OpenSees Command Language Manual

Software Authors:

Frank McKenna, Gregory L. Fenves, et al.

Manual Authors:

Silvia Mazzoni, Frank McKenna, Gregory L. Fenves, et al.

OpenSees release 1.6

February 2005

Printed on 27 February, 2005

Contents

CHAPTER 1

Introduction

This document is intended to outline the basic commands currently available with the OpenSees interpreter. This interpreter is an extension of the Tcl/Tk language for use with OpenSees.

OpenSees is an object-oriented framework for finite element analysis. OpenSees' intended users are in the research community. A key feature of OpenSees is the interchangeability of components and the ability to integrate existing libraries and new components into the framework (not just new element classes) without the need to change the existing code. Core components, that is the abstract base classes, define the minimal interface (minimal to make adding new component classes easier but large enough to ensure all that is required can be accommodated).

In This Chapter

Notation

The notation presented in this chapter is used throughout this document.

Input values are a string, unless the first character is a **$**, in which case an integer, floating point number or variable is to be provided. In the Tcl language, variable references start with the **$** character. Tcl expressions can also be used as input to the commands where the input value is specified by the first character being a **$**.

Optional values are identified in enclosing **<>** braces.

When specifying a variable quantity of values, the command line contains **(x $values)**. The number of values required, x, and the types of values, $values, are specified in the description of the command.

An arbitrary number of input values is indicated with the dot-dot-dot notation, i.e. **$value1 $value2 ...**

The OpenSees interpreter constructs objects in the order they are specified by the user. New objects are often based on previously-defined objects. When specified as an object parameter, a previously-defined object must have already been added to the Domain. This requirement is specified in the description of the command arguments.

Example command:

node $nodeTag (ndm $coords) <-mass (ndf $MassValues)>

This line executes the *node command* (page 28) assigns coordinates and masses to a specified node. The **$nodeTag** argument is an integer tag identifying the node.The coordinate arguments are specified with the parentheses **()** because the number of arguments is dependent on the definition of the model (*ndm* (page 26)): two arguments in 2D and three in 3D.

The mass specification at the node definition is optional. Therefore, it is enclosed in **<>** braces. The number of mass arguments is also dependent on the model definition, depending on the number of degrees of freedom assigned to a node (*ndf* (page 26)).

Copyright

Introduction to the Tcl command language

The Tcl scripting language was chosen to support the OpenSees commands, which are used to define the problem geometry, loading, formulation and solution. These commands are one-line commands which have specific tasks, as described in this manual. The Tcl language provides useful programming tools, such as variables manipulation, mathematical-expression evaluation and control structures.

Tcl is a string-based scripting language which allows the following:

- Variables and variable substitution
- Mathematical-expression evaluation
- Basic control structures (if , while, for, foreach)
- Procedures
- File manipulation

More information on Tcl commands can be found at its web site: *Tcl/Tk Primer* (http://dev.scriptics.com/scripting/primer.html) (http://dev.scriptics.com/scripting/primer.html)

Handy Tcl commands:

incr - Increment the value of a variable:

```
set a 1
incr a
```

a book reference:

Brent Welch <welch@acm.org>, Ken Jones, and Jeff Hobbs: **Practical Programming in Tcl and Tk**, (http://www.beedub.com/book/) 4th Edition ISBN: 0-13-038560-3, June, 2003 (http://www.beedub.com/book/)

Tcl Commands Format

Tcl scripts are made up of commands separated by new lines or semicolon (;).

The basic syntax for a Tcl command is:

```
command $arg1 $arg2  ...
```

command	name of the Tcl command or user-defined procedure
$arg1 $arg2 ...	arguments for the command

Tcl allows any argument to be nested command:

```
command [nested-command1] [nested-command2]
```

where the [] are used to delimit the nested commands. The Tcl interpreter will first evaluate the nested commands and will then evaluate the outer command with the results to the nested commands.

The most basic command in Tcl is the set command:

```
set variable $value
```

for example:

```
set a 5
```

The Tcl interpreter regards a command starting with the pond sign (#) to be a comment statement, so it does not execute anything following the #. For example:

```
# this command assigns the value 5 to the variable a
set a 5
```

The pound sign and the semicolon can be used together to put comments on the same line as the command. For example:

```
set a 5;  # this command assigns the value 5 to the variable a
```

Example Tcl Commands

arithmetic	procedure	for & foreach functions

>set a 1
1
>set b a
a
>set b $a
1
>expr 2 + 3
5
>expr 2 + $a
3
>set b [expr 2 + $a]
3
>
	>proc sum {a b} {
	return [expr $a + $b]
	}
	>sum 2 3
	5
	>set c [sum 2 3]
	5
	>
		for {set i 1} {$i < 10} {incr i 1} {
		puts “i equals $i”
		}
		
		set sum 0
		foreach value {1 2 3 4} {
		set sum [expr $sum + $value]
		}
		puts $sum
		10
		>

file manipulation	**procedure & if statement**	

>set fileId [open tmp w]
anumber
>puts $fileId “hello”
>close $fileID
>type tmp
hello
>
 	
>source Example1.tcl
	>proc guess {value} {
	global sum
	if {$value < $sum} {
	puts “too low”
	} else {
	if {$value > $sum} {
	puts “too high”
	} else { puts “you got it!”}
	}
	}
	> guess 9
	too low
	>	

Additional Tcl Resources

Here are additional resources for Tcl:

http://www.freeprogrammingresources.com/tcl.html (http://www.freeprogrammingresources.com/tcl.html) (a large list of helpful resources)
http://www.tcl.tk/man/ (http://www.tcl.tk/man/) (Tcl/Tk manual pages)
http://www.mit.edu/afs/sipb/user/golem/doc/tcltk-iap2000/TclTk1.html (http://www.mit.edu/afs/sipb/user/golem/doc/tcltk-iap2000/TclTk1.html) (a tutorial describing many commands by describing their implementation in a short program)
http://www.beedub.com/book/ (http://www.beedub.com/book/) (some sample chapters from Practical Programming in Tcl and Tk, by Welch and Jones)
http://philip.greenspun.com/tcl/ (http://philip.greenspun.com/tcl/) (not the most readable tutorial IMHO, but it does have Tickle-me-Elmo ;) It can be accessed from the link below as well.) http://www.tcl.tk/scripting/ (http://www.tcl.tk/scripting/)

http://hegel.ittc.ukans.edu/topics/tcltk/tutorial-noplugin/index.html (http://hegel.ittc.ukans.edu/topics/tcltk/tutorial-noplugin/index.html) (a short tutorial on the essential Tcl commands, also includes a manual of Tcl/Tk commands at the website below) http://hegel.ittc.ukans.edu/topics/linux/man-pages/index/index-mann.html (http://hegel.ittc.ukans.edu/topics/linux/man-pages/index/index-mann.html)
http://pages.cpsc.ucalgary.ca/~saul/personal/archives/Tcl-Tk_stuff/tcl_examples/ (http://pages.cpsc.ucalgary.ca/~saul/personal/archives/Tcl-Tk_stuff/tcl_examples/) (Tk widgets with screenshots)

OpenSees Interpreter

The main abstractions of OpenSees will be explained using the OpenSees interpreter. The interpreter is an extension of the *Tcl* (page 14) scripting language. The OpenSees interpreter adds commands to Tcl for finite element analysis. Each of these commands is associated (bound) with a C++ procedure that is provided. It is this procedure that is called upon by the interpreter to parse the command. In this document we outline only those commands which have been added to Tcl by OpenSees.

For OpenSees we have added commands to Tcl for finite element analysis:

- Modeling – create nodes, elements, loads and constraints
- Analysis – specify the analysis procedure.
- Output specification – specify what it is you want to monitor during the analysis.

➢ ***HELP***

OpenSees Documentation Web Page (http://opensees.berkeley.edu/OpenSees/primer.html)

http://opensees.berkeley.edu/cgi-bin/OpenSeesCommands.pl (http://opensees.berkeley.edu/cgi-bin/OpenSeesCommands.pl)

CHAPTER 2

OpenSees

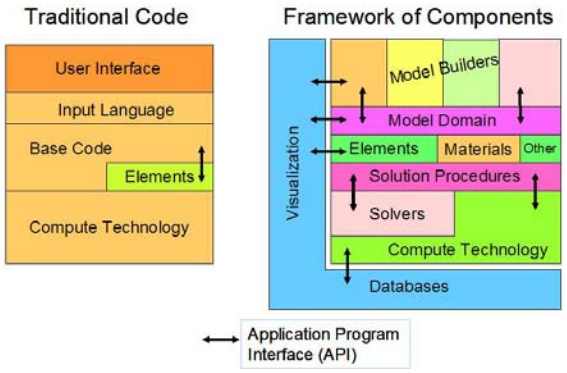
➢ *What is OpenSees?*

- • An object-oriented software framework for simulation applications in earthquake engineering using finite element methods. OpenSees is not a code.
- • A communication mechanism within PEER for exchanging and building upon research accomplishments.
- • As open-source software, it has the potential for a community code for earthquake engineering.

why OpenSEES?

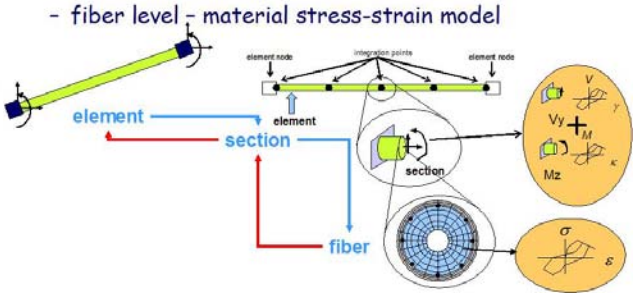
- The library of materials, elements and analysis commands makes OpenSEES a powerful tool for numerical simulation of nonlinear structural and geotechnical systems
- The OpenSEES library of components is ever-growing and at the leading edge of numerical-simulation models
- The OpenSEES interface is based on a command-driven scripting language which enables the user to create more-versatile input files. (*I'll show you how!*)
- OpenSEES is not a black box, making it a useful educational tool for numerical modeling
- You can create your own material, element or analysis tools and incorporate them into OpenSEES
- NEES is supporting integration of OpenSEES as the simulation component of laboratory testing

traditional code vs. OpenSEES

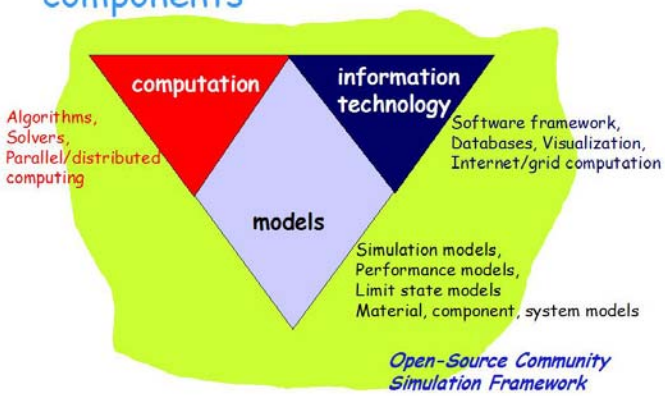


my favorite:

- You can describe a structural/geotech. component at a number of levels:
  - element level – force-deformation model
  - section level – moment-curvature model
  - fiber level – material stress-strain model



simulation-framework components



OpenSEES features

- models:
  - linear & nonlinear structural and geotechnical models
- simulations:
  - static push-over analyses
  - static reversed-cyclic analyses
  - dynamic time-series analyses
    - uniform-support excitation
    - multi-support excitation

simulation features

- Model-Building
  - model
  - node
  - mass
  - Constraints
  - uniaxialMaterial
  - nDMaterial
  - section
  - element
  - block
  - region
  - Geometric Transformation
  - Time Series
  - pattern
- Analysis
  - constraints
  - numberer
  - system
  - test
  - algorithm
  - integrator
  - analysis
  - rayleigh
  - eigen
  - dataBase
- Recorders
  - Node
  - EnvelopeNode
  - Drift
  - Element
  - EnvelopeElement
  - Display
  - Plot
  - playback

modeling features

- uniaxial materials:
  - Elastic Material
  - Elastic-Perfectly Plastic Material
  - Elastic-Perfectly Plastic Gap Material
  - Parallel Material
  - Series Material
  - Hardening Material
  - Steel01 Material
  - Concrete01 Material
  - Elastic-No Tension Material
  - Hysteretic Material
  - Viscous Material
  - PINCHING4 Material
  - Fedeas Materials
  - PyTzQz Uniaxial Materials
- nD materials:
  - Elastic Isotropic Material
  - J2 Plasticity Material
  - Plane Stress Material
  - Plate Fiber Material
  - Template Elasto-Plastic Material
  - FluidSolidPorousMaterial Material
  - PressureIndependMultiYield Material
  - PressureDependMultiYield Material

modeling features

- elements:
  - Truss
  - Corotational Truss
  - Elastic Beam Column
  - NonLinear Beam-Column
  - Zero-Length
  - Quadrilateral
  - Brick
  - FourNodeQuadUP
  - BeamColumnJoint
- sections:
  - Elastic Section
  - Uniaxial Section
  - Fiber Section
  - Section Aggregator
  - Elastic Membrane Plate Section
  - Plate Fiber Section
  - Bidirectional Section

