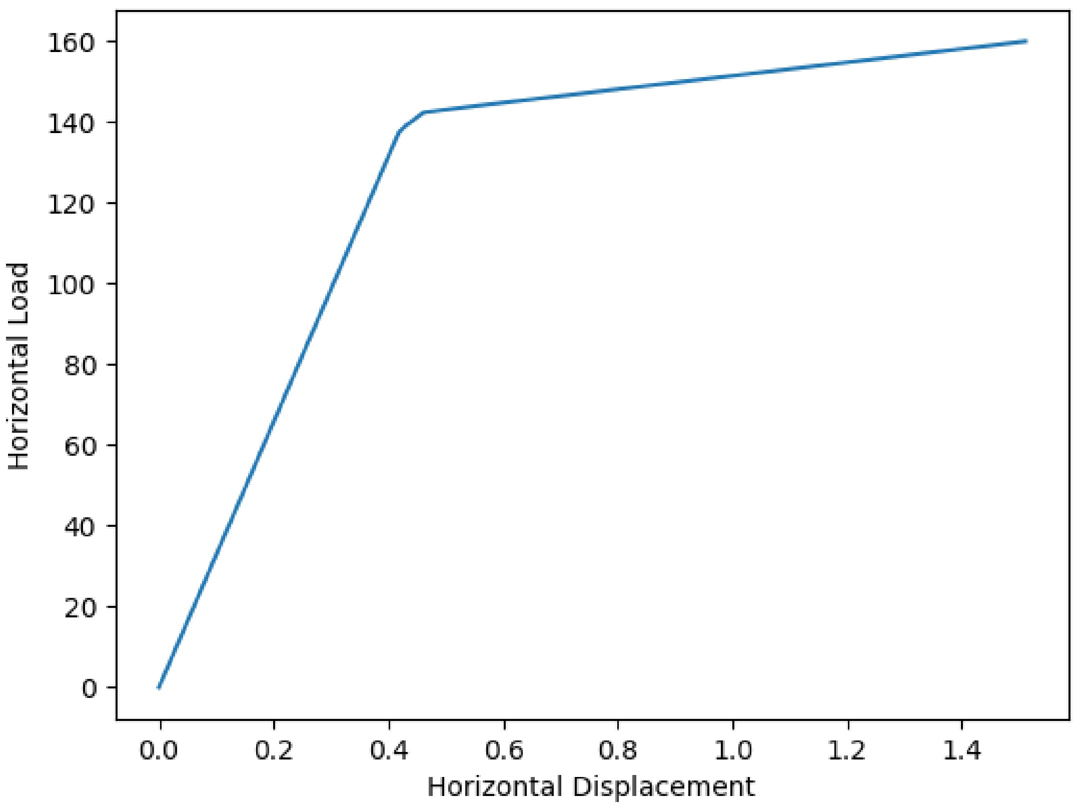
(continues on next page)

(continued from previous page)

```
    print("Passed!")
else:
    print("Failed!")
```

1.11.2 Nonlinear Truss Analysis

1. The source code is shown below, which can be downloaded `here`.
2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. Make sure the numpy and matplotlib packages are installed in your Python distribution.
4. Run the source code in your favorite Python program and should see




```
1 import sys
2 sys.path.append('/path/to/OpenSeesPy')
3 from opensees import *
4 import numpy as np
5 import matplotlib.pyplot as plt
6 
7 # ------------------------------
8 # Start of model generation
9 # -----------------------------
10 
```

(continues on next page)

(continued from previous page)

```
# set modelbuilder
wipe()
model('basic', '-ndm', 2, '-ndf', 2)

# variables
A = 4.0
E = 29000.0
alpha = 0.05
sY = 36.0
udisp = 2.5
Nsteps = 1000
Px = 160.0
Py = 0.0

# create nodes
node(1, 0.0, 0.0)
node(2, 72.0, 0.0)
node(3, 168.0, 0.0)
node(4, 48.0, 144.0)

# set boundary condition
fix(1, 1, 1)
fix(2, 1, 1)
fix(3, 1, 1)

# define materials
uniaxialMaterial("Hardening", 1, E, sY, 0.0, alpha/(1-alpha)*E)

# define elements
element("Truss",1,1,4,A,1)
element("Truss",2,2,4,A,1)
element("Truss",3,3,4,A,1)

# create TimeSeries
timeSeries("Linear", 1)

# create a plain load pattern
pattern("Plain", 1, 1)

# Create the nodal load
load(4, Px, Py)

# ------------------------------
# Start of analysis generation
# ------------------------------

# create SOE
system("ProfileSPD")

# create DOF number
numberer("Plain")

# create constraint handler
constraints("Plain")

# create integrator
integrator("LoadControl", 1.0/Nsteps)
```

(continues on next page)

(continued from previous page)

```

# create algorithm
algorithm("Newton")

# create test
test('NormUnbalance',1e-8, 10)

# create analysis object
analysis("Static")

# ------------------------------
# Finally perform the analysis
# ------------------------------

# perform the analysis
data = np.zeros((Nsteps+1,2))
for j in range(Nsteps):
    analyze(1)
    data[j+1,0] = nodeDisp(4,1)
    data[j+1,1] = getLoadFactor(1)*Px

plt.plot(data[:,0], data[:,1])
plt.xlabel('Horizontal Displacement')
plt.ylabel('Horizontal Load')
plt.show()

```

1.11.3 Portal Frame 2d Analysis

1. The source code is shown below, which can be downloaded `here`.
2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. Run the source code in your favorate Python program and should see results below

```
Period Comparisons:
 Period       OpenSees        SAP2000   SeismoStruct
      1        1.27321         1.2732         1.2732
      2        0.43128         0.4313         0.4313
      3        0.24204         0.2420         0.2420
      4        0.16018         0.1602         0.1602
      5        0.11899         0.1190         0.1190
      6        0.09506         0.0951         0.0951
      7        0.07951         0.0795         0.0795


 tSatic Analysis Result Comparisons:
                 Parameter       OpenSees        SAP2000   SeismoStruct
                  Disp Top          1.451           1.45           1.45
    Axial Force Bottom Left         69.987          69.99          70.01
         Moment Bottom Left       2324.677        2324.68        2324.71
 PASSED Verification Test PortalFrame2d.py
```

```
1 import sys
2
```

(continues on next page)

(continued from previous page)

```
sys.path.append('path/to/directory/of/pyd/file')

from opensees import *
from math import asin, sqrt

# Two dimensional Frame: Eigenvalue & Static Loads


# REFERENCES:
# used in verification by SAP2000:
# SAP2000 Integrated Finite Element Analysis and Design of Structures, Verification␣
↪Manual,
# Computers and Structures, 1997. Example 1.
# and seismo-struct (Example 10)
# SeismoStruct, Verification Report For Version 6, 2012. Example 11.


# set some properties
wipe()

model('Basic', '-ndm', 2)

# properties

#    units kip, ft

numBay = 2
numFloor = 7

bayWidth = 360.0
storyHeights = [162.0, 162.0, 156.0, 156.0, 156.0, 156.0, 156.0]

E = 29500.0
massX = 0.49
M = 0.
coordTransf = "Linear"  # Linear, PDelta, Corotational
massType = "-lMass"  # -lMass, -cMass

beams = ['W24X160', 'W24X160', 'W24X130', 'W24X130', 'W24X110', 'W24X110', 'W24X110']
eColumn = ['W14X246', 'W14X246', 'W14X246', 'W14X211', 'W14X211', 'W14X176', 'W14X176
↪']
iColumn = ['W14X287', 'W14X287', 'W14X287', 'W14X246', 'W14X246', 'W14X211', 'W14X211
↪']
columns = [eColumn, iColumn, eColumn]

WSection = {
    'W14X176': [51.7, 2150.],
    'W14X211': [62.1, 2670.],
    'W14X246': [72.3, 3230.],
    'W14X287': [84.4, 3910.],
    'W24X110': [32.5, 3330.],
    'W24X130': [38.3, 4020.],
    'W24X160': [47.1, 5120.]
}

nodeTag = 1

```

(continues on next page)

(continued from previous page)

```

# procedure to read
def ElasticBeamColumn(eleTag, iNode, jNode, sectType, E, transfTag, M, massType):
    found = 0

    prop = WSection[sectType]

    A = prop[0]
    I = prop[1]
    element('elasticBeamColumn', eleTag, iNode, jNode, A, E, I, transfTag, '-mass', M,
↪ massType)


# add the nodes
#  - floor at a time
yLoc = 0.
for j in range(0, numFloor + 1):

    xLoc = 0.
    for i in range(0, numBay + 1):
        node(nodeTag, xLoc, yLoc)
        xLoc += bayWidth
        nodeTag += 1

    if j < numFloor:
        storyHeight = storyHeights[j]

    yLoc += storyHeight

# fix first floor
fix(1, 1, 1, 1)
fix(2, 1, 1, 1)
fix(3, 1, 1, 1)

# rigid floor constraint & masses
nodeTagR = 5
nodeTag = 4
for j in range(1, numFloor + 1):
    for i in range(0, numBay + 1):

        if nodeTag != nodeTagR:
            equalDOF(nodeTagR, nodeTag, 1)
        else:
            mass(nodeTagR, massX, 1.0e-10, 1.0e-10)

        nodeTag += 1

    nodeTagR += numBay + 1

# add the columns
# add column element
geomTransf(coordTransf, 1)
eleTag = 1
for j in range(0, numBay + 1):

    end1 = j + 1
    end2 = end1 + numBay + 1
```

(continues on next page)

(continued from previous page)

```
    thisColumn = columns[j]

    for i in range(0, numFloor):
        secType = thisColumn[i]
        ElasticBeamColumn(eleTag, end1, end2, secType, E, 1, M, massType)
        end1 = end2
        end2 += numBay + 1
        eleTag += 1

# add beam elements
for j in range(1, numFloor + 1):
    end1 = (numBay + 1) * j + 1
    end2 = end1 + 1
    secType = beams[j - 1]
    for i in range(0, numBay):
        ElasticBeamColumn(eleTag, end1, end2, secType, E, 1, M, massType)
        end1 = end2
        end2 = end1 + 1
        eleTag += 1

# calculate eigenvalues & print results
numEigen = 7
eigenValues = eigen(numEigen)
PI = 2 * asin(1.0)

#
# apply loads for static analysis & perform analysis
#

timeSeries('Linear', 1)
pattern('Plain', 1, 1)
load(22, 20.0, 0., 0.)
load(19, 15.0, 0., 0.)
load(16, 12.5, 0., 0.)
load(13, 10.0, 0., 0.)
load(10, 7.5, 0., 0.)
load(7, 5.0, 0., 0.)
load(4, 2.5, 0., 0.)

integrator('LoadControl', 1.0)
algorithm('Linear')
analysis('Static')
analyze(1)

# determine PASS/FAILURE of test
ok = 0

#
# print pretty output of comparisons
#

#               SAP2000   SeismoStruct
comparisonResults = [[1.2732, 0.4313, 0.2420, 0.1602, 0.1190, 0.0951, 0.0795],
                     [1.2732, 0.4313, 0.2420, 0.1602, 0.1190, 0.0951, 0.0795]]
print("\n\nPeriod Comparisons:")
print('{:>10}{:>15}{:>15}{:>15}'.format('Period', 'OpenSees', 'SAP2000', 'SeismoStruct
↪'))
```

(continues on next page)

(continued from previous page)

```

# formatString {%10s%15.5f%15.4f%15.4f}
for i in range(0, numEigen):
    lamb = eigenValues[i]
    period = 2 * PI / sqrt(lamb)
    print('{:>10}{:>15.5f}{:>15.4f}{:>15.4f}'.format(i + 1, period,␣
↪comparisonResults[0][i], comparisonResults[1][i]))
    resultOther = comparisonResults[0][i]
    if abs(period - resultOther) > 9.99e-5:
        ok - 1

# print table of comparision
#       Parameter          SAP2000   SeismoStruct
comparisonResults = [["Disp Top", "Axial Force Bottom Left", "Moment Bottom Left"],
                     [1.45076, 69.99, 2324.68],
                     [1.451, 70.01, 2324.71]]
tolerances = [9.99e-6, 9.99e-3, 9.99e-3]

print("\n\nSatic Analysis Result Comparisons:")
print('{:>30}{:>15}{:>15}{:>15}'.format('Parameter', 'OpenSees', 'SAP2000',
↪'SeismoStruct'))
for i in range(3):
    response = eleResponse(1, 'forces')
    if i == 0:
        result = nodeDisp(22, 1)
    elif i == 1:
        result = abs(response[1])
    else:
        result = response[2]

    print('{:>30}{:>15.3f}{:>15.2f}{:>15.2f}'.format(comparisonResults[0][i],
                                                     result,
                                                     comparisonResults[1][i],
                                                     comparisonResults[2][i]))
    resultOther = comparisonResults[1][i]
    tol = tolerances[i]
    if abs(result - resultOther) > tol:
        ok - 1
        print("failed-> ", i, abs(result - resultOther), tol)

if ok == 0:
    print("PASSED Verification Test PortalFrame2d.py \n\n")
else:
    print("FAILED Verification Test PortalFrame2d.py \n\n")

```

1.11.4 Moment Curvature Analysis

1. The source code is shown below, which can be downloaded here.
2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. Run the source code in your favorate Python program and should see results below

```
Start MomentCurvature.py example
Estimated yield curvature:  0.000126984126984127
```

(continues on next page)

(continued from previous page)

```
Passed!
==========================
```

```
import sys
sys.path.append('path')
from opensees import *

def MomentCurvature(secTag, axialLoad, maxK, numIncr=100):

    # Define two nodes at (0,0)
    node(1, 0.0, 0.0)
    node(2, 0.0, 0.0)

    # Fix all degrees of freedom except axial and bending
    fix(1, 1, 1, 1)
    fix(2, 0, 1, 0)

    # Define element
    #                             tag ndI ndJ  secTag
    element('zeroLengthSection',  1,   1,   2,  secTag)

    # Define constant axial load
    timeSeries('Constant', 1)
    pattern('Plain', 1, 1)
    load(2, axialLoad, 0.0, 0.0)

    # Define analysis parameters
    integrator('LoadControl', 0.0)
    system('SparseGeneral', '-piv')
    test('NormUnbalance', 1e-9, 10)
    numberer('Plain')
    constraints('Plain')
    algorithm('Newton')
    analysis('Static')

    # Do one analysis for constant axial load
    analyze(1)

    # Define reference moment
    timeSeries('Linear', 2)
    pattern('Plain',2, 2)
    load(2, 0.0, 0.0, 1.0)

    # Compute curvature increment
    dK = maxK / numIncr

    # Use displacement control at node 2 for section analysis
    integrator('DisplacementControl', 2,3,dK,1,dK,dK)

    # Do the section analysis
    analyze(numIncr)


wipe()
print("Start MomentCurvature.py example")

```

(continues on next page)

(continued from previous page)

```
# Define model builder
# --------------------
model('basic','-ndm',2,'-ndf',3)

# Define materials for nonlinear columns
# ------------------------------------------
# CONCRETE                  tag   f'c        ec0   f'cu        ecu
# Core concrete (confined)
uniaxialMaterial('Concrete01',1, -6.0,  -0.004,  -5.0,  -0.014)

# Cover concrete (unconfined)
uniaxialMaterial('Concrete01',2, -5.0,  -0.002,  0.0,  -0.006)

# STEEL
# Reinforcing steel
fy = 60.0      # Yield stress
E = 30000.0    # Young's modulus

#                        tag  fy E0    b
uniaxialMaterial('Steel01', 3, fy, E, 0.01)

# Define cross-section for nonlinear columns
# ------------------------------------------

# set some paramaters
colWidth = 15
colDepth = 24

cover = 1.5
As = 0.60;     # area of no. 7 bars

# some variables derived from the parameters
y1 = colDepth/2.0
z1 = colWidth/2.0


section('Fiber', 1)

# Create the concrete core fibers
patch('rect',1,10,1 ,cover-y1, cover-z1, y1-cover, z1-cover)

# Create the concrete cover fibers (top, bottom, left, right)
patch('rect',2,10,1 ,-y1, z1-cover, y1, z1)
patch('rect',2,10,1 ,-y1, -z1, y1, cover-z1)
patch('rect',2,2,1 ,-y1, cover-z1, cover-y1, z1-cover)
patch('rect',2,2,1 ,y1-cover, cover-z1, y1, z1-cover)

# Create the reinforcing fibers (left, middle, right)
layer('straight', 3, 3, As, y1-cover, z1-cover, y1-cover, cover-z1)
layer('straight', 3, 2, As, 0.0     , z1-cover, 0.0      , cover-z1)
layer('straight', 3, 3, As, cover-y1, z1-cover, cover-y1, cover-z1)

# Estimate yield curvature
# (Assuming no axial load and only top and bottom steel)
# d -- from cover to rebar
d = colDepth-cover
# steel yield strain
```

(continues on next page)

(continued from previous page)

```
epsy = fy/E
Ky = epsy/(0.7*d)

# Print estimate to standard output
print("Estimated yield curvature: ", Ky)

# Set axial load
P = -180.0

# Target ductility for analysis
mu = 15.0

# Number of analysis increments
numIncr = 100

# Call the section analysis procedure
MomentCurvature(1, P, Ky*mu, numIncr)

results = open('results.out','a+')

u = nodeDisp(2,3)
if abs(u-0.00190476190476190541)<1e-12:
    results.write('PASSED : MomentCurvature.py\n');
    print("Passed!")
else:
    results.write('FAILED : MomentCurvature.py\n');
    print("Failed!")

results.close()

print("==========================")
```

1.11.5 Reinforced Concrete Frame Gravity Analysis

1. The source code is shown below, which can be downloaded `here`.
2. Change the line 4 below to set the right path where the OpenSeesPy library located.
3. Run the source code in your favorate Python program and should see `Passed!` in the results.

```
1  print("==========================")
2
3  import sys
4
5  sys.path.append('/scratch/opensees/SRC/interpreter')
6  from opensees import *
7
8  print("Starting RCFrameGravity example")
9
10 # Create ModelBuilder (with two-dimensions and 3 DOF/node)
11 model('basic', '-ndm', 2, '-ndf', 3)
12
13 # Create nodes
14 # ------------
15
16 # Set parameters for overall model geometry
```

(continues on next page)