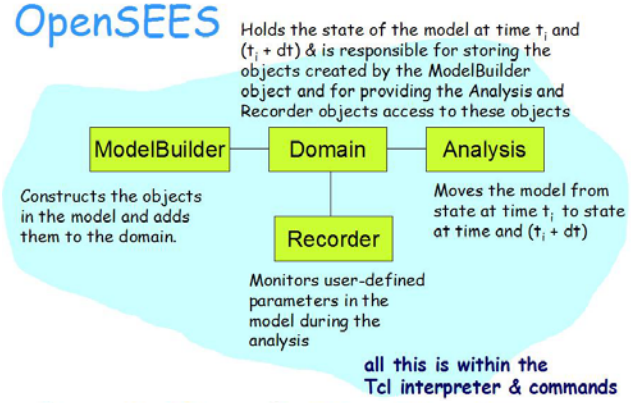
analysis features

- **Linear Equation Solvers** -- provide the solution of the linear system of equations Ku = P. Each solver is tailored to a specific matrix topology.
  - Profile SPD -- Direct profile solver for symmetric positive definite matrices
  - Band General -- Direct solver for banded unsymmetric matrices
  - Band SPD -- Direct solver for banded symmetric positive definite matrices
  - Sparse General -- Direct solver for unsymmetric sparse matrices
  - Sparse Symmetric -- Direct solver for symmetric sparse matrices
  - UmfPack General -- Direct UmfPack solver for unsymmetric matrices
  - Full General -- Direct solver for unsymmetric dense matrices
  - Conjugate Gradient -- Iterative solver using the preconditioned conjugate gradient method
- **Eigenvalue Solvers** -- provide the solution of the generalized eigenvalue problem Kv = MvL
  - Symmetric Arpack -- Arpack solver for symmetric matrices
  - Band Arpack -- Arpack solver for banded matrices

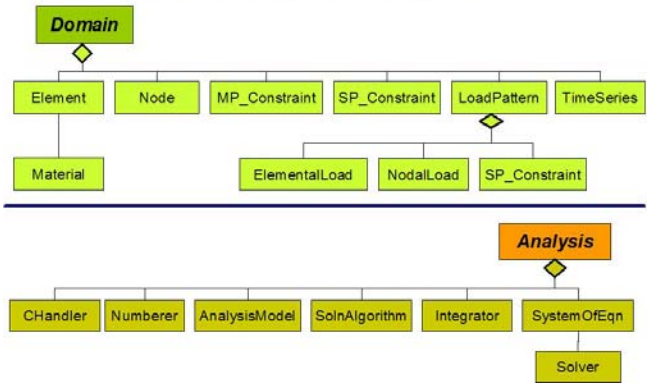
- **DOF Numberers** -- number the degrees of freedom in the domain
  - Plain -- Uses the numbering provided by the user
  - RCM -- Renumbers the DOF to minimize the matrix band-width using the Reverse Cuthill-McKee algorithm
- **Static Integrators** -- determine the next time step for an analysis
  - Load Control -- Specifies the incremental load factor to be applied to the loads in the domain
  - Displacement Control -- Specifies the incremental displacement at a specified DOF in the domain
  - Minimum Unbalanced Displacement Norm -- Specifies the incremental load factor such that the residual displacement norm in minimized
  - Arc Length -- Specifies the incremental arc-length of the load-displacement path
- **Transient Integrators** -- determine the next time step for an analysis including inertial effects
  - Newmark -- The two parameter time-stepping method developed by Newmark
  - HHT -- The three parameter Hilbert-Hughes-Taylor time-stepping method
  - Central Difference -- Approximates velocity and acceleration by centered finite differences of displacement
- **Solution Algorithms** -- Iterate from the last time step to the current
  - Linear -- Uses the solution at the first iteration and continues
  - Newton -- Uses the tangent at the current iteration to iterate to convergence
  - Modified Newton -- Uses the tangent at the first iteration to iterate to convergence

OpenSees is comprised of a set of modules to perform creation of the finite element model, specification of an analysis procedure, selection of quantities to be monitored during the analysis, and the output of results. In each finite element analysis, an analysis is used to construct 4 main types of objects, as shown

main abstractions in OpenSEES



domain & analysis objects



In This Chapter

ModelBuilder Object

The model builder constructs As in any finite element analysis, the analyst's first step is to subdivide the body being studied into elements and nodes, to define loads acting on the elements and nodes, and to define constraints acting on the nodes.

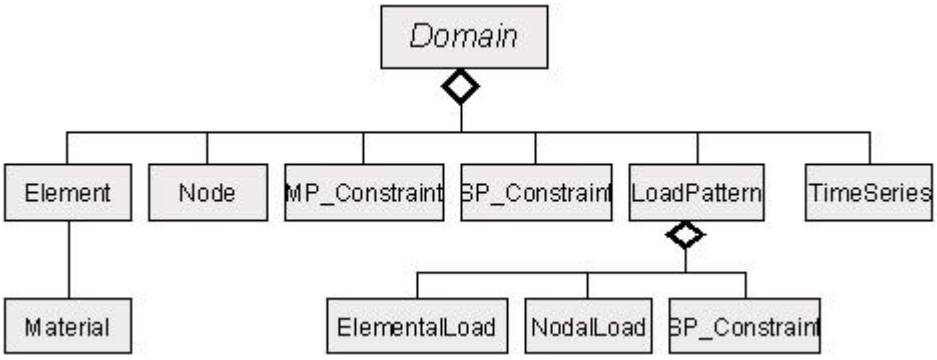
The ModelBuilder is the object in the program responsible for building the following objects in the model and adding them to the domain:

- **Node** (page 28)
- **Mass** (page 29)
- **Material** (page 108, page 35)
- **Section** (page 129)
- **Element** (page 146)
- **LoadPattern** (page 214)
- **TimeSeries** (page 208)
- **Transformation** (page 200)
- **Block** (page 193)
- **Constraint** (page 232)

Domain Object

The Domain object is responsible for storing the objects created by the *ModelBuilder* (page 22) object and for providing the *Analysis* (page 229) and *Recorder* (page 24) objects access to these objects.

Figure 1: Domain Object



Recorder Object

The recorder object monitors user-defined parameters in the model during the analysis. This, for example, could be the displacement history at a node in a transient analysis, or the entire state of the model at each step of the solution procedure. Several *Recorder* (page 221) objects are created by the analyst to monitor the analysis.

➢ ***What does a recorder do?***

- Monitors the state of a domain component (node, element, etc.) during an analysis
- Writes this state to a file or to a database at selected intervals during the analysis
- There are also recorders for plotting and monitoring residuals

Once in a file, the information can be easily post-processed.

Analysis Object

The Analysis objects are responsible for performing the analysis. The analysis moves the model along from state at time t to state at time t + dt. This may vary from a simple *static* (page 256) linear analysis to a *transient* (page 257, page 258) non-linear analysis. In OpenSees each Analysis object is composed of several component objects, which define the type of analysis how the analysis is performed.

CHAPTER 3

Model-Building Objects

These objects are used to create the physical model.

In This Chapter

CHAPTER 4

model Command

This command is used to construct a ModelBuilder object.

Currently there is only one type of ModelBuilder accepted.

For an example of this command, refer to the *Model Building Example* (page 278)

In This Chapter

Basic Model Builder

This command is used to construct the BasicBuilder object.

model BasicBuilder -ndm $ndm <-ndf $ndf>

$ndm	dimension of problem (1,2 or 3)
$ndf	number of degrees of freedom at node (optional)
	(default value depends on value of ndm:
	ndm=1 -> ndf=1
	ndm=2 -> ndf=3
	ndm=3 -> ndf=6)

These additional commands allow for the construction of *Nodes* (page 28), *Masses* (page 29), *Materials* (page 108, page 35), *Sections* (page 129), *Elements* (page 146), *LoadPatterns* (page 214), *TimeSeries* (page 208), *Transformations* (page 200), *Blocks* (page 193) and *Constraints* (page 232). These additional commands are described in the subsequent chapters.

EXAMPLE:

model basic -ndm 3 -ndf 6; # 3 spacial dimensions, 6 DOF's per node

For an example of this command, refer to the *Model Building Example* (page 278)

build Command

This command is used to invoke build() (????) on the *ModelBuilder* (page 22) object.

build

This command has no effect a *BasicBuilder* (page 26) object, but will on other types of *ModelBuilder* (page 22) objects.

CHAPTER 5

node Command

This command is used to construct a Node object. It assigns coordinates and masses to the Node object.

node $nodeTag (ndm $coords) <-mass (ndf $MassValues)>

$nodeTag	integer tag identifying node
$coords	nodal coordinates (*ndm* (page 26) arguments)
$MassValues	nodal mass corresponding to each DOF (*ndf* (page 26) arguments) (optional)

The optional **-mass** string allows analyst the option of associating nodal mass with the node

EXAMPLE:

```
node 1   0.0   0.0   0.0;  # x,y,z coordinates (0,0,0) of node 1
node 2   0.0   120.   0.0;  # x,y,z coordinates (0,120,0) of node 2
```

For an example of this command, refer to the *Model Building Example* (page 278)

CHAPTER 6

mass Command

This command is used to set the mass at a node.

mass $nodeTag (ndf $MassValues)

$nodeTag	integer tag identifying the node associated with the mass
$MassValues	mass values corresponding to each nodal degrees of freedom (*ndf* (page 26) values)

EXAMPLE:

mass 2 2.5 0.0 2.5 0.0 0.0 0.0; # define mass in x and z coordinates

For an example of this command, refer to the *Model Building Example* (page 278)

CHAPTER 7

constraints objects

From Cook: " A constraint either prescribes the value of a DOF (as in imposing a support condition) or prescribes a relationship among DOF. In common terminology, a single-point constraint sets a single DOF to a known value (often zero) and a multi-point constraint imposes a relationship between two or more DOF.... For example, support conditions on a three-bar truss invoke single-point constraints, while rigid links and rigid elements each invoke a multi-point constraint."

In This Chapter

Single-Point Constraints

The following commands construct homogeneous single-point boundary constraints.

Fix Command

This command is used to construct homogeneous single-point boundary constraints.

fix $nodeTag (ndf $ConstrValues)

$nodeTag	integer tag identifying the node to be constrained
$ConstrValues	constraint type (0 or 1). *ndf* (page 22) values are specified, corresponding to the ndf degrees-of-freedom.
	The two constraint types are:
	0 unconstrained
	1 constrained

EXAMPLE:

fix 1 1 1 1 1 1 1; # node 1: fully fixed

fix 2 0 1 0 0 1 0 -mass 2.5 0.0 2.5 0.0 0.0 0.0; # node 2: restrain axial elongation and torsion, translational masses in x-z plane only

For an example of this command, refer to the *Model Building Example* (page 278)

fixX Command

This command is used to construct multiple homogeneous single-point boundary constraints for all nodes whose x-coordinate lies within a specified distance from a specified coordinate.

fixX $xCoordinate (ndf $ConstrValues) <-tol $tol>

For example, this command is used when specifying boundary conditions for a series of nodes lying in a plane parallel to the y-z plane in global coordinates.

$xCoordinate	x-coordinate of nodes to be constrained
$ConstrValues	constraint type (0 or 1). *ndf* (page 22) values are specified, corresponding to the ndf degrees-of-freedom. The two constraint types are: **0** unconstrained **1** constrained
$tol	user-defined tolerance (optional, default = 1e-10)

EXAMPLE:

fixX 0.0 1 1 1 1 1 1 -tol 0.1; # fully restrain all nodes in y-z plane at origin (x=0.0)

fixY Command

This command is used to construct multiple homogeneous single-point boundary constraints for all nodes whose y-coordinate lies within a specified distance from a specified coordinate.

fixY $yCoordinate (ndf $ConstrValues) <-tol $tol>

For example, this command is used when specifying boundary conditions for a series of nodes lying in a plane parallel to the x-z plane in global coordinates.

$yCoordinate	y-coordinate of nodes to be constrained
$ConstrValues	constraint type (0 or 1). ndf values are specified, corresponding *to* the ndf (page 26) degrees-of-freedom. The two constraint types are: **0** unconstrained **1** constrained
$tol	user-defined tolerance (optional, default = 1e-10)

EXAMPLE:

fixY 0.0 1 1 1 1 1 1 -tol 0.1; # fully restrain all nodes in x-z plane at origin (y=0.0)

fixZ Command

This command is used to construct multiple homogeneous single-point boundary constraints for all nodes whose z-coordinate lies within a specified distance from a specified coordinate.

fixZ $zCoordinate (ndf $ConstrValues) <-tol $tol>

For example, this command is used when specifying boundary conditions for a series of nodes lying in a plane parallel to the x-y plane in global coordinates.