(continued from previous page)

```
width = 360.0
height = 144.0

# Create nodes
#    tag, X, Y
node(1, 0.0, 0.0)
node(2, width, 0.0)
node(3, 0.0, height)
node(4, width, height)

# Fix supports at base of columns
#   tag, DX, DY, RZ
fix(1, 1, 1, 1)
fix(2, 1, 1, 1)

# Define materials for nonlinear columns
# ------------------------------------------
# CONCRETE                   tag  f'c    ec0    f'cu   ecu
# Core concrete (confined)
uniaxialMaterial('Concrete01', 1, -6.0, -0.004, -5.0, -0.014)

# Cover concrete (unconfined)
uniaxialMaterial('Concrete01', 2, -5.0, -0.002, 0.0, -0.006)

# STEEL
# Reinforcing steel
fy = 60.0;  # Yield stress
E = 30000.0;  # Young's modulus
#                          tag  fy E0    b
uniaxialMaterial('Steel01', 3, fy, E, 0.01)

# Define cross-section for nonlinear columns
# ------------------------------------------

#  some parameters
colWidth = 15
colDepth = 24

cover = 1.5
As = 0.60  # area of no. 7 bars

# some variables derived from the parameters
y1 = colDepth / 2.0
z1 = colWidth / 2.0

section('Fiber', 1)

# Create the concrete core fibers
patch('rect', 1, 10, 1, cover - y1, cover - z1, y1 - cover, z1 - cover)

# Create the concrete cover fibers (top, bottom, left, right)
patch('rect', 2, 10, 1, -y1, z1 - cover, y1, z1)
patch('rect', 2, 10, 1, -y1, -z1, y1, cover - z1)
patch('rect', 2, 2, 1, -y1, cover - z1, cover - y1, z1 - cover)
patch('rect', 2, 2, 1, y1 - cover, cover - z1, y1, z1 - cover)

# Create the reinforcing fibers (left, middle, right)
```

(continues on next page)

(continued from previous page)

```
layer('straight', 3, 3, As, y1 - cover, z1 - cover, y1 - cover, cover - z1)
layer('straight', 3, 2, As, 0.0, z1 - cover, 0.0, cover - z1)
layer('straight', 3, 3, As, cover - y1, z1 - cover, cover - y1, cover - z1)

# Define column elements
# ----------------------

# Geometry of column elements
#                tag

geomTransf('PDelta', 1)

# Number of integration points along length of element
np = 5

# Lobatto integratoin
beamIntegration('Lobatto', 1, 1, np)

# Create the coulumns using Beam-column elements
#               e            tag ndI ndJ transfTag integrationTag
eleType = 'forceBeamColumn'
element(eleType, 1, 1, 3, 1, 1)
element(eleType, 2, 2, 4, 1, 1)

# Define beam elment
# -----------------------------

# Geometry of column elements
#                tag
geomTransf('Linear', 2)

# Create the beam element
#                          tag, ndI, ndJ, A,     E,    Iz, transfTag
element('elasticBeamColumn', 3, 3, 4, 360.0, 4030.0, 8640.0, 2)

# Define gravity loads
# --------------------

#  a parameter for the axial load
P = 180.0;  # 10% of axial capacity of columns

# Create a Plain load pattern with a Linear TimeSeries
timeSeries('Linear', 1)
pattern('Plain', 1, 1)

# Create nodal loads at nodes 3 & 4
#    nd  FX,  FY, MZ
load(3, 0.0, -P, 0.0)
load(4, 0.0, -P, 0.0)

# ------------------------------
# End of model generation
# ------------------------------


# ------------------------------
# Start of analysis generation
```

(continues on next page)

(continued from previous page)

```
# ------------------------------

# Create the system of equation, a sparse solver with partial pivoting
system('BandGeneral')

# Create the constraint handler, the transformation method
constraints('Transformation')

# Create the DOF numberer, the reverse Cuthill-McKee algorithm
numberer('RCM')

# Create the convergence test, the norm of the residual with a tolerance of
# 1e-12 and a max number of iterations of 10
test('NormDispIncr', 1.0e-12, 10, 3)

# Create the solution algorithm, a Newton-Raphson algorithm
algorithm('Newton')

# Create the integration scheme, the LoadControl scheme using steps of 0.1
integrator('LoadControl', 0.1)

# Create the analysis object
analysis('Static')

# ------------------------------
# End of analysis generation
# ------------------------------


# ------------------------------
# Finally perform the analysis
# ------------------------------

# perform the gravity load analysis, requires 10 steps to reach the load level
analyze(10)

# Print out the state of nodes 3 and 4
# print node 3 4

# Print out the state of element 1
# print ele 1

u3 = nodeDisp(3, 2)
u4 = nodeDisp(4, 2)

results = open('results.out', 'a+')

if abs(u3 + 0.0183736) < 1e-6 and abs(u4 + 0.0183736) < 1e-6:
    results.write('PASSED : RCFrameGravity.py\n')
    print("Passed!")
else:
    results.write('FAILED : RCFrameGravity.py\n')
    print("Failed!")

results.close()

print("==========================")
```

1.11.6 Reinforced Concrete Frame Pushover Analysis

1. The source code is shown below, which can be downloaded `here`.
2. The file for gravity analysis is also needed :`here`.
3. Change the line 9 below to set the right path where the OpenSeesPy library located.
4. Run the source code in your favorate Python program and should see `Passed!` in the results.

```
1 print("==========================")
2 print("Start RCFramePushover Example")
3
4 # Units: kips, in, sec
5 #
6 # Written: GLF/MHS/fmk
7 # Date: January 2001
8 import sys
9
10 sys.path.append('/scratch/opensees/SRC/interpreter')
11 from opensees import *
12
13 wipe()
14 # ----------------------------------------------------
15 # Start of Model Generation & Initial Gravity Analysis
16 # ----------------------------------------------------
17
18 # Do operations of Example3.1 by sourcing in the tcl file
19 exec(open('RCFrameGravity.py').read())
20 print("Gravity Analysis Completed")
21
22 # Set the gravity loads to be constant & reset the time in the domain
23 loadConst('-time', 0.0)
24
25 # ----------------------------------------------------
26 # End of Model Generation & Initial Gravity Analysis
27 # ----------------------------------------------------
28
29
30 # ----------------------------------------------------
31 # Start of additional modelling for lateral loads
32 # ----------------------------------------------------
33
34 # Define lateral loads
35 # -------------------
36
37 # Set some parameters
38 H = 10.0  # Reference lateral load
39
40 # Set lateral load pattern with a Linear TimeSeries
41 pattern('Plain', 2, 1)
42
43 # Create nodal loads at nodes 3 & 4
44 #    nd    FX  FY  MZ
45 load(3, H, 0.0, 0.0)
46 load(4, H, 0.0, 0.0)
47
48 # ----------------------------------------------------
49 # End of additional modelling for lateral loads
```

(continues on next page)

(continued from previous page)

```python
# ----------------------------------------------------


# ----------------------------------------------------
# Start of modifications to analysis for push over
# ----------------------------------------------------

# Set some parameters
dU = 0.1  # Displacement increment

# Change the integration scheme to be displacement control
#                             node dof init Jd min max
integrator('DisplacementControl', 3, 1, dU, 1, dU, dU)

# ----------------------------------------------------
# End of modifications to analysis for push over
# ----------------------------------------------------


# ------------------------------
# Start of recorder generation
# ------------------------------

# Stop the old recorders by destroying them
# remove recorders

# Create a recorder to monitor nodal displacements
# recorder Node -file node32.out -time -node 3 4 -dof 1 2 3 disp

# Create a recorder to monitor element forces in columns
# recorder EnvelopeElement -file ele32.out -time -ele 1 2 forces

# --------------------------------
# End of recorder generation
# ---------------------------------


# ------------------------------
# Finally perform the analysis
# ------------------------------

# Set some parameters
maxU = 15.0  # Max displacement
currentDisp = 0.0
ok = 0

test('NormDispIncr', 1.0e-12, 1000)
algorithm('ModifiedNewton', '-initial')

while ok == 0 and currentDisp < maxU:

    ok = analyze(1)

    # if the analysis fails try initial tangent iteration
    if ok != 0:
        print("modified newton failed")
        break
```

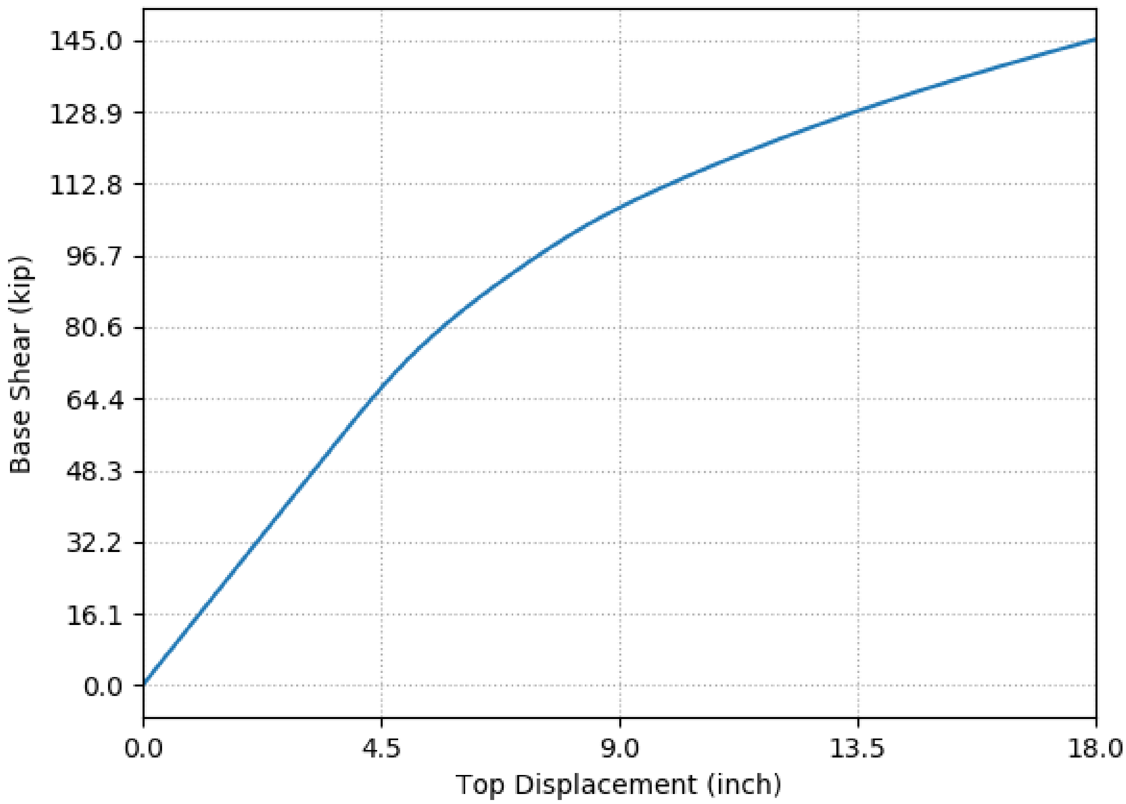
(continues on next page)

(continued from previous page)

```
    # print "regular newton failed .. lets try an initail stiffness for this step"
    # test('NormDispIncr', 1.0e-12,  1000)
    # # algorithm('ModifiedNewton', '-initial')
    # ok = analyze(1)
    # if ok == 0:
    #     print "that worked .. back to regular newton"

    # test('NormDispIncr', 1.0e-12,  10)
    # algorithm('Newton')

    currentDisp = nodeDisp(3, 1)

results = open('results.out', 'a+')

if ok == 0:
    results.write('PASSED : RCFramePushover.py\n')
    print("Passed!")
else:
    results.write('FAILED : RCFramePushover.py\n')
    print("Failed!")

results.close()

# Print the state at node 3
# print node 3


print("==========================")
```

1.11.7 Three story steel building with rigid beam-column connections and W-section

1. The source code is developed by Anurag Upadhyay from University of Utah.
2. The source code is shown below, which can be downloaded here.
3. Change the line 17 below to set the right path where the OpenSeesPy library located.
4. Run the source code in your favorate Python program and should see following plot.




```

##################################################################
## 2D steel frame example.
## 3 story steel building with rigid beam-column connections.
## This script uses W-section command inOpensees to create steel..
## .. beam-column fiber sections.
##
## By - Anurag Upadhyay, PhD Student, University of Utah.
## Date - 08/06/2018
##################################################################

print("=========================================================")
print("Start 2D Steel Frame Example")

import sys
#sys.path.append('C:/OpenSeesPy')                              # for Windows Computer
sys.path.append('/home/anurag/OpenSeesPy')                # For linux Computer
from opensees import *

import numpy as np
import matplotlib.pyplot as plt
import os

AnalysisType='Pushover'         ;                #  Pushover  Gravity
```
(continues on next page)

(continued from previous page)

```

## -------------------------------
## Start of model generation
## -----------------------------
# remove existing model
wipe()

# set modelbuilder
model('basic', '-ndm', 2, '-ndf', 3)

import math

############################################
### Units and Constants  ###################
############################################

inch = 1;
kip = 1;
sec = 1;

# Dependent units
sq_in = inch*inch;
ksi = kip/sq_in;
ft = 12*inch;

# Constants
g = 386.2*inch/(sec*sec);
pi = math.acos(-1);

#######################################
##### Dimensions
#######################################

# Dimensions Input
H_story=10.0*ft;
W_bayX=16.0*ft;
W_bayY_ab=5.0*ft+10.0*inch;
W_bayY_bc=8.0*ft+4.0*inch;
W_bayY_cd=5.0*ft+10.0*inch;

# Calculated dimensions
W_structure=W_bayY_ab+W_bayY_bc+W_bayY_cd;

################
### Material
################

# Steel02 Material

matTag=1;
matConnAx=2;
matConnRot=3;

Fy=60.0*ksi;                    # Yield stress
Es=29000.0*ksi;                    # Modulus of Elasticity of Steel
v=0.2;                                 # Poisson's ratio
Gs=Es/(1+v);                    # Shear modulus
```

(continues on next page)

(continued from previous page)

```
b=0.10;                                    # Strain hardening ratio
params=[18.0,0.925,0.15]                    # R0,cR1,cR2
R0=18.0
cR1=0.925
cR2=0.15
a1=0.05
a2=1.00
a3=0.05
a4=1.0
sigInit=0.0
alpha=0.05

uniaxialMaterial('Steel02', matTag, Fy, Es, b, R0, cR1, cR2, a1, a2, a3, a4, sigInit)

# #################
# ## Sections
# #################

colSecTag1=1;
colSecTag2=2;
beamSecTag1=3;
beamSecTag2=4;
beamSecTag3=5;

# COMMAND: section('WFSection2d', secTag, matTag, d, tw, bf, tf, Nfw, Nff)

section('WFSection2d', colSecTag1, matTag, 10.5*inch, 0.26*inch, 5.77*inch, 0.44*inch,
↪ 15, 16)                    # outer Column
section('WFSection2d', colSecTag2, matTag, 10.5*inch, 0.26*inch, 5.77*inch, 0.44*inch,
↪ 15, 16)                    # Inner Column

section('WFSection2d', beamSecTag1, matTag, 8.3*inch, 0.44*inch, 8.11*inch, 0.
↪685*inch, 15, 15)                    # outer Beam
section('WFSection2d', beamSecTag2, matTag, 8.2*inch, 0.40*inch, 8.01*inch, 0.
↪650*inch, 15, 15)                    # Inner Beam
section('WFSection2d', beamSecTag3, matTag, 8.0*inch, 0.40*inch, 7.89*inch, 0.
↪600*inch, 15, 15)                    # Inner Beam

# Beam size - W10x26
Abeam=7.61*inch*inch;
IbeamY=144.*(inch**4);                          # Inertia along horizontal axis
IbeamZ=14.1*(inch**4);                          # inertia along vertical axis

# BRB input data
Acore=2.25*inch;
Aend=10.0*inch;
LR_BRB=0.55;

# ###########################
# ##### Nodes
# ###########################

# Create All main nodes
node(1, 0.0, 0.0)
node(2, W_bayX, 0.0)
node(3, 2*W_bayX, 0.0)

```

(continues on next page)

(continued from previous page)

```
node(11, 0.0, H_story)
node(12, W_bayX, H_story)
node(13, 2*W_bayX, H_story)

node(21, 0.0, 2*H_story)
node(22, W_bayX, 2*H_story)
node(23, 2*W_bayX, 2*H_story)

node(31, 0.0, 3*H_story)
node(32, W_bayX, 3*H_story)
node(33, 2*W_bayX, 3*H_story)

# Beam Connection nodes

node(1101, 0.0, H_story)
node(1201, W_bayX, H_story)
node(1202, W_bayX, H_story)
node(1301, 2*W_bayX, H_story)

node(2101, 0.0, 2*H_story)
node(2201, W_bayX, 2*H_story)
node(2202, W_bayX, 2*H_story)
node(2301, 2*W_bayX, 2*H_story)

node(3101, 0.0, 3*H_story)
node(3201, W_bayX, 3*H_story)
node(3202, W_bayX, 3*H_story)
node(3301, 2*W_bayX, 3*H_story)

# ###############
#  Constraints
# ###############

fix(1, 1, 1, 1)
fix(2, 1, 1, 1)
fix(3, 1, 1, 1)

# #######################
# ### Elements
# #######################

# ### Assign beam-integration tags

ColIntTag1=1;
ColIntTag2=2;
BeamIntTag1=3;
BeamIntTag2=4;
BeamIntTag3=5;

beamIntegration('Lobatto', ColIntTag1, colSecTag1, 4)
beamIntegration('Lobatto', ColIntTag2, colSecTag2, 4)
beamIntegration('Lobatto', BeamIntTag1, beamSecTag1, 4)
beamIntegration('Lobatto', BeamIntTag2, beamSecTag2, 4)
beamIntegration('Lobatto', BeamIntTag3, beamSecTag3, 4)

# Assign geometric transformation

```

(continues on next page)

(continued from previous page)

```
ColTransfTag=1
BeamTranfTag=2

geomTransf('PDelta', ColTransfTag)
geomTransf('Linear', BeamTranfTag)


# Assign Elements  ##############

# ## Add non-linear column elements
element('forceBeamColumn', 1, 1, 11, ColTransfTag, ColIntTag1, '-mass', 0.0)
element('forceBeamColumn', 2, 2, 12, ColTransfTag, ColIntTag2, '-mass', 0.0)
element('forceBeamColumn', 3, 3, 13, ColTransfTag, ColIntTag1, '-mass', 0.0)

element('forceBeamColumn', 11, 11, 21, ColTransfTag, ColIntTag1, '-mass', 0.0)
element('forceBeamColumn', 12, 12, 22, ColTransfTag, ColIntTag2, '-mass', 0.0)
element('forceBeamColumn', 13, 13, 23, ColTransfTag, ColIntTag1, '-mass', 0.0)

element('forceBeamColumn', 21, 21, 31, ColTransfTag, ColIntTag1, '-mass', 0.0)
element('forceBeamColumn', 22, 22, 32, ColTransfTag, ColIntTag2, '-mass', 0.0)
element('forceBeamColumn', 23, 23, 33, ColTransfTag, ColIntTag1, '-mass', 0.0)

#

#  ### Add linear main beam elements, along x-axis
#element('elasticBeamColumn', 101, 1101, 1201, Abeam, Es, Gs, Jbeam, IbeamY, IbeamZ,␣
↪beamTransfTag, '-mass', 0.0)

element('forceBeamColumn', 101, 1101, 1201, BeamTranfTag, BeamIntTag1, '-mass', 0.0)
element('forceBeamColumn', 102, 1202, 1301, BeamTranfTag, BeamIntTag1, '-mass', 0.0)

element('forceBeamColumn', 201, 2101, 2201, BeamTranfTag, BeamIntTag2, '-mass', 0.0)
element('forceBeamColumn', 202, 2202, 2301, BeamTranfTag, BeamIntTag2, '-mass', 0.0)

element('forceBeamColumn', 301, 3101, 3201, BeamTranfTag, BeamIntTag3, '-mass', 0.0)
element('forceBeamColumn', 302, 3202, 3301, BeamTranfTag, BeamIntTag3, '-mass', 0.0)

# Assign constraints between beam end nodes and column nodes (RIgid beam column␣
↪connections)
equalDOF(11, 1101, 1,2,3)
equalDOF(12, 1201, 1,2,3)
equalDOF(12, 1202, 1,2,3)
equalDOF(13, 1301, 1,2,3)

equalDOF(21, 2101, 1,2,3)
equalDOF(22, 2201, 1,2,3)
equalDOF(22, 2202, 1,2,3)
equalDOF(23, 2301, 1,2,3)

equalDOF(31, 3101, 1,2,3)
equalDOF(32, 3201, 1,2,3)
equalDOF(32, 3202, 1,2,3)
equalDOF(33, 3301, 1,2,3)


###############
## Gravity Load
```