$zCoordinate	z-coordinate of nodes to be constrained
$ConstrValues	constraint type (0 or 1). ndf values are specified, corresponding *to* (page 26)the ndf degrees-of-freedom. The two constraint types are: **0** unconstrained **1** constrained
$tol	user-defined tolerance (optional, default = 1e-10)

EXAMPLE:

fixZ 0.0 1 1 1 1 1 1 -tol 0.1; # fully restrain all nodes in x-y plane at origin (z=0.0)

Multi-Point Constraints

The following commands construct multi-point boundary constraints.

equalDOF Command

This command is used to construct a multi-point constraint between nodes.

equalDOF $rNodeTag $cNodeTag $dof1 $dof2 ...

$rNodeTag	integer tag identifying the retained, or master node (rNode)
$cNodeTag	integer tag identifying the constrained, or slave node (cNode)
$dof1 $dof2 ...	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* (page 26), the number of nodal degrees-of-freedom.

EXAMPLE:

equalDOF 2 3 1 3 5; # impose the traslational displacements in x and z directions, and rotation about the y-axis of node 3 to be the same as those of node 2.

rigidDiaphragm Command

This command is used to construct a number of Multi-Point Constraint (MP_Constraint) objects. 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.

rigidDiaphragm $perpDirn $masterNodeTag $slaveNodeTag1 $slaveNodeTag2 ...

$perpDirn	direction perpendicular to the rigid plane (i.e. direction 3 corresponds to the 1-2 plane) The rigid plane can be the 1-2, 1-3 or 2-3 plane
$masterNodeTag	integer tag identifying the master node
$slaveNodeTag1 $slaveNodeTag2 ...	nodes that are to be constrained to the master node

NOTE: The constraint object is constructed assuming small rotations.

NOTE: The rigidDiaphragm command works only for problems in three dimensions with six-degrees-of-freedom at the nodes (*ndf* (page 26) = 6).

EXAMPLE:

rigidDiaphragm 2 2 4 5 6; constrain nodes 4,5,6 to move as if in the same x-z plane as node 2.

rigidLink Command

This command is used to construct a single MP_Constraint object.

rigidLink $type $masterNodeTag $slaveNodeTag

$type	string-based argument for rigid-link type: **rod** 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.
$masterNodeTag	integer tag identifying the master node
$slaveNodeTag	integer tag identifying the slave node to be constrained to master node

NOTE: The constraint object constructed for the beam option assumes small rotations

EXAMPLE:

rigidLink beam 2 3; # connect node 3 to node 2 via a rigid link-beam.

CHAPTER 8

uniaxialMaterial Command

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

The valid queries to any uniaxial material when creating an *ElementRecorder* (page 224) are 'strain,' 'stress,' and 'tangent.'

In This Chapter

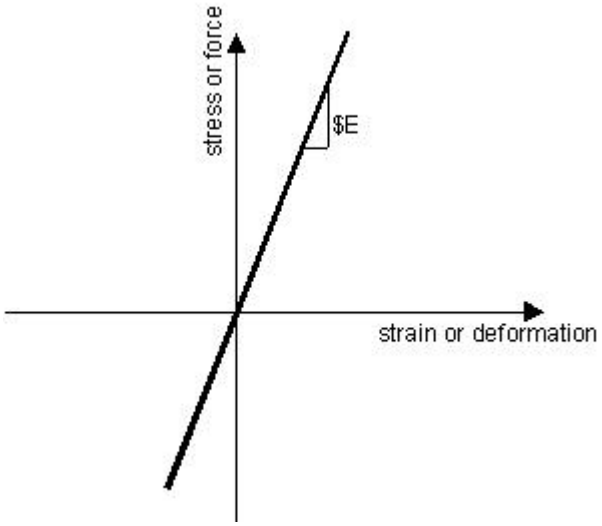
Elastic Material

This command is used to construct an elastic uniaxial material object.

uniaxialMaterial Elastic $matTag $E <$eta>

$matTag	unique material object integer tag
$E	tangent
$eta	damping tangent (optional, default=0.0)

Figure 2: Elastic Material



Elastic-Perfectly Plastic Material

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

uniaxialMaterial ElasticPP $matTag $E $epsyP <$epsyN $eps0>

$matTag	unique material object integer tag
$E	tangent

$epsyP strain or deformation at which material reaches plastic state in tension

$epsyN strain at which material reaches plastic state in compression (optional, default: tension value)

$eps0 initial strain (optional, default: zero)

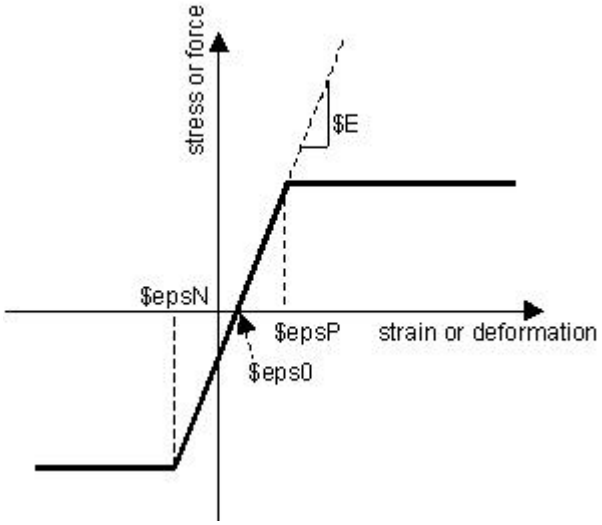


Figure 3: Elastic-Perfectly Plastic Material

Elastic-Perfectly Plastic Gap Material

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

uniaxialMaterial ElasticPPGap $matTag $E $Fy $gap

$matTag	unique material object integer tag
$E	tangent stiffness
$Fy	stress or force at which material reaches plastic state
$gap	initial gap (strain or deformation)

NOTE: To create a compression-only gap element, NEGATIVE values need to be specified for $Fy and $gap.

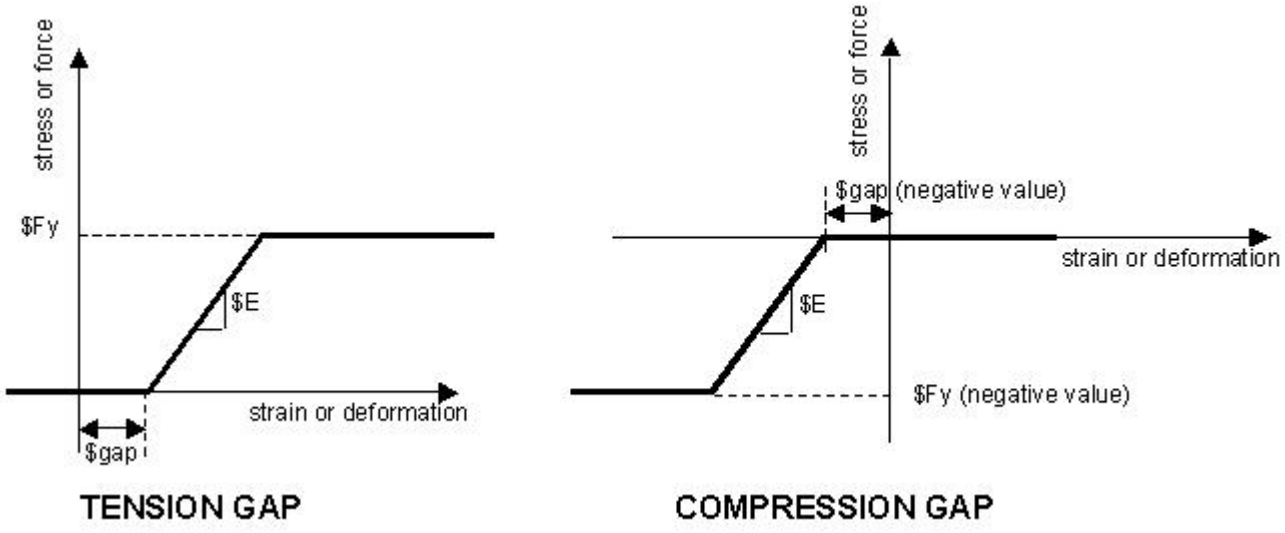


Figure 4: Elastic-Perfectly Plastic Gap Material

Parallel Material

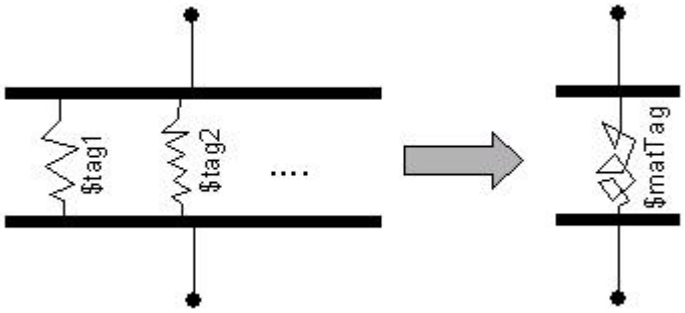
This command is used to construct a parallel material object made up of an arbitrary number of previously-constructed *UniaxialMaterial* (page 35) objects.

uniaxialMaterial Parallel $matTag $tag1 $tag2 ...

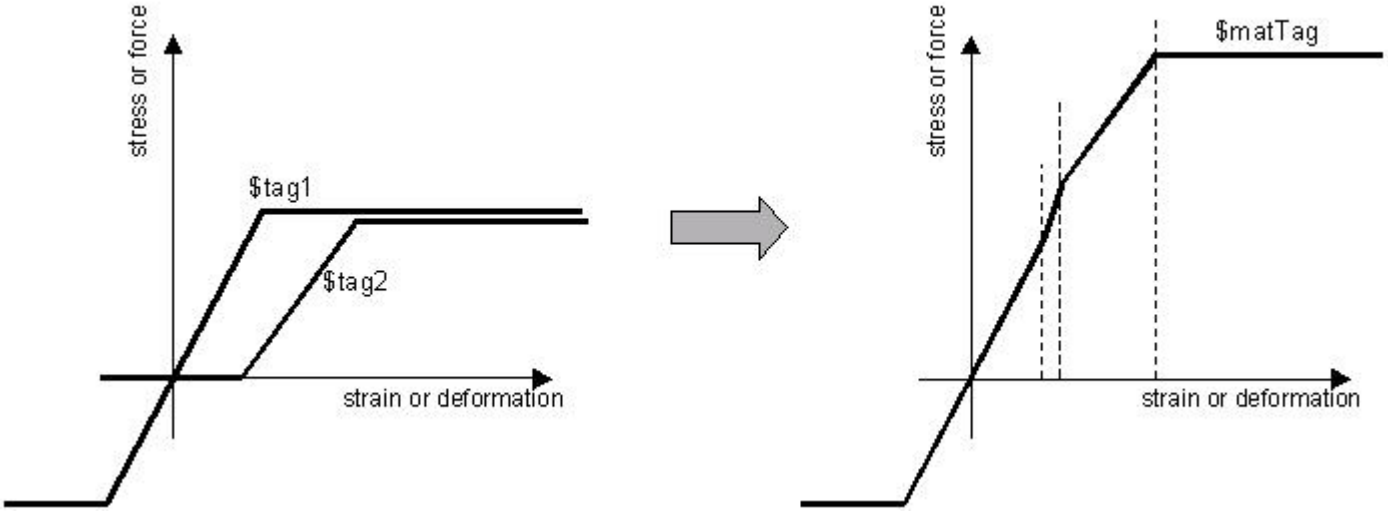
$matTag	unique material object integer tag
$tag1 $tag2 ...	identification of materials making up the material model

The parallel material is represented graphically:

Figure 5: Parallel Material



In a parallel model, strains are equal and stresses and stiffnesses are additive:



Series Material

This command is used to construct a series material object made up of an arbitrary number of previously-constructed *UniaxialMaterial* (page 35) objects.

uniaxialMaterial Series $matTag $tag1 $tag2 ...