(continues on next page)

(continued from previous page)

```
################
# create TimeSeries
timeSeries("Linear", 1)

# create a plain load pattern
pattern("Plain", 1, 1)

# Create the nodal load
load(11, 0.0, -5.0*kip, 0.0)
load(12, 0.0, -6.0*kip, 0.0)
load(13, 0.0, -5.0*kip, 0.0)

load(21, 0., -5.*kip, 0.0)
load(22, 0., -6.*kip,0.0)
load(23, 0., -5.*kip, 0.0)

load(31, 0., -5.*kip, 0.0)
load(32, 0., -6.*kip, 0.0)
load(33, 0., -5.*kip, 0.0)


# ------------------------------
# Start of analysis generation
# ------------------------------

NstepsGrav = 10

system("BandGEN")
numberer("Plain")
constraints("Plain")
integrator("LoadControl", 1.0/NstepsGrav)
algorithm("Newton")
test('NormUnbalance',1e-8, 10)
analysis("Static")


# perform the analysis
data = np.zeros((NstepsGrav+1,2))
for j in range(NstepsGrav):
    analyze(1)
    data[j+1,0] = nodeDisp(31,2)
    data[j+1,1] = getLoadFactor(1)*5

loadConst('-time', 0.0)

print("Gravity analysis complete")

wipeAnalysis()

###############################
### PUSHOVER ANALYSIS
###############################

if(AnalysisType=="Pushover"):

        print("<<<< Running Pushover Analysis >>>>")

```

(continues on next page)

(continued from previous page)

```
        # Create load pattern for pushover analysis
        # create a plain load pattern
        pattern("Plain", 2, 1)

        load(11, 1.61, 0.0, 0.0)
        load(21, 3.22, 0.0, 0.0)
        load(31, 4.83, 0.0, 0.0)

        ControlNode=31
        ControlDOF=1
        MaxDisp=0.15*H_story
        DispIncr=0.1
        NstepsPush=int(MaxDisp/DispIncr)

        system("ProfileSPD")
        numberer("Plain")
        constraints("Plain")
        integrator("DisplacementControl", ControlNode, ControlDOF, DispIncr)
        algorithm("Newton")
        test('NormUnbalance',1e-8, 10)
        analysis("Static")

        PushDataDir = r'PushoverOut'
        if not os.path.exists(PushDataDir):
                os.makedirs(PushDataDir)
        recorder('Node', '-file', "PushoverOut/Node2React.out", '-closeOnWrite', '-node', 2, '-dof',1, 'reaction')
        recorder('Node', '-file', "PushoverOut/Node31Disp.out", '-closeOnWrite', '-node', 31, '-dof',1, 'disp')
        recorder('Element', '-file', "PushoverOut/BeamStress.out", '-closeOnWrite', '-ele', 102, 'section', '4', 'fiber','1', 'stressStrain')

        # analyze(NstepsPush)

        # Perform pushover analysis
        dataPush = np.zeros((NstepsPush+1,5))
        for j in range(NstepsPush):
                analyze(1)
                dataPush[j+1,0] = nodeDisp(31,1)
                reactions()
                dataPush[j+1,1] = nodeReaction(1, 1) + nodeReaction(2, 1) + nodeReaction(3, 1)

        plt.plot(dataPush[:,0], -dataPush[:,1])
        plt.xlim(0, MaxDisp)
        plt.xticks(np.linspace(0,MaxDisp,5,endpoint=True))
        plt.yticks(np.linspace(0, -int(dataPush[NstepsPush,1]),10,endpoint=True))
        plt.grid(linestyle='dotted')
        plt.xlabel('Top Displacement (inch)')
        plt.ylabel('Base Shear (kip)')
        plt.show()


        print("Pushover analysis complete")



```

(continues on next page)

(continued from previous page)

```

```

1.12 Earthquake Examples

1.12.1 Cantilever 2D EQ ground motion with gravity Analysis

1. The source code is shown below, which can be downloaded `here`.
2. The ground motion data file `here` must be put in the same folder.
3. Change the line 5 below to set the right path where the OpenSeesPy library located.
4. Run the source code in your favorate Python program and should see results below

```
=========================================================
Start cantilever 2D EQ ground motion with gravity example
u2 =  -0.07441860465116278
Passed!
=========================================
```

```python
1  print("=========================================================")
2  print("Start cantilever 2D EQ ground motion with gravity example")
3  
4  import sys
5  sys.path.append('/path/to/direction/of/pyd/file')
6  from opensees import *
7  
8  
9  # -----------------------------------------------------------------------------------------------------
10 # Example 1. cantilever 2D
11 # EQ ground motion with gravity
12 # all units are in kip, inch, second
13 # elasticBeamColumn ELEMENT
14 #               Silvia Mazzoni & Frank McKenna, 2006
15 #
16 #    ^Y
17 #    |
18 #    2       __
19 #    |         |
20 #    |         |
21 #    |         |
22 #   (1)      36'
23 #    |         |
24 #    |         |
25 #    |         |
26 #  =1=    ----  -------->X
27 #
28 
29 # SET UP ----------------------------------------------------------------------------
30 wipe()                                              # clear opensees model
31 model('basic', '-ndm', 2, '-ndf', 3)               # 2 dimensions, 3 dof per node
32 # file mkdir data                                     # create data directory
33 
```

(continues on next page)

(continued from previous page)

```
# define GEOMETRY -------------------------------------------------------------
# nodal coordinates:
node(1, 0., 0.)                                         # node#, X Y
node(2, 0., 432.)

# Single point constraints -- Boundary Conditions
fix(1, 1, 1, 1)                                  # node DX DY RZ

# nodal masses:
mass(2, 5.18, 0., 0.)                         # node#, Mx My Mz, Mass=Weight/g.

# Define ELEMENTS -------------------------------------------------------------
# define geometric transformation: performs a linear geometric transformation of beam␣
↪stiffness and resisting force from the basic system to the global-coordinate system
geomTransf('Linear', 1)                       # associate a tag to transformation

# connectivity:
element('elasticBeamColumn', 1, 1, 2, 3600.0, 3225.0,1080000.0, 1)

# define GRAVITY -------------------------------------------------------------
timeSeries('Linear', 1)
pattern('Plain', 1, 1,)
load(2, 0., -2000., 0.)                          # node#, FX FY MZ -- ␣
↪superstructure-weight

constraints('Plain')                                # how it handles boundary␣
↪conditions
numberer('Plain')                          # renumber dof's to minimize band-width␣
↪(optimization), if you want to
system('BandGeneral')                   # how to store and solve the system of␣
↪equations in the analysis
algorithm('Linear')                # use Linear algorithm for linear analysis
integrator('LoadControl', 0.1)                      # determine the next time step␣
↪for an analysis, # apply gravity in 10 steps
analysis('Static')                                          # define type of␣
↪analysis static or transient
analyze(10)                                              # perform gravity analysis
loadConst('-time', 0.0)                              # hold gravity constant and␣
↪restart time

# DYNAMIC ground-motion analysis -----------------------------------------------------
↪--------
# create load pattern
G = 386.0
timeSeries('Path', 2, '-dt', 0.005, '-filePath', 'A10000.dat', '-factor', G) # define␣
↪acceleration vector from file (dt=0.005 is associated with the input file gm)
pattern('UniformExcitation', 2, 1, '-accel', 2)                      # define␣
↪where and how (pattern tag, dof) acceleration is applied

# set damping based on first eigen mode
freq = eigen('-fullGenLapack', 1)**0.5
dampRatio = 0.02
rayleigh(0., 0., 0., 2*dampRatio/freq)

# create the analysis
wipeAnalysis()                          # clear previously-define analysis␣
↪parameters
```