$matTag	unique material object integer tag
$tag1 $tag2 ...	identification of materials making up the material model

The series material is represented graphically:

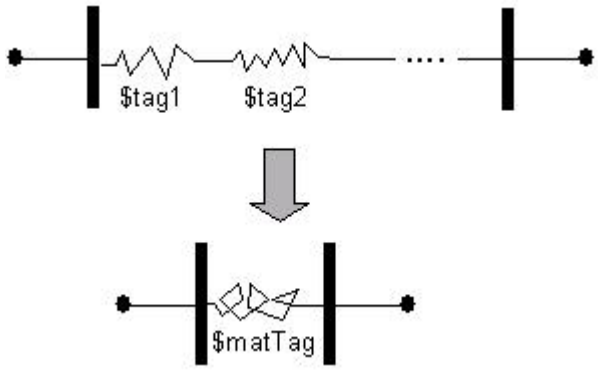


Figure 6: Series Material

In a series model, stresses are equal and strains and flexibilities are additive:

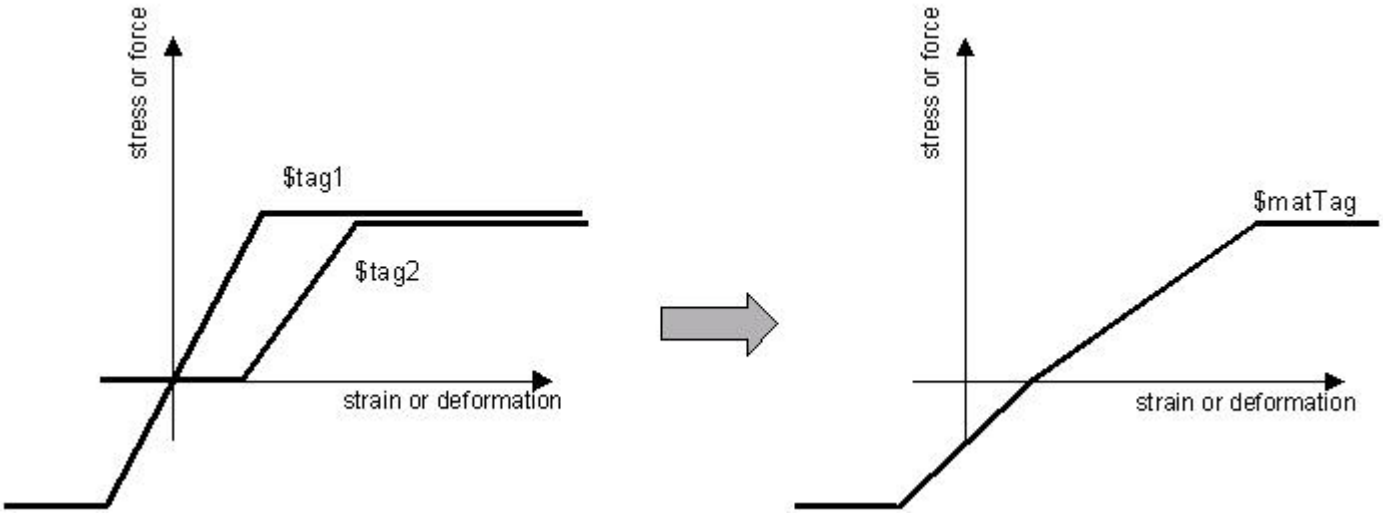


Figure 7: Series Material Relationship

Hardening Material

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 (REF???)

uniaxialMaterial Hardening $matTag $E $sigmaY $H_iso $H_kin <$eta>

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

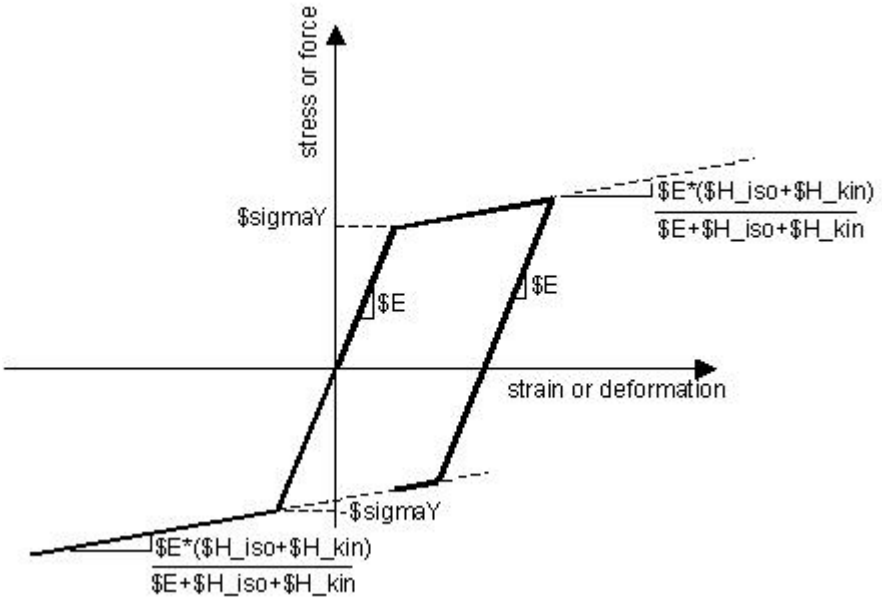


Figure 8: Hardening Material

Steel01 Material

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).

uniaxialMaterial Steel01 $matTag $Fy $E0 $b <$a1 $a2 $a3 $a4>

$matTag	unique material object integer tag
$Fy	yield strength
$E0	initial elastic tangent
$b	strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
$a1, $a2, $a3, $a4	isotropic hardening parameters: (optional, default: no isotropic hardening)

- **$a1** isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a2*($Fy/E0).
- **$a2** isotropic hardening parameter (see explanation under $a1)
- **$a3** isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a4*($Fy/E0)
- **$a4** isotropic hardening parameter (see explanation under $a3)

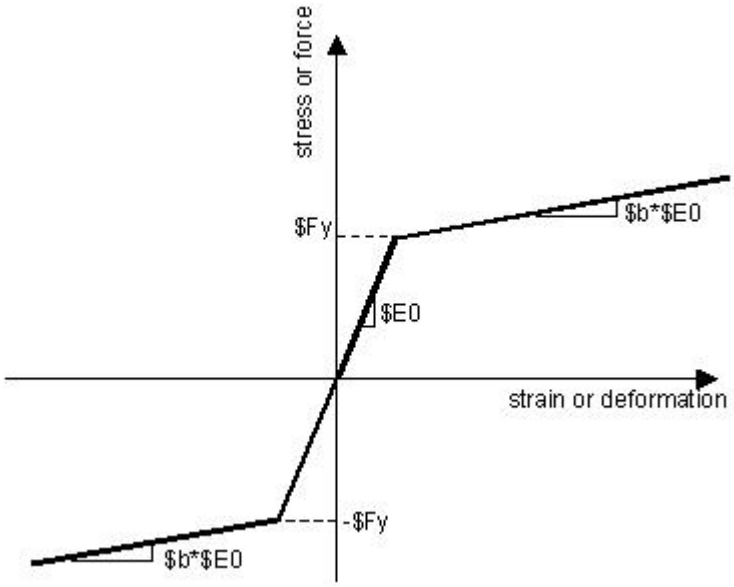

stress or force
$Fy
$b*$E0
$E0
strain or deformation
-$Fy
$b*$E0

Figure 9: Steel01 Material -- Material Parameters of Monotonic Envelope

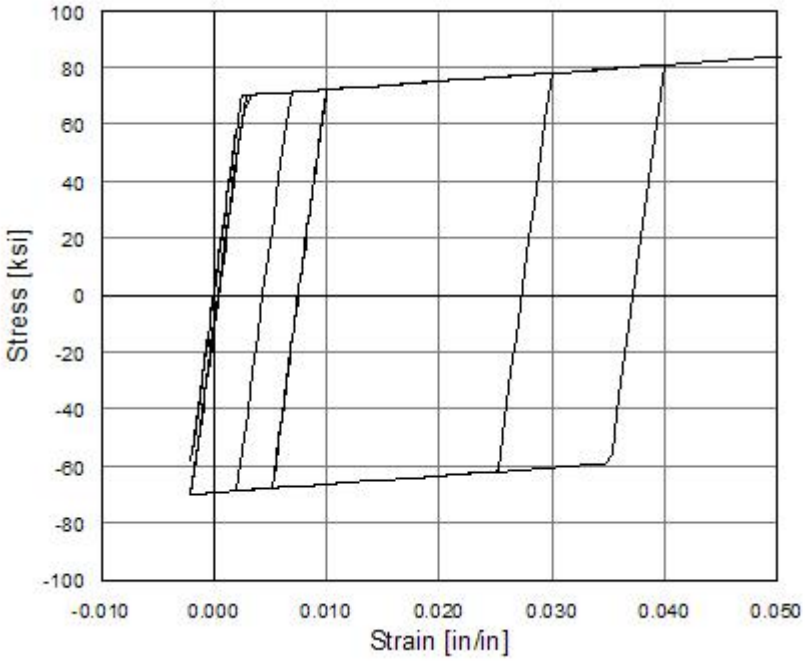
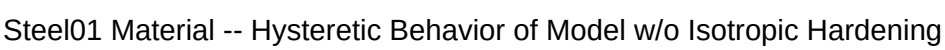


Figure 10: Steel01 Material -- Hysteretic Behavior of Model w/o Isotropic Hardening



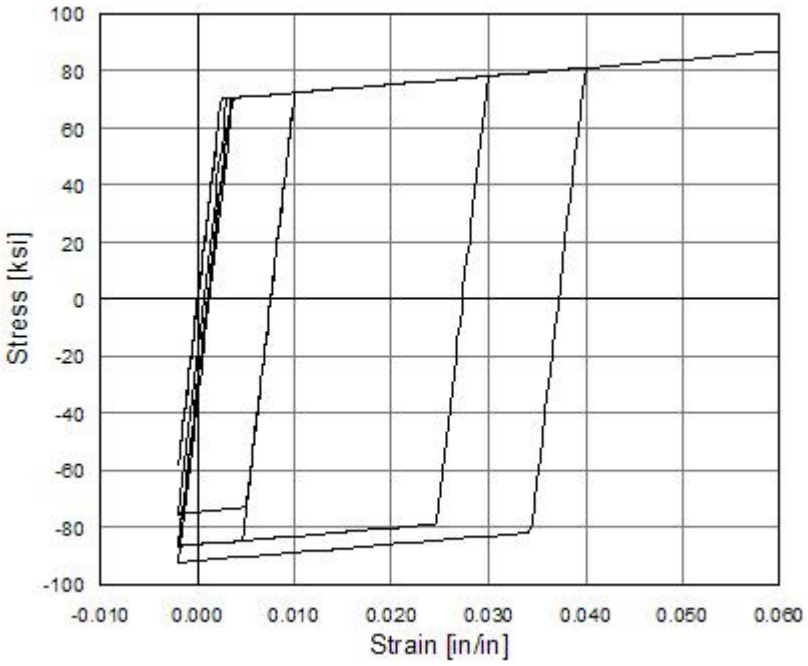

100
80
60
40
20
0
-20
-40
-60
-80
-100
Stress [ksi]
-0.010
0.000
0.010
0.020
0.030
0.040
0.050
0.060
Strain [in/in]

Figure 11: Hysteretic Behavior of Model with Isotropic Hardening in Compression

Steel01 Material -- Hysteretic Behavior of Model with Isotropic Hardening in Compression

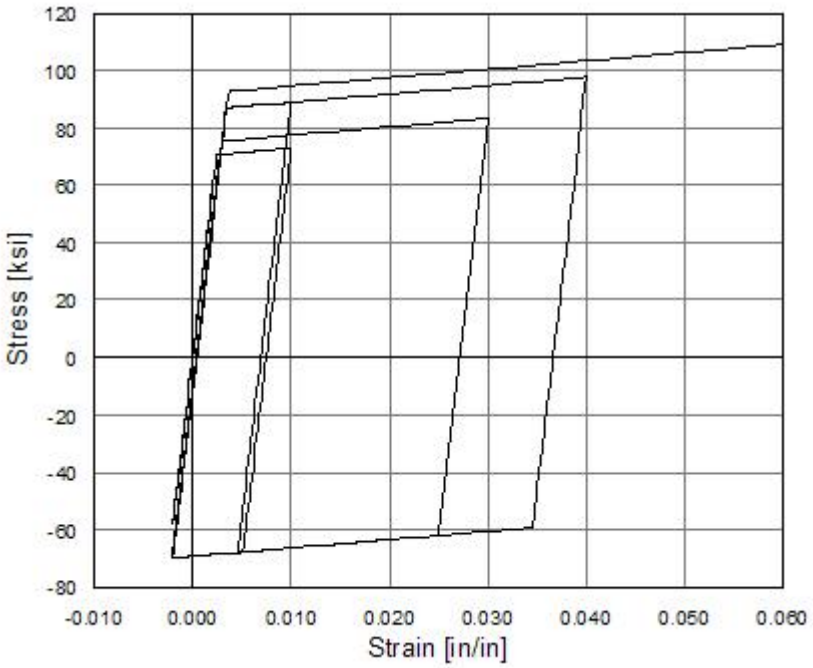


Figure 12: Steel01 Material -- Hysteretic Behavior of Steel_1 Model with Isotropic Hardening in Tension

Steel01 Material -- Hysteretic Behavior of Steel_1 Model with Isotropic Hardening in Tension

Concrete01 Material

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).