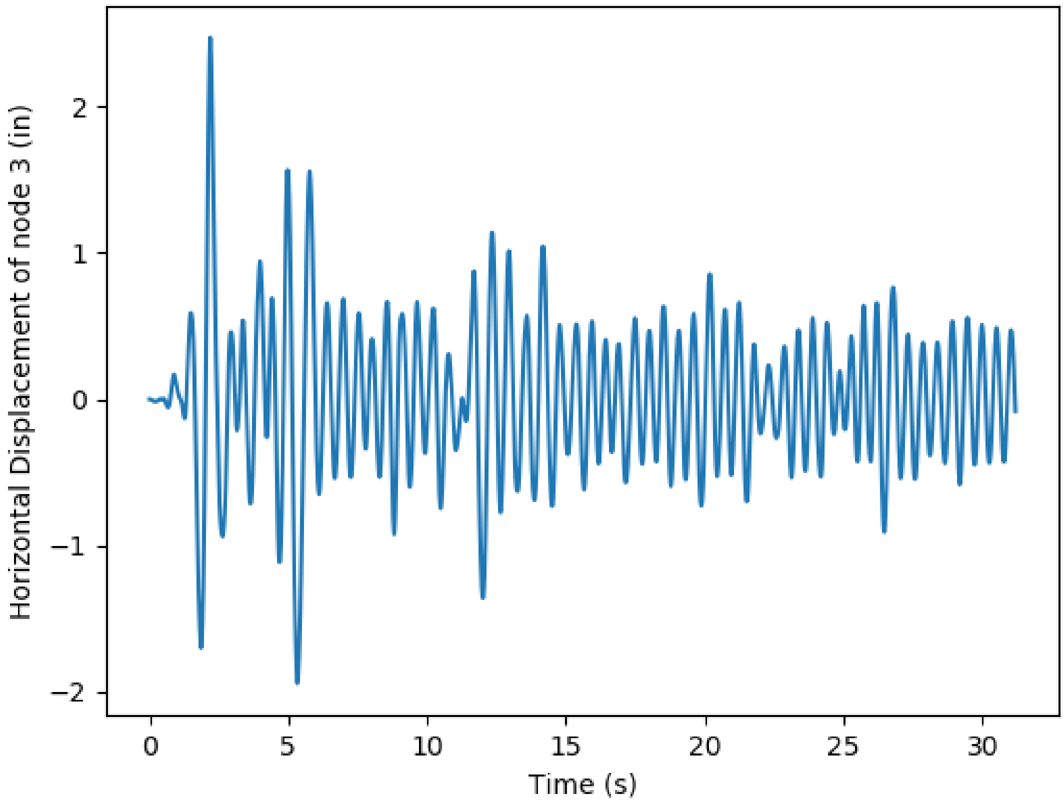
(continues on next page)

(continued from previous page)

```
constraints('Plain')              # how it handles boundary conditions
numberer('Plain')    # renumber dof's to minimize band-width (optimization), if you want to
system('BandGeneral') # how to store and solve the system of equations in the analysis
algorithm('Linear')          # use Linear algorithm for linear analysis
integrator('Newmark', 0.5, 0.25)    # determine the next time step for an analysis
analysis('Transient')   # define type of analysis: time-dependent
analyze(3995, 0.01)          # apply 3995 0.01-sec time steps in analysis

u2 = nodeDisp(2, 2)
print("u2 = ", u2)


if abs(u2+0.07441860465116277579) < 1e-12:
    print("Passed!")
else:
    print("Failed!")

wipe()

print("=========================================")
```

1.12.2 Reinforced Concrete Frame Earthquake Analysis

1. The source code is shown below, which can be downloaded `here`.
2. The file for gravity analysis is also needed `here`.
3. The ReadRecord is a useful python function for parsing the PEER strong motion data base files and returning the `dt`, `nPts` and creating a file containing just data points. The function is kept in a seperate file `here` and is imported in the example.
4. The ground motion data file `here` must be put in the same folder.
5. Change the line 9 below to set the right path where the OpenSeesPy library located.
6. Run the source code in your favorate Python program and should see `Passed!` in the results and a plotting of displacement for node 3




```python
print("=========================")
print("Start RCFrameEarthquake Example")

# Units: kips, in, sec
#
# Written: Minjie

import sys
sys.path.append('/scratch/opensees/SRC/interpreter')
from opensees import *

import ReadRecord
import numpy as np
import matplotlib.pyplot as plt

wipe()
# ----------------------------------------------------
# Start of Model Generation & Initial Gravity Analysis
# ----------------------------------------------------

# Do operations of Example3.1 by sourcing in the tcl file
exec(open('RCFrameGravity.py').read())
print("Gravity Analysis Completed")

# Set the gravity loads to be constant & reset the time in the domain
```

(continues on next page)

(continued from previous page)

```
loadConst('-time', 0.0)

# ----------------------------------------------------
# End of Model Generation & Initial Gravity Analysis
# ----------------------------------------------------

# Define nodal mass in terms of axial load on columns
g = 386.4
m = P/g

mass(3, m, m, 0.0)
mass(4, m, m, 0.0)

# Set some parameters
record = 'elCentro'

# Permform the conversion from SMD record to OpenSees record
dt, nPts = ReadRecord.ReadRecord(record+'.at2', record+'.dat')

# Set time series to be passed to uniform excitation
timeSeries('Path', 2, '-filePath', record+'.dat', '-dt', dt, '-factor', g)

# Create UniformExcitation load pattern
#                         tag dir
pattern('UniformExcitation',  2,   1,  '-accel', 2)

# set the rayleigh damping factors for nodes & elements
rayleigh(0.0, 0.0, 0.0, 0.000625)

# Delete the old analysis and all it's component objects
wipeAnalysis()

# Create the system of equation, a banded general storage scheme
system('BandGeneral')

# Create the constraint handler, a plain handler as homogeneous boundary
constraints('Plain')

# Create the convergence test, the norm of the residual with a tolerance of
# 1e-12 and a max number of iterations of 10
test('NormDispIncr', 1.0e-12,  10 )

# Create the solution algorithm, a Newton-Raphson algorithm
algorithm('Newton')

# Create the DOF numberer, the reverse Cuthill-McKee algorithm
numberer('RCM')

# Create the integration scheme, the Newmark with alpha =0.5 and beta =.25
integrator('Newmark',  0.5,  0.25 )

# Create the analysis object
analysis('Transient')

# Perform an eigenvalue analysis
numEigen = 2
eigenValues = eigen(numEigen)
```

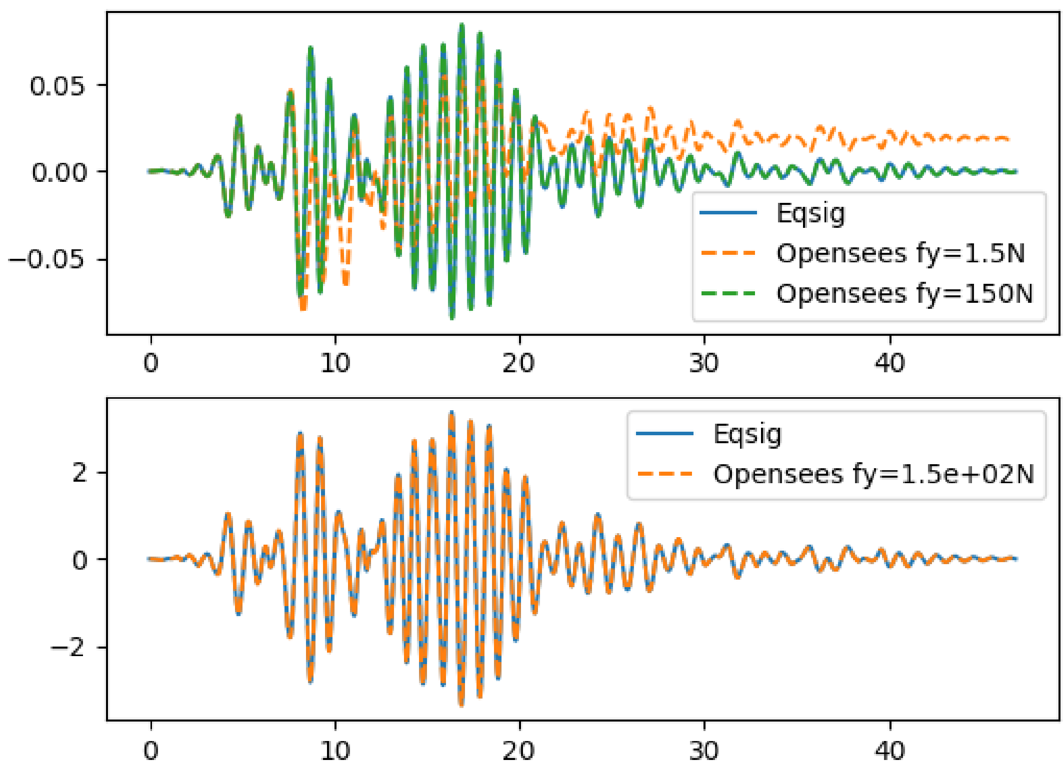
(continues on next page)

(continued from previous page)

```
print("eigen values at start of transient:",eigenValues)

# set some variables
tFinal = nPts*dt
tCurrent = getTime()
ok = 0

time = [tCurrent]
u3 = [0.0]

# Perform the transient analysis
while ok == 0 and tCurrent < tFinal:

    ok = analyze(1, .01)

    # if the analysis fails try initial tangent iteration
    if ok != 0:
        print("regular newton failed .. lets try an initail stiffness for this step")
        test('NormDispIncr', 1.0e-12,  100, 0)
        algorithm('ModifiedNewton', '-initial')
        ok =analyze( 1, .01)
        if ok == 0:
            print("that worked .. back to regular newton")
        test('NormDispIncr', 1.0e-12,  10 )
        algorithm('Newton')

    tCurrent = getTime()

    time.append(tCurrent)
    u3.append(nodeDisp(3,1))



# Perform an eigenvalue analysis
eigenValues = eigen(numEigen)
print("eigen values at end of transient:",eigenValues)

results = open('results.out','a+')

if ok == 0:
    results.write('PASSED : RCFrameEarthquake.py\n');
    print("Passed!")
else:
    results.write('FAILED : RCFrameEarthquake.py\n');
    print("Failed!")

results.close()

plt.plot(time, u3)
plt.ylabel('Horizontal Displacement of node 3 (in)')
plt.xlabel('Time (s)')

plt.show()



print("==========================")
```