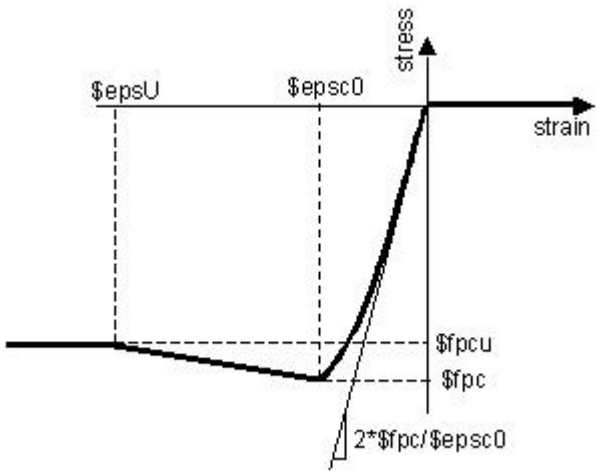
uniaxialMaterial Concrete01 $matTag $fpc $epsc0 $fpcu $epsU

$matTag	unique material object integer tag
$fpc	concrete compressive strength at 28 days (compression is negative)*****
$epsc0	concrete strain at maximum strength*****
$fpcu	concrete crushing strength *****
$epsU	concrete strain at crushing strength*****

***NOTE:** Compressive concrete parameters should be input as negative values.

The initial slope for this model is (2***$fpc**/**$epsc0**)

Figure 13: Concrete01 Material -- Material Parameters



Concrete01 Material -- Material Parameters

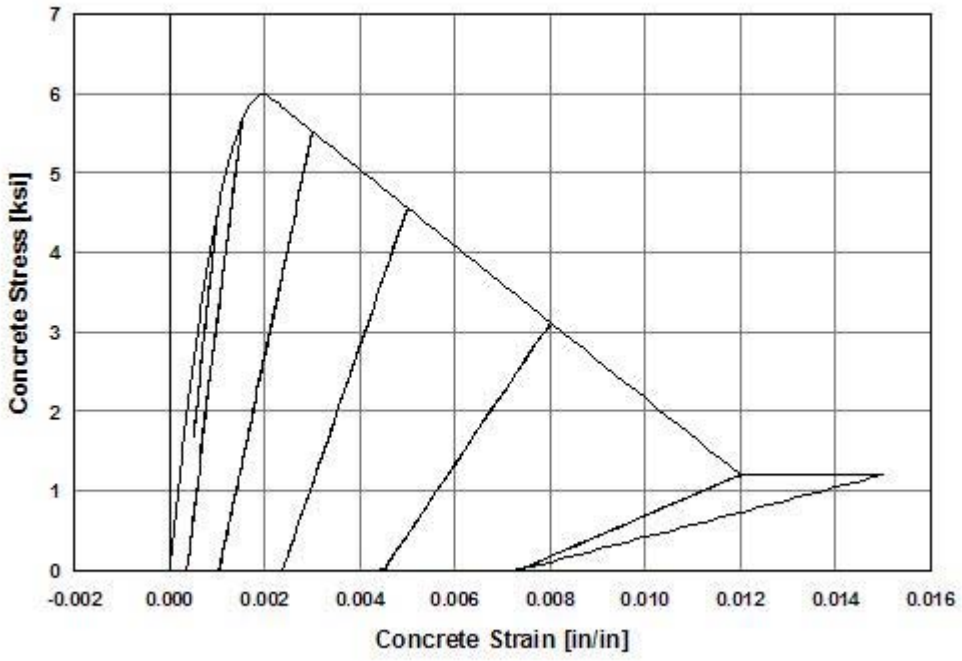

Concrete Stress [ksi]
Concrete Strain [in/in]
7
6
5
4
3
2
1
0
-0.002
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016

Figure 14: Typical Hysteretic Stress-Strain Relation of Concrete_1 Model

Typical Hysteretic Stress-Strain Relation of Concrete_1 Model

Elastic-No Tension Material

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

uniaxialMaterial ENT $matTag $E

$matTag	unique material object integer tag
$E	elastic model in compression

In tension, there is zero stress.

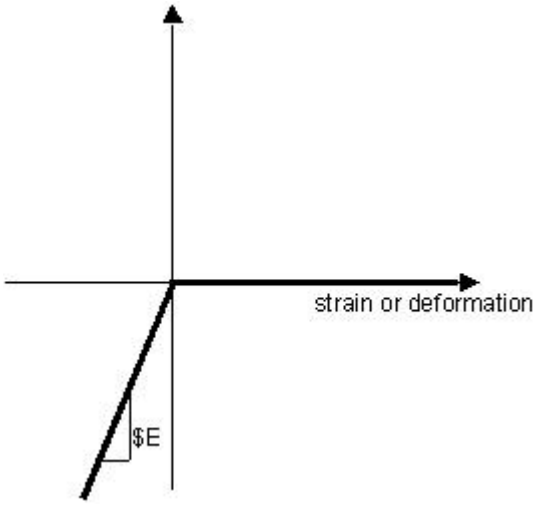


Figure 15: Elastic-No Tension Material

Hysteretic Material

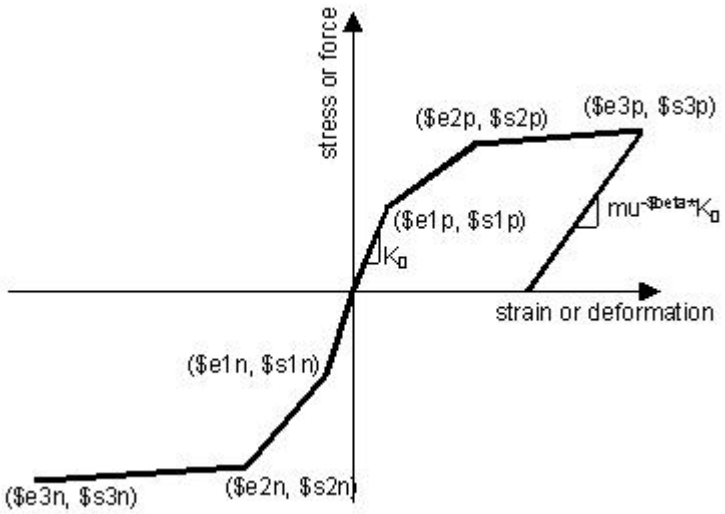
This command is used to construct a uniaxial bilinear hysteretic material object with pinching of force and deformation, damage due to ductility and energy, and degraded unloading stiffness based on ductility.

uniaxialMaterial Hysteretic $matTag $s1p $e1p $s2p $e2p <$s3p $e3p> $s1n $e1n $s2n $e2n <$s3n $e3n> $pinchX $pinchY $damage1 $damage2 <$beta>

$matTag	unique material object integer tag
$s1p $e1p	stress and strain (or force & deformation) at first point of the envelope in the positive direction
$s2p $e2p	stress and strain (or force & deformation) at second point of the envelope in the positive direction
$s3p $e3p	stress and strain (or force & deformation) at third point of the envelope in the positive direction (optional)
$s1n $e1n	stress and strain (or force & deformation) at first point of the envelope in the negative direction*****
$s2n $e2n	stress and strain (or force & deformation) at second point of the envelope in the negative direction*****
$s3n $e3n	stress and strain (or force & deformation) at third point of the envelope in the negative direction (optional)*****
$pinchX	pinching factor for strain (or deformation) during reloading
$pinchY	pinching factor for stress (or force) during reloading
$damage1	damage due to ductility: D_1(mu-1)
$damage2	damage due to energy: $D_2(E_{ii}/E_{ult})$
$beta	power used to determine the degraded unloading stiffness based on ductility, mu^{-beta} (optional, default=0.0)

***NOTE:** negative backbone points should be entered as negative numeric values

Figure 16: Hysteretic Material



Viscous Material

This command is used to construct a uniaxial material object with a non-linear elastic stress-strain-rate relation given by:

stress =C(strain-rate)^{alpha}.

uniaxialMaterial Viscous $matTag $C $alpha

$matTag	unique material object integer tag
$C	tangent
$alpha	damping tangent

PINCHING4 Material

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.

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

$matTag	unique material object integer tag
$ePf1 $ePf2 $ePf3 $ePf4	floating point values defining force points on the positive response envelope
$ePd1 $ePd2 $ePd3 $ePd4	floating point values defining deformation points on the positive response envelope
$eNf1 $eNf2 $eNf3 $eNf4	floating point values defining force points on the negative response envelope (optional, default: negative of positive envelope values)
$eNd1 $eNd2 $eNd3 $eNd4	floating point values defining deformations points on the negative response envelope (optional, default: negative of positive envelope values)
$rDispP	floating point value defining the ratio of the deformation at which reloading occurs to the maximum historic deformation demand
$rForceP	floating point value defining the ratio of the force at which reloading begins to force corresponding to the maximum historic deformation demand
$uForceP	floating point value defining the ratio of strength developed upon unloading from negative load to the maximum strength developed under monotonic loading
$rDispN	floating point value defining the ratio of the deformation at which reloading occurs to the minimum historic deformation demand (optional, default: $rDispP)

$rForceN	floating point value defining the ratio of the force at which reloading begins to the force corresponding to the minimum historic deformation demand (optional, default: $rForceP)
$uForceN	floating point value defining the ratio of the strength developed upon unloading from a positive load to the minimum strength developed under monotonic loading (optional, default: $rForceP)
$gK1 $gK2 $gK3 $gK4 $gKLim	floating point values controlling cyclic degradation model for unloading stiffness degradation
$gD1 $gD2 $gD3 $gD4 $gDLim	floating point values controlling cyclic degradation model for reloading stiffness degradation
$gF1 $gF2 $gF3 $gF4 $gFLim	floating point values controlling cyclic degradation model for strength degradation
$gE	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	string to indicate type of damage (option: “cycle”, “energy”)

NOTE:

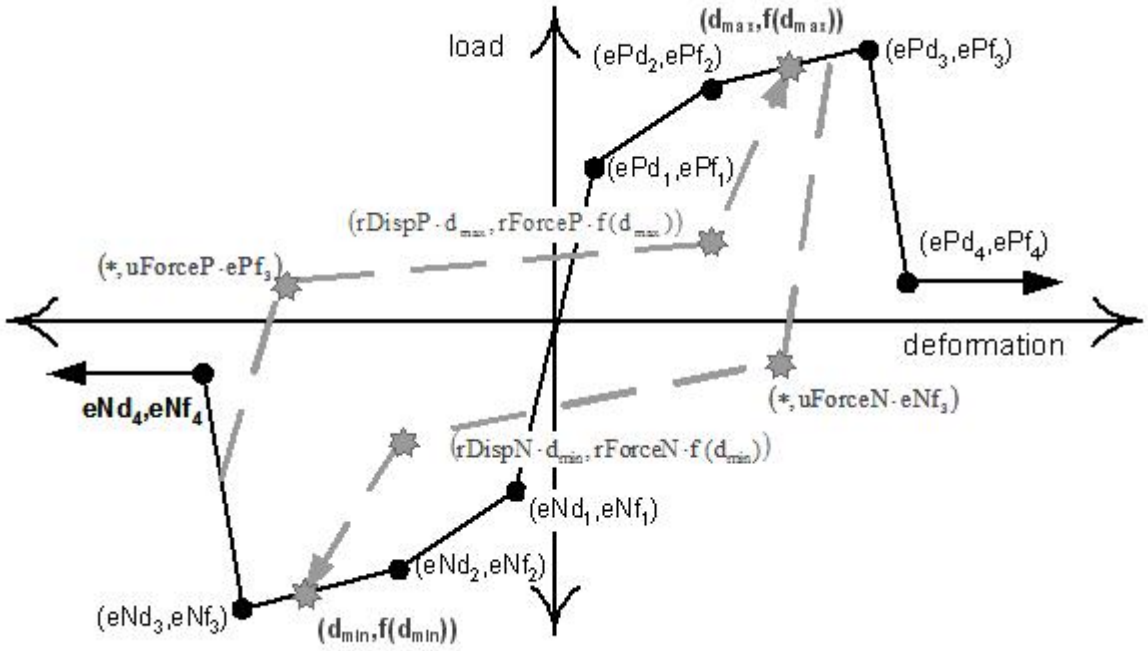


Figure 1: Definition of Pinching4 Uniaxial Material Model

Damage Models:

Stiffness and strength are assumed to deteriorate due to the imposed “load” history. The same basic equations are used to describe deterioration in strength, unloading stiffness and reloading stiffness:

$$k_i = k_0 \cdot (1 - \delta k_i)$$

where k_i is the unloading stiffness at time t_i, k_o is the initial unloading stiffness (for the case of no damage), and δk_i (defined below) is the value of the stiffness damage index at time t_i.

$$d_{\max i} = d_{\max 0} \cdot (1 + \delta d_i)$$

where $d_{\max i}$ is the deformation demand that defines the end of the reload cycle for increasing deformation demand, $d_{\max 0}$ is the maximum historic deformation demand (which would be the deformation demand defining the end of the reload cycle if degradation of reloading stiffness is ignored), and δd_i (defined below) is the value of reloading stiffness damage index at time t_i.

$$(f_{\max})_i = (f_{\max})_0 \cdot (1 - \delta f_i)$$

where $(f_{\max})_i$ is the current envelope maximum strength at time t_i, $(f_{\max})_0$ is the initial envelope maximum strength for the case of no damage, and δf_i (defined below) is the value of strength value index at time t_i.

The damage indices, δk_i, δd_i and δf_i, may be defined to be a function of displacement history only ($dmgType = "cycle") or displacement history and energy accumulation ($dmgType = "energy"). For either case, all of the damage indices are computed using the same basic equation.

If the damage indices are assumed to be a function of displacement history and energy accumulation, the unloading stiffness damage index, δk_i is computed as follows:

$$\delta k_i = \left(gK1 \cdot \left(\tilde{d}_{\max} \right)^{gK3} + gK2 \cdot \left(\frac{E_i}{E_{monotonic}} \right)^{gK4} \right) \leq gKLim$$

where

$$\tilde{d}_{\max} = \max\left[\frac{d_{\max i}}{def_{\max}}, \quad \frac{d_{\min i}}{def_{\min}} \right]$$

$$E_i = \int_{load\ history} dE$$

$$E_{monotonic} = gE \cdot \left(\int_{monotonic\ load\ history} dE \right)$$

with $E_{monotic}$ equal to the energy required to achieve under monotonic loading the deformation that defines failure, def_{max} and def_{min} the positive and negative deformations that define failure. The other damage indices, δd_i and δf_i, are computed using the same equations with degradation model parameters gK* replaced by gF* and gD*, as is appropriate.

The above expressions were meant for “Energy” type damage. The user specification of “Energy” type damage implements damage due to displacement as well as energy. Other type of damage can be activated: “Cycle” which implements damage due to displacement as well as damage accrued due to load cycle counting. The expressions for the “Cycle” damage are given below.

If the damage indices are assumed to be a function only of the displacement history, the unloading stiffness damage index, δk_i is computed as follows:

$$\delta k_i = \left(gK1 \cdot \left(\tilde{d}_{\max}\right)^{gK3} + gK2 \cdot \left(Cycle\right)^{gK4} \right) \le gKLim$$

where

$$\tilde{d}_{\max} = \max\left[\frac{d_{\max i}}{def_{\max}}, \ \frac{d_{\min i}}{def_{\min}} \right]$$

with Cycle equal to the number of cycles accrued in the loading history, def_{max} and def_{min} the positive and negative deformations that define failure. The other damage indices, δd_i and δf_i, are computed using the same equations with degradation model parameters gK* replaced by gF* and gD*, as is appropriate.

- ***EXAMPLE:***

main input file:

- *RCyclicPinch.tcl* (page 67)

supporting files:

- *procUniaxialPinching.tcl* (page 70)
- *procRCycDAns.tcl* (page 71)

PINCHING4 Uniaxial Material Model Discussion

PINCHING4 Uniaxial Material Model Discussion

The example files (*RCyclicPinch.tcl* (page 67), *procUniaxialPinching.tcl* (page 70), *procRCycDAns.tc* (page 71)l) create a one-dimensional structural model consisting of a single truss element of length 1.0 and area 1.0 (Figure 1). The Pinching4 uniaxial material model is used to simulate the stress-strain response of the truss material. The truss is subjected to a pseudo-static cyclic loading. Several files are provided that include different input parameters for the Pinching4 material model and result in different load-displacement histories for the truss structure. Refer to the documentation about the Pinching4 uniaxial material model for additional information.

Input for the Pinching4 Uniaxial Material Model

Refer to the documentation of the Pinching4 uniaxial material model for an explanation of the following notation.

Response Envelopes:

In these examples the pinching material model is demonstrated with two different load-deformation response envelopes. Envelope 1 (Figure 2) defines a hardening-type response while Envelope 2 (Figure 2) defines a softening-type response.

Envelope 1 (Figure 2):

$$\begin{bmatrix} ePd1 & ePf1 \\ ePd2 & ePf2 \\ ePd3 & ePf3 \\ ePd4 & ePf4 \end{bmatrix} = \begin{bmatrix} 0.0001 & 2 \\ 0.0055 & 6 \\ 0.0188 & 7 \\ 0.0189 & 7.2 \end{bmatrix} = \begin{bmatrix} -eNd1 & -eNf1 \\ -eNd2 & -eNf2 \\ -eNd3 & -eNf3 \\ -eNd4 & -eNf4 \end{bmatrix}$$

Envelope 2 (Figure 2):

$$\begin{bmatrix} ePd1 & ePf1 \\ ePd2 & ePf2 \\ ePd3 & ePf3 \\ ePd4 & ePf4 \end{bmatrix} = \begin{bmatrix} 0.0001 & 2 \\ 0.0055 & 6 \\ 0.0188 & 7 \\ 0.0189 & 0.2 \end{bmatrix} = \begin{bmatrix} -eNd1 & -eNf1 \\ -eNd2 & -eNf2 \\ -eNd3 & -eNf3 \\ -eNd4 & -eNf4 \end{bmatrix}$$

Load-Unload Response Parameters:

The form of the load-unload response, and the extent of pinching in the response history, is defined by the following six parameters. In each of the examples, the following values are used.

$$\begin{bmatrix} rDispP & rForceP & uForceP \end{bmatrix} = \begin{bmatrix} rDispN & rForceN & uForceN \end{bmatrix} = \begin{bmatrix} 0.5 & 0.25 & 0.05 \end{bmatrix}$$

Strength and Stiffness Degradation Parameters:

The Pinching4 uniaxial material model simulates degradation of stiffness and strength under cyclic loading. The example files demonstrate, individually and in combination, each of the two stiffness degradation options and the one strength degradation option. The following parameters are used to define strength and stiffness degradation, as needed, in the example files.

$$\begin{bmatrix} gK1 & gK2 & gK3 & gK4 & gKLim \\ gD1 & gD2 & gD3 & gD4 & gDLim \\ gF1 & gF2 & gF3 & gF4 & gFLim \\ gE & & & & \end{bmatrix} = \begin{bmatrix} 1.0 & 0.2 & 0.3 & 0.2 & 0.9 \\ 0.5 & 0.5 & 2.0 & 2.0 & 0.5 \\ 1.0 & 0.0 & 1.0 & 1.0 & 0.9 \\ 10.0 & & & & \end{bmatrix}$$

Tcl Scripts:

The following tcl script files are used to run the examples:

RCyclicPinch.tcl (page 67)

procUniaxialPinching.tcl (page 70)

procRCycDAns.tcl (page 71)

Lines should be commented out as necessary within RCyclicPinch.tcl to exercise different degradation models of the Pinching4 uniaxial material model and different load histories.

If the structure is subjected to a monotonic load history, the load-displacement history is shown in Figure 2 results with the actual response history depending on the envelope chosen for the material model. If the structure is subjected to a cyclic load history (Figure 3), one of the load-displacement histories shown in Figure 4 results, with the exact response depending on the strength and stiffness degradation model employed. In this case the damage type activated was “Energy”. The case in which the damage type activated was “Cycle” is shown in Figure 5.

Figures:

Figure 17: Structural Model

Figure 1: Structural Model

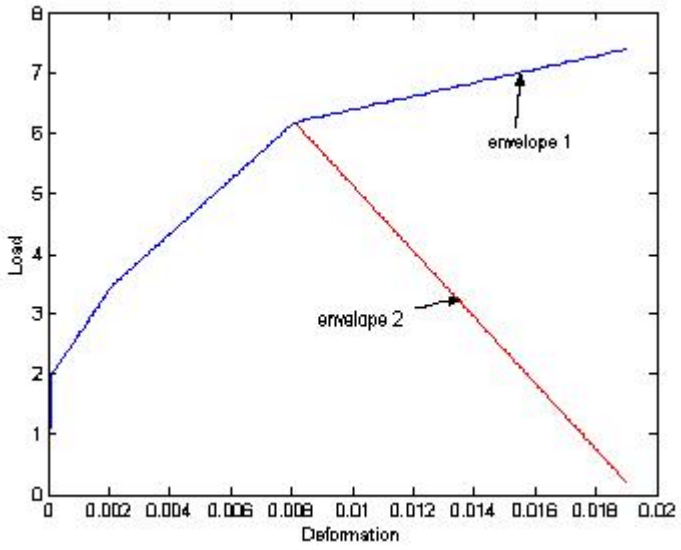


Figure 18: Response Envelopes

Figure 2: Response Envelopes

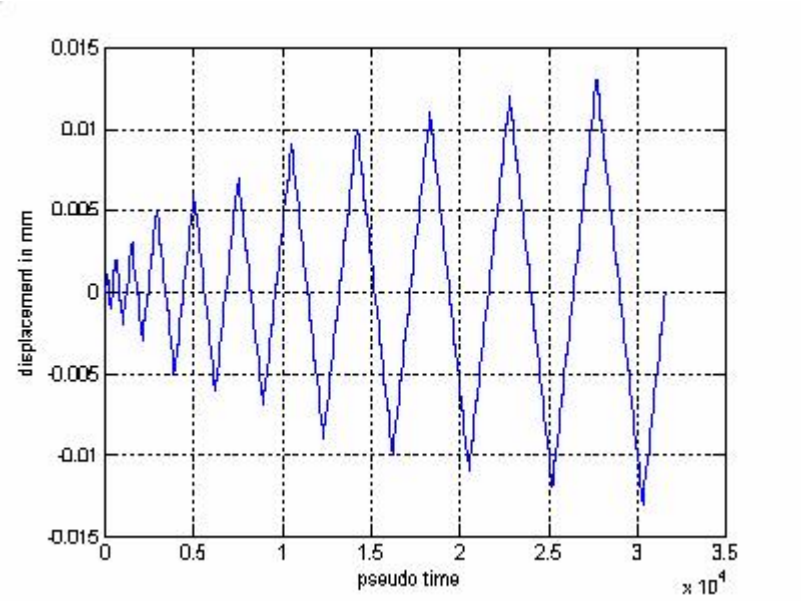
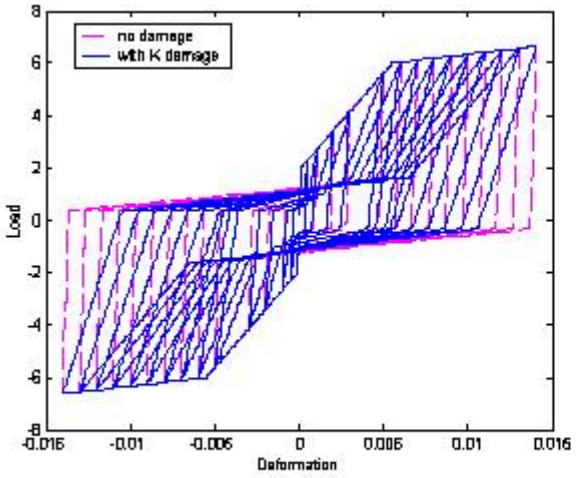
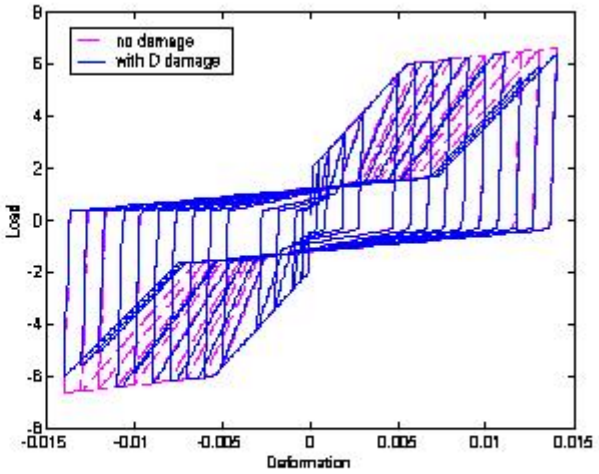


Figure 19: Cyclic Displacement History

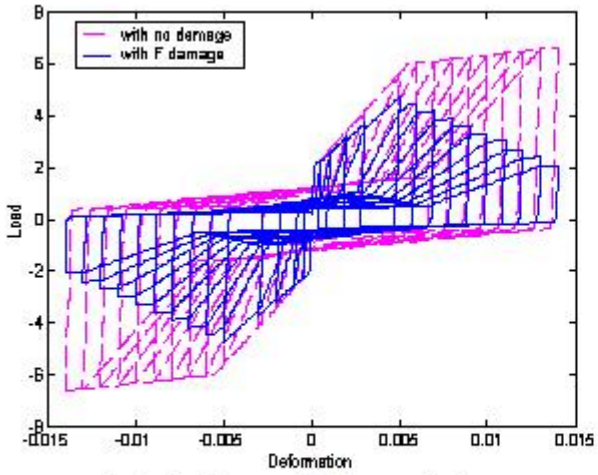
Figure 3: Cyclic Displacement History



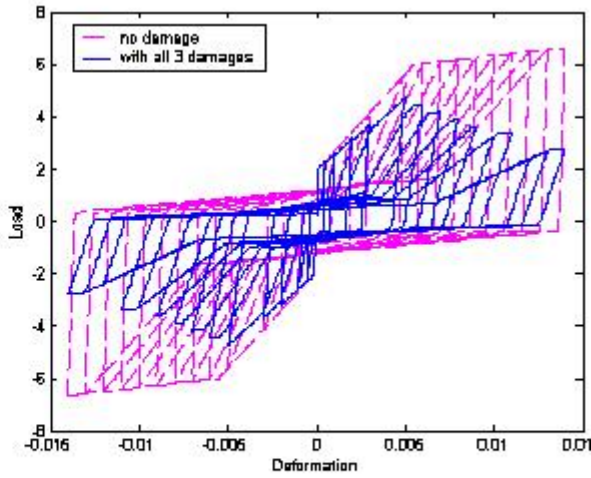
a) Only Unloading Stiffness Degradation



b) Only Re-Loading Stiffness Degradatio



c) Only Strength Degradation



d) Both Stiffness and Strength Degradati

Figure 20: Load-Deformation Response Histories ("Energy type damage")