1.12.3 Example name spaced nonlinear SDOF

1. The source code is developed by Maxim Millen from University of Porto.
2. The source code is shown below, which can be downloaded `here`.
3. Also download the constants file `here`, and the `ground motion file`
4. Change the line 7 below to set the right path where the OpenSeesPy library located.
5. Make sure the numpy, matplotlib and eqsig packages are installed in your Python distribution.
6. Run the source code in your favorite Python program and should see




```python
1  import eqsig
2  from eqsig import duhamels
3  import matplotlib.pyplot as plt
4  import numpy as np
5  
6  import sys
7  sys.path.append('/scratch/opensees/SRC/interpreter')
8  
9  import opensees as op  # change this to the path where opensees python is stored
10 import opensees_constants as opc
11 
12 
13 def get_inelastic_response(mass, k_spring, f_yield, motion, dt, xi=0.05, r_post=0.0):
```

(continues on next page)

(continued from previous page)

```python
    """
    Run seismic analysis of a nonlinear SDOF

    :param mass: SDOF mass
    :param k_spring: spring stiffness
    :param f_yield: yield strength
    :param motion: list, acceleration values
    :param dt: float, time step of acceleration values
    :param xi: damping ratio
    :param r_post: post-yield stiffness
    :return:
    """

    op.wipe()
    op.model('basic', '-ndm', 2, '-ndf', 3)  # 2 dimensions, 3 dof per node

    # Establish nodes
    bot_node = 1
    top_node = 2
    op.node(bot_node, 0., 0.)
    op.node(top_node, 0., 0.)

    # Fix bottom node
    op.fix(top_node, opc.FREE, opc.FIXED, opc.FIXED)
    op.fix(bot_node, opc.FIXED, opc.FIXED, opc.FIXED)
    # Set out-of-plane DOFs to be slaved
    op.equalDOF(1, 2, *[2, 3])

    # nodal mass (weight / g):
    op.mass(top_node, mass, 0., 0.)

    # Define material
    bilinear_mat_tag = 1
    mat_type = "Steel01"
    mat_props = [f_yield, k_spring, r_post]
    op.uniaxialMaterial(mat_type, bilinear_mat_tag, *mat_props)

    # Assign zero length element
    beam_tag = 1
    op.element('zeroLength', beam_tag, bot_node, top_node, "-mat", bilinear_mat_tag,
↪"-dir", 1, '-doRayleigh', 1)

    # Define the dynamic analysis
    load_tag_dynamic = 1
    pattern_tag_dynamic = 1

    values = list(-1 * motion)  # should be negative
    op.timeSeries('Path', load_tag_dynamic, '-dt', dt, '-values', *values)
    op.pattern('UniformExcitation', pattern_tag_dynamic, opc.X, '-accel', load_tag_
↪dynamic)

    # set damping based on first eigen mode
    angular_freq = op.eigen('-fullGenLapack', 1) ** 0.5
    alpha_m = 0.0
    beta_k = 2 * xi / angular_freq
    beta_k_comm = 0.0
    beta_k_init = 0.0
```

(continues on next page)

(continued from previous page)

```python

    op.rayleigh(alpha_m, beta_k, beta_k_init, beta_k_comm)

    # Run the dynamic analysis

    op.wipeAnalysis()

    op.algorithm('Newton')
    op.system('SparseGeneral')
    op.numberer('RCM')
    op.constraints('Transformation')
    op.integrator('Newmark', 0.5, 0.25)
    op.analysis('Transient')

    tol = 1.0e-10
    iterations = 10
    op.test('EnergyIncr', tol, iterations, 0, 2)
    analysis_time = (len(values) - 1) * dt
    analysis_dt = 0.001
    outputs = {
        "time": [],
        "rel_disp": [],
        "rel_accel": [],
        "rel_vel": [],
        "force": []
    }

    while op.getTime() < analysis_time:
        curr_time = op.getTime()
        op.analyze(1, analysis_dt)
        outputs["time"].append(curr_time)
        outputs["rel_disp"].append(op.nodeDisp(top_node, 1))
        outputs["rel_vel"].append(op.nodeVel(top_node, 1))
        outputs["rel_accel"].append(op.nodeAccel(top_node, 1))
        op.reactions()
        outputs["force"].append(-op.nodeReaction(bot_node, 1))  # Negative since diff␣
↪node
    op.wipe()
    for item in outputs:
        outputs[item] = np.array(outputs[item])

    return outputs


def show_single_comparison():
    """
    Create a plot of an elastic analysis, nonlinear analysis and closed form elastic

    :return:
    """

    record_filename = 'test_motion_dt0p01.txt'
    motion_step = 0.01
    rec = np.loadtxt(record_filename)
    acc_signal = eqsig.AccSignal(rec, motion_step)
    period = 1.0
    xi = 0.05
```

(continues on next page)

(continued from previous page)

```
    mass = 1.0
    f_yield = 1.5  # Reduce this to make it nonlinear
    r_post = 0.0

    periods = np.array([period])
    resp_u, resp_v, resp_a = duhamels.response_series(motion=rec, dt=motion_step, periods=periods, xi=xi)

    k_spring = 4 * np.pi ** 2 * mass / period ** 2
    outputs = get_inelastic_response(mass, k_spring, f_yield, rec, motion_step, xi=xi, r_post=r_post)
    outputs_elastic = get_inelastic_response(mass, k_spring, f_yield * 100, rec, motion_step, xi=xi, r_post=r_post)
    ux_opensees = outputs["rel_disp"]
    ux_opensees_elastic = outputs_elastic["rel_disp"]

    bf, sps = plt.subplots(nrows=2)
    sps[0].plot(acc_signal.time, resp_u[0], label="Eqsig")
    sps[0].plot(outputs["time"], ux_opensees, label="Opensees fy=%.3gN" % f_yield, ls="--")
    sps[0].plot(outputs["time"], ux_opensees_elastic, label="Opensees fy=%.3gN" % (f_yield * 100), ls="--")
    sps[1].plot(acc_signal.time, resp_a[0], label="Eqsig")  # Elastic solution
    time = acc_signal.time
    acc_opensees_elastic = np.interp(time, outputs_elastic["time"], outputs_elastic["rel_accel"]) - rec
    print("diff", sum(acc_opensees_elastic - resp_a[0]))
    sps[1].plot(time, acc_opensees_elastic, label="Opensees fy=%.2gN" % (f_yield * 100), ls="--")
    sps[0].legend()
    sps[1].legend()
    plt.show()


if __name__ == '__main__':
    show_single_comparison()
```

1.13 Tsunami Examples

1.13.1 Dambreak Analysis

1. The source code is shown below, which can be downloaded `here`.
2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. The folder `dambreak/` must exist before running the script.
4. Run the source code in your favorate Python program.
5. The ParaView is needed to view the results. To view the displaced shape of fluid, use the "Warp By Vector" filter with scale factor = 1.0.

```
1 import sys
2 sys.path.append('/path/to/OpenSeesPy')
3 from opensees import *
```

(continues on next page)

(continued from previous page)

```

import numpy as np
import matplotlib.pyplot as plt

# ------------------------------
# Start of model generation
# -----------------------------

# remove existing model
wipe()

# set modelbuilder
model('basic', '-ndm', 2, '-ndf', 2)

# geometric
L = 0.146
H = L*2
H2 = 0.3
h = 0.005
alpha = 1.2
tw =  3*h

# material
rho = 1000.0
mu = 0.0001
b1 = 0.0
b2 = -9.81
thk = 0.012
kappa = -1.0

# time steps
dtmax = 1e-3
dtmin = 1e-6
totaltime = 1.0

# filename
filename = 'dambreak'

# recorder
recorder('PVD', filename, 'disp', 'vel', 'pressure')

# nodes
node(1, 0.0, 0.0)
node(2, L, 0.0)
node(3, L, H)
node(4, 0.0, H)
node(5, 0.0, H2)
node(6, 4*L, 0.0)
node(7, 4*L, H2)
node(8, -tw, H2)
node(9, -tw, -tw)
node(10, 4*L+tw, -tw)
node(11, 4*L+tw, H2)

# fluid mesh
fluid = 4
ndf = 2
```

(continues on next page)

(continued from previous page)

```
id = -1
mesh('line', 1, 9, 4,5,8,9,10,11,7,6,2, id, ndf, h)
mesh('line', 2, 3, 2,1,4, id, ndf, h)
mesh('line', 3, 3, 2,3,4, id, ndf, h)

eleArgs = ['PFEMElementBubble',rho,mu,b1,b2,thk,kappa]
mesh('tri', fluid, 2, 2,3, id, ndf, h, *eleArgs)

# wall mesh
wall = 5
id = 1
mesh('tri', wall, 2, 1,2, id, ndf, h)

for nd in getNodeTags('-mesh', wall):
    fix(nd, 1,1)

# save the original modal
record()

# create constraint object
constraints('Plain')

# create numberer object
numberer('Plain')

# create convergence test object
test('PFEM', 1e-5, 1e-5, 1e-5, 1e-5, 1e-15, 1e-15, 100, 3, 1, 2)

# create algorithm object
algorithm('Newton')

# create integrator object
integrator('PFEM')

# create SOE object
system('PFEM')

# create analysis object
analysis('PFEM', dtmax, dtmin, b2)

# analysis
while getTime() < totaltime:

    # analysis
    if analyze() < 0:
        break

    remesh(alpha)
```