Figure 4: Load-Deformation Response Histories (“Energy type damage”)

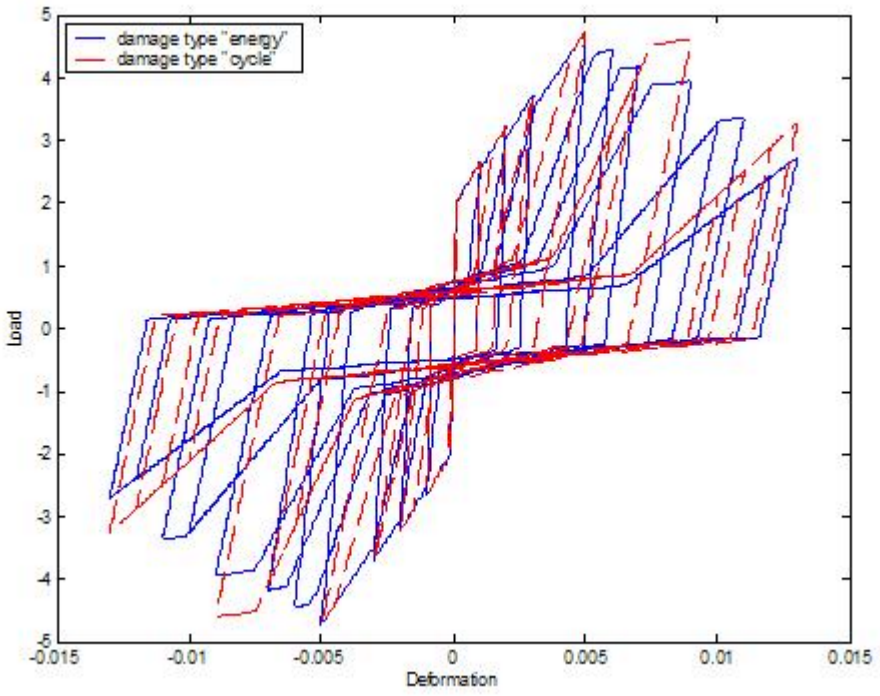


Figure 21: Plot showing Both Stiffness and Strength Degradation for Damage Type "Energy" and "Cycle"

Figure 5. Plot showing Both Stiffness and Strength Degradation for Damage Type "Energy" and "Cycle"

RCyclicPinch.tcl

```
##########################################################################################################################
# Test example for PINCHING MATERIAL                                                                                     #
```

```
# Written: N.Mitra                                                                    #
# Description: uniaxial material with user defined envelope (softening type used here) and damage parameters    #
# Date: May 04 2002                                                                   #
## Model subjected to reverse Cyclic Loading                                          #
## File Name: RCyclicPinch.tcl                                                        #
# refer to Pinching-Type Material Model.doc for full explanation of the parameters    #
#########################################################################################################
#create the ModelBuilder object
model BasicBuilder -ndm 2 -ndf 2

# add nodes - command: node nodeId xCrd yCrd
node 1   0.0  0.0
node 2   1.0  0.0

## please keep the follwoing procedures on the same path
source procUniaxialPinching.tcl
source procRCycDAns.tcl

##### Positive/Negative envelope Stress/Load
###                    stress1 stress2 stress3 stress4
set pEnvelopeStress [list   2.0     6.0      7.0     0.2]
set nEnvelopeStress [list   -2.0    -6.0     -7.0    -0.2]

##### Positive/Negative envelope Strain/Deformation
###                     strain1  strain2 strain3  strain4
set pEnvelopeStrain [list  0.0001  0.0055   0.0188   0.0189]
set nEnvelopeStrain [list  -0.0001  -0.0055   -0.0188   -0.0189]

##### Ratio of maximum deformation at which reloading begins
###            Pos_env.   Neg_env.
set rDisp [list   0.5      0.5]

##### Ratio of envelope force (corresponding to maximum deformation) at which reloading begins
###            Pos_env.   Neg_env.
set rForce [list 0.25       0.25]
```

```
##### Ratio of monotonic strength developed upon unloading
###            Pos_env.   Neg_env.
set uForce [list 0.05       0.05]

##### Coefficients for Unloading Stiffness degradation
###             gammaK1   gammaK2  gammaK3  gammaK4   gammaKLimit
set gammaK [list   1.0       0.2       0.3     0.2        0.9]
#set gammaK [list   0.0       0.0       0.0     0.0        0.0]

##### Coefficients for Reloading Stiffness degradation
###             gammaD1   gammaD2  gammaD3  gammaD4   gammaDLimit
set gammaD [list    0.5       0.5      2.0      2.0       0.5]
#set gammaD [list    0.0       0.0      0.0      0.0       0.0]

##### Coefficients for Strength degradation
###             gammaF1   gammaF2  gammaF3  gammaF4   gammaFLimit
set gammaF [list   1.0       0.0      1.0       1.0       0.9]
#set gammaF [list   0.0       0.0      0.0       0.0       0.0]

set gammaE 10

# material ID
set matID 1

# damage type (option: "energy", "cycle")
set dam "energy"

# add the material to domain through the use of a procedure
procUniaxialPinching $matID  $pEnvelopeStress $nEnvelopeStress $pEnvelopeStrain $nEnvelopeStrain $rDisp
$rForce $uForce $gammaK $gammaD $gammaF $gammaE $dam

# add truss elements - command: element truss trussID node1 node2 A matID
element truss 1 1 2 1.0 1

# set the boundary conditions - command: fix nodeID xResrnt? yRestrnt?
fix 1 1 1
fix 2 0 1
```

```
pattern Plain 1 Linear {
    load 2 1 0
}

recorder Node RCyclicPinchR.out disp -load -node 2  -dof 1

# build the components for the analysis object
system ProfileSPD
constraints Plain
test NormDispIncr 1.0e-8 20
algorithm Newton
numberer RCM

## analysis type used in the procedure is Static

set peakpts [list 0.0001 0.001 0.002 0.003 0.005 0.006 0.007 0.009 0.01 0.011 0.012 0.013 ]
set increments 10
set nodeTag 2
set dofTag 1

## start procedure for feeding in
## Reverse Cyclic loading to the model by Disp. control
procRCycDAns $increments $nodeTag $dofTag $peakpts

# print the results at nodes
print node
```

procUniaxialPinching.tcl

```
##########################################################################################################
########
#                                                                                #
#                          procUniaxialPinching.tcl                              #
# procedure for activating the pinching material given its parameters in the form of list          #
# created NM (nmitra@u.washington.edu) dated : Feb 2002                                  #
##########################################################################################################
########
proc procUniaxialPinching { materialTag pEnvelopeStress nEnvelopeStress pEnvelopeStrain nEnvelopeStrain rDisp
rForce uForce gammaK gammaD gammaF gammaE damage} {
```

```
# add material - command: uniaxialMaterial ..... paramaters as shown
#uniaxialMaterial Pinching4 tag
####          stress1P strain1P stress2P strain2P stress3P strain3P stress4P strain4P
####          stress1N strain1N stress2N strain2N stress3N strain3N stress4N strain4N
####          rDispP rForceP uForceP rDispN rForceN uForceN
####          gammaK1 gammaK2 gammaK3 gammaK4 gammaKLimit
####          gammaD1 gammaD2 gammaD3 gammaD4 gammaDLimit
####          gammaF1 gammaF2 gammaF3 gammaF4 gammaFLimit gammaE $damage


uniaxialMaterial Pinching4 $materialTag [lindex $pEnvelopeStress 0] [lindex $pEnvelopeStrain 0] \
 [lindex $pEnvelopeStress 1] [lindex $pEnvelopeStrain 1] [lindex $pEnvelopeStress 2] \
 [lindex $pEnvelopeStrain 2] [lindex $pEnvelopeStress 3] [lindex $pEnvelopeStrain 3] \
 [lindex $nEnvelopeStress 0] [lindex $nEnvelopeStrain 0] \
 [lindex $nEnvelopeStress 1] [lindex $nEnvelopeStrain 1] [lindex $nEnvelopeStress 2] \
 [lindex $nEnvelopeStrain 2] [lindex $nEnvelopeStress 3] [lindex $nEnvelopeStrain 3] \
 [lindex $rDisp 0] [lindex $rForce 0] [lindex $uForce 0] \
 [lindex $rDisp 1] [lindex $rForce 1] [lindex $uForce 1] \
 [lindex $gammaK 0] [lindex $gammaK 1] [lindex $gammaK 2] [lindex $gammaK 3] [lindex $gammaK 4] \
 [lindex $gammaD 0] [lindex $gammaD 1] [lindex $gammaD 2] [lindex $gammaD 3] [lindex $gammaD 4] \
 [lindex $gammaF 0] [lindex $gammaF 1] [lindex $gammaF 2] [lindex $gammaF 3] [lindex $gammaF 4] \
 $gammaE $damage


}
```

procRCycDAns.tcl

```
##########################################################################################################
########
#                                                                       #
#                       procRCycDAns.tcl                                #
#  procedure for reverse cyclic displacement control analysis given the peak pts.          #
# analysis type used : STATIC                                           #
# Written : N.Mitra                                                     #
##########################################################################################################
########
proc procRCycDAns { incre nodeTag dofTag peakpts} {
```

```
set x [lindex $peakpts 0]
set fir [expr $x/$incre]

integrator DisplacementControl $nodeTag $dofTag 0.0 1 $fir $fir

# create the analysis object
analysis Static
# perform the analysis
analyze $incre
integrator DisplacementControl $nodeTag $dofTag 0.0 1 [expr -$fir] [expr -$fir]
analyze [expr 2*$incre]
integrator DisplacementControl $nodeTag $dofTag 0.0 1 $fir $fir
analyze $incre

for {set j 1} {$j < [llength $peakpts]} {incr j 1} {
  set tx [lindex $peakpts $j]
  set tinc [expr $tx/$fir]
  set rt [expr int($tinc)]

  integrator DisplacementControl $nodeTag $dofTag 0.0 1 $fir $fir
  analyze $rt
  integrator DisplacementControl $nodeTag $dofTag 0.0 1 [expr -$fir] [expr -$fir]
  analyze [expr 2*$rt]
  integrator DisplacementControl $nodeTag $dofTag 0.0 1 $fir $fir
  analyze $rt
 }
################################### end procRCycDAns.tcl ########################################
}
```

Fedeas Materials

This section lists the uniaxial material objects available from the Fedeas ML1D library developed by F.C. Filippou. For more information see the Fedeas materials web page:

http://www.ce.berkeley.edu/~filippou/Research/Fedeas/material.htm
(http://www.ce.berkeley.edu/~filippou/Research/Fedeas/material.htm)

Further information on the *Concrete01* (page 47) and *Steel01* (page 43) materials described earlier in this document can also be found at this web page. Currently, each of the following Fedeas materials are available only on the Win32 version of OpenSees

Concrete02 Material

This command is used to construct a uniaxial concrete material object with tensile strength and linear tension softening.