1.13.2 Dambreak with Elastic Obstacle Analysis

1. The source code is shown below, which can be downloaded `here`.

2. Change the line 2 below to set the right path where the OpenSeesPy library located.
3. The folder `obstacle/` must exist before running the script.
4. Run the source code in your favorate Python program.
5. The ParaView is needed to view the results. To view the displaced shape of fluid, use the "Warp By Vector" filter with scale factor = 1.0.

```
1  import sys
2  sys.path.append('/path/to/OpenSeesPy')
3  from opensees import *
4
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # ------------------------------
9  # Start of model generation
10 # -----------------------------
11
12 # remove existing model
13 wipe()
14
15 # set modelbuilder
16 model('basic', '-ndm', 2, '-ndf', 3)
17
18 # geometric
19 L = 0.146
20 H = 2*L
21 H2 = 0.3
22 b = 0.012
23 h = 0.005
24 alpha = 1.2
25 Hb = 20.0*b/3.0
26 tw =  3*h
27
28 # material
29 rho = 1000.0
30 mu = 0.0001
31 b1 = 0.0
32 b2 = -9.81
33 thk = 0.012
34 kappa = -1.0
35 #kappa = 2.15e9
36
37 rhos = 2500.0
38 A = thk*thk
39 E = 1e6
40 Iz = thk*thk*thk*thk/12.0
41 bmass = A*Hb*rhos
42
43 # analysis
44 dtmax = 1e-3
45 dtmin = 1e-6
46 totaltime = 1.0
47
48 filename = 'obstacle'
49
50
```

(continues on next page)

(continued from previous page)

```
# recorder
recorder('PVD', filename, 'disp', 'vel', 'pressure')

# nodes
node(1, 0.0, 0.0)
node(2, L, 0.0)
node( 3, L, H, '-ndf', 2)
node(4, 0.0, H)
node(5, 0.0, H2)
node(6, 4*L, 0.0)
node(7, 4*L, H2)
node(8, -tw, H2)
node(9, -tw, -tw)
node(10, 4*L+tw, -tw)
node(11, 4*L+tw, H2)
node(12, 2*L, 0.0)
node(13, 2*L, Hb)

# transformation
transfTag = 1
geomTransf('Corotational', transfTag)

# section
secTag = 1
section( 'Elastic', secTag, E, A, Iz)

# beam integration
inteTag = 1
numpts = 2
beamIntegration( 'Legendre', inteTag, secTag, numpts)

# beam mesh
beam = 6
id = 1
ndf = 3
mesh('line', beam, 2, 12, 13, id, ndf, h, 'dispBeamColumn', transfTag, inteTag)

# fluid mesh
fluid = 4
ndf = 2
id = -1
mesh('line', 1, 10, 4,5,8,9,10,11,7,6,12,2, id, ndf, h)
mesh('line', 2, 3, 2,1,4, id, ndf, h)
mesh('line', 3, 3, 2,3,4, id, ndf, h)

eleArgs = ['PFEMElementBubble',rho,mu,b1,b2,thk,kappa]
mesh('tri', fluid, 2, 2,3, id, ndf, h, *eleArgs)

# wall mesh
wall = 5
id = 1
mesh('tri', wall, 2, 1,2, id, ndf, h)

for nd in getNodeTags('-mesh', wall):
    fix(nd, 1,1,1)

# save the original modal
```

(continues on next page)

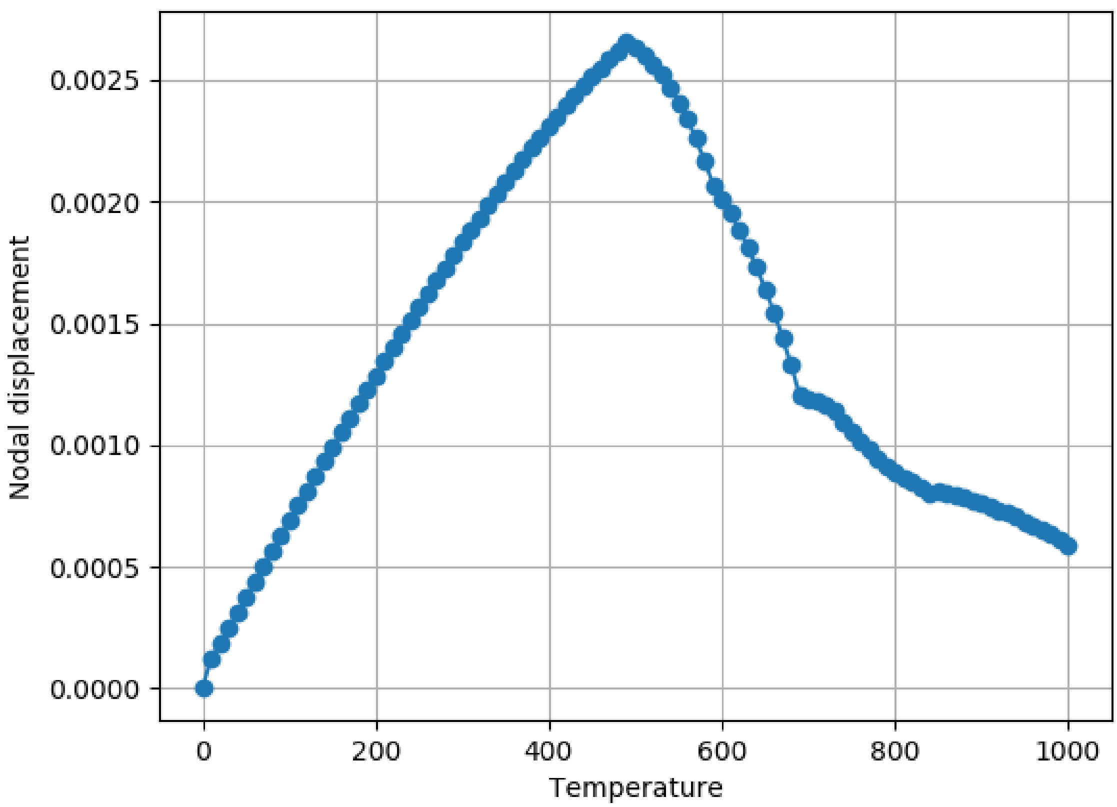
(continued from previous page)

```
record()

# create constraint object
constraints('Plain')

# create numberer object
numberer('Plain')

# create convergence test object
test('PFEM', 1e-5, 1e-5, 1e-5, 1e-5, 1e-15, 1e-15, 100, 3, 1, 2)

# create algorithm object
algorithm('Newton')

# create integrator object
integrator('PFEM')

# create SOE object
system('PFEM')

# create analysis object
analysis('PFEM', dtmax, dtmin, b2)

# analysis
while getTime() < totaltime:

    # analysis
    if analyze() < 0:
        break

    remesh(alpha)
```

1.14 Other Examples

1.14.1 Restrained beam under thermal expansion

1. The original model can be found here.
2. The Pypton source code is shown below, which can be downloaded `here`.
3. Change the line 2 below to set the right path where the OpenSeesPy library located.
4. Make sure the numpy and matplotlib packages are installed in your Python distribution.
5. Run the source code in your favorate Python program and should see




```
import sys
sys.path.append('/path/to/OpenSeesPy')
from opensees import *

import numpy as np
import matplotlib.pyplot as plt

# define model
model('basic', '-ndm', 2, '-ndf', 3)

#define node
node(1, 0.0, 0.0)
node(2, 2.0, 0.0)
node(3, 1.0, 0.0)

#define boundary condition
fix(1, 1, 1, 1)
fix(2, 1, 1, 1)
fix(3, 0, 1, 1)

#define an elastic material with Tag=1 and E=2e11.
matTag = 1
uniaxialMaterial('Steel01Thermal', 1, 2e11, 2e11, 0.01)

#define fibred section Two fibres: fiber $yLoc $zLoc $A $matTag
```

(continues on next page)

(continued from previous page)

```
secTag = 1
section('FiberThermal',secTag)
fiber(-0.025, 0.0, 0.005, matTag)
fiber(0.025, 0.0, 0.005, matTag)

#define coordinate transforamtion
#three transformation types can be chosen: Linear, PDelta, Corotational)
transfTag = 1
geomTransf('Linear', transfTag)

# beam integration
np = 3
biTag = 1
beamIntegration('Lobatto',biTag, secTag, np)

#define beam element
element('dispBeamColumnThermal', 1, 1, 3, transfTag, biTag)
element('dispBeamColumnThermal', 2, 3, 2, transfTag, biTag)

# define time series
tsTag = 1
timeSeries('Linear',tsTag)

# define load pattern
patternTag = 1
maxtemp = 1000.0
pattern('Plain', patternTag, tsTag)
eleLoad('-ele', 1, '-type', '-beamThermal', 1000.0, -0.05, 1000.0, 0.05)
#eleLoad -ele 2 -type -beamThermal 0 -0.05 0 0.05

# define analysis
incrtemp = 0.01
system('BandGeneral')
constraints('Plain')
numberer('Plain')
test('NormDispIncr', 1.0e-3,  100, 1)
algorithm('Newton')
integrator('LoadControl', incrtemp)
analysis('Static')

# analysis
nstep = 100
temp = [0.0]
disp = [0.0]
for i in range(nstep):
    if analyze(1) < 0:
        break

    temp.append(getLoadFactor(patternTag)*maxtemp)
    disp.append(nodeDisp(3,1))


plt.plot(temp,disp,'-o')
plt.xlabel('Temperature')
plt.ylabel('Nodal displacement')
plt.grid()
plt.show()
```

Index

P

R

S

T

U

V

W