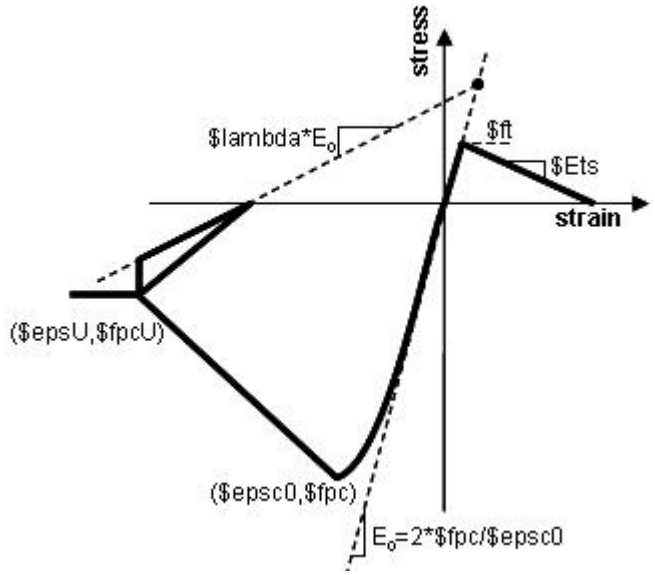
uniaxialMaterial Concrete02 $matTag $fpc $epsc0 $fpcu $epscu $lambda $ft $Ets

$matTag	unique material object integer tag
$fpc	compressive strength*****
$epsc0	strain at compressive strength*****
$fpcu	crushing strength*****
$epsU	strain at crushing strength*****
$lambda	ratio between unloading slope at $epscu and initial slope
$ft	tensile strength
$Ets	tension softening stiffness (absolute value) (slope of the linear tension softening branch)

***NOTE:** Compressive concrete parameters should be input as negative values.

The initial slope for this model is (2***$fpc/$epsc0**)

Figure 22: Concrete02 Material -- Material parameters



Concrete02 Material -- Material parameters

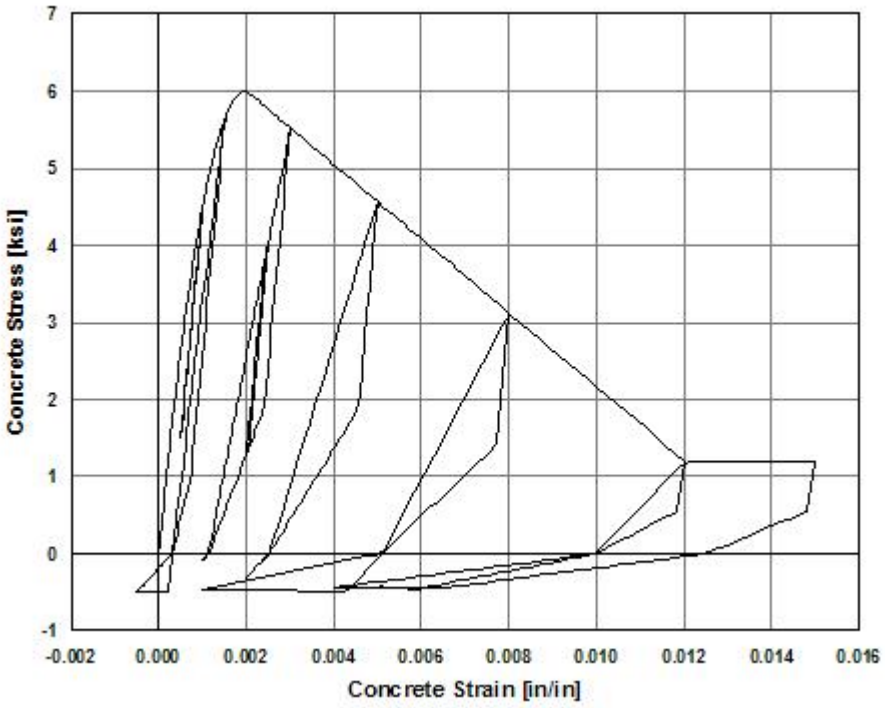

Concrete Stress [ksi]
7
6
5
4
3
2
1
0
-1
-0.002
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
Concrete Strain [in/in]

Figure 23: Typical Hysteretic Stress-Strain Relation of Concrete_2 Model

Typical Hysteretic Stress-Strain Relation of Concrete_2 Model

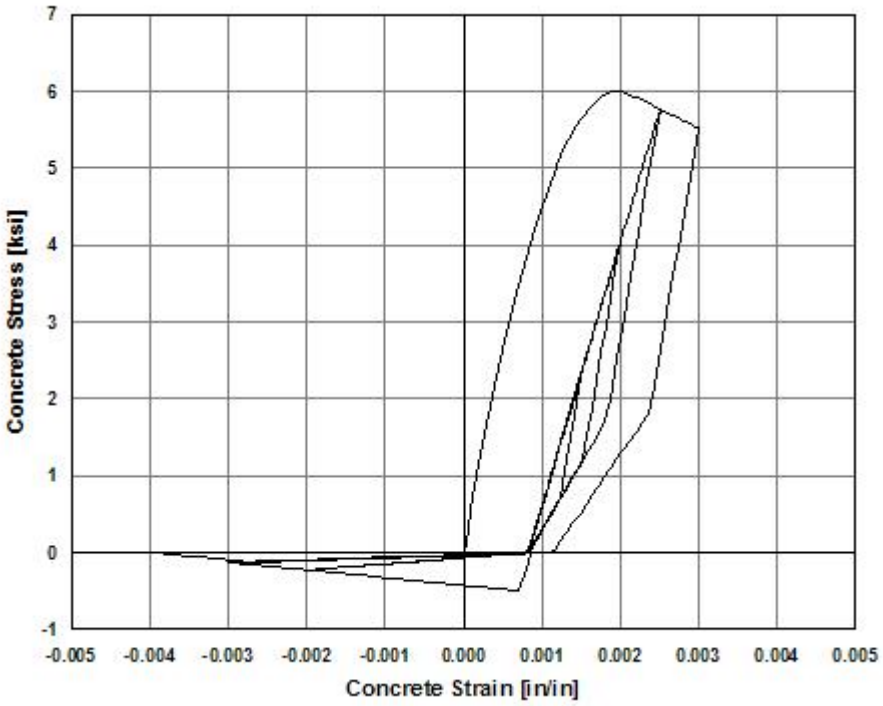


Figure 24: Hysteretic Stress-Strain Relation of Concrete_2 Model in Tension-Compression

Hysteretic Stress-Strain Relation of Concrete_2 Model in Tension-Compression

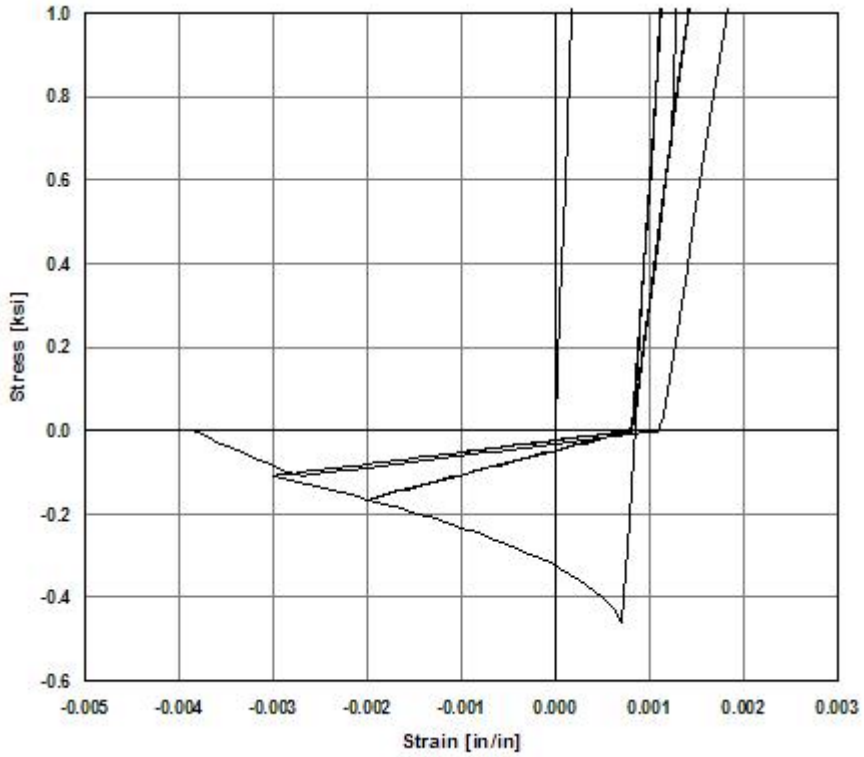

1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
Stress [ksi]
-0.005
-0.004
-0.003
-0.002
-0.001
0.000
0.001
0.002
0.003
Strain [in/in]

Figure 25: Hysteretic Stress-Strain Relation of Concrete_2 Model in Tension-Compression (Detail)

Hysteretic Stress-Strain Relation of Concrete_2 Model in Tension-Compression (Detail)

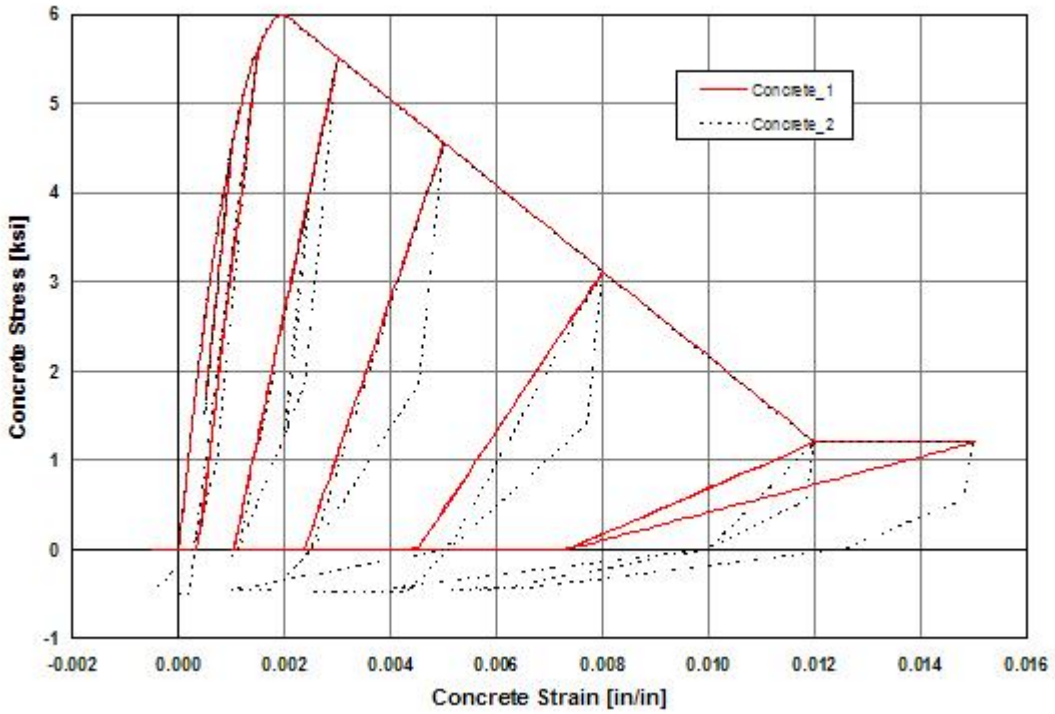


Figure 26: Comparison of Hysteretic Behavior of Concrete_1 and Concrete_2 model

Comparison of Hysteretic Behavior of Concrete_1 and Concrete_2 model

Concrete03 Material

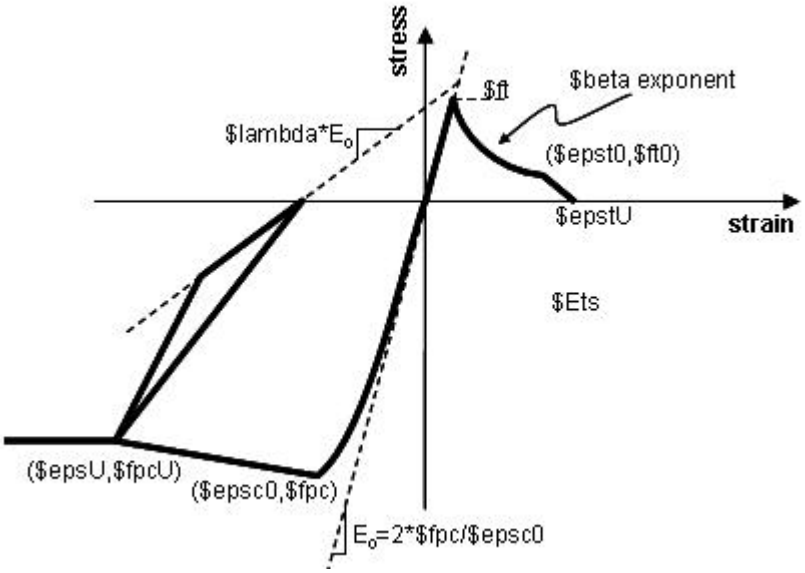
This command is used to construct a uniaxial concrete material object with tensile strength and nonlinear tension softening.

uniaxialMaterial Concrete03 $matTag $fpc $epsc0 $fpcu $epscu $lambda $ft $epst0 $ft0 $beta $epstu

$matTag	unique material object integer tag
$fpc	compressive strength*****
$epsc0	strain at compressive strength*****
$fpcu	crushing strength*****
$epsU	strain at crushing strength*****
$lambda	ratio between unloading slope at $epscu and initial slope (=2***$fpc**/**$epsc0**)
$ft	tensile strength
$epst0	tensile strain at the transition from nonlinear to linear softening
$ft0	tensile stress at the transition from nonlinear to linear softening
$beta	exponent of the tension softening curve
$epstu	ultimate tensile strain

***NOTE:** Compressive concrete parameters should be input as negative values.

Figure 27: Concrete03 Material -- Material Parameters



Concrete03 Material -- Material Parameters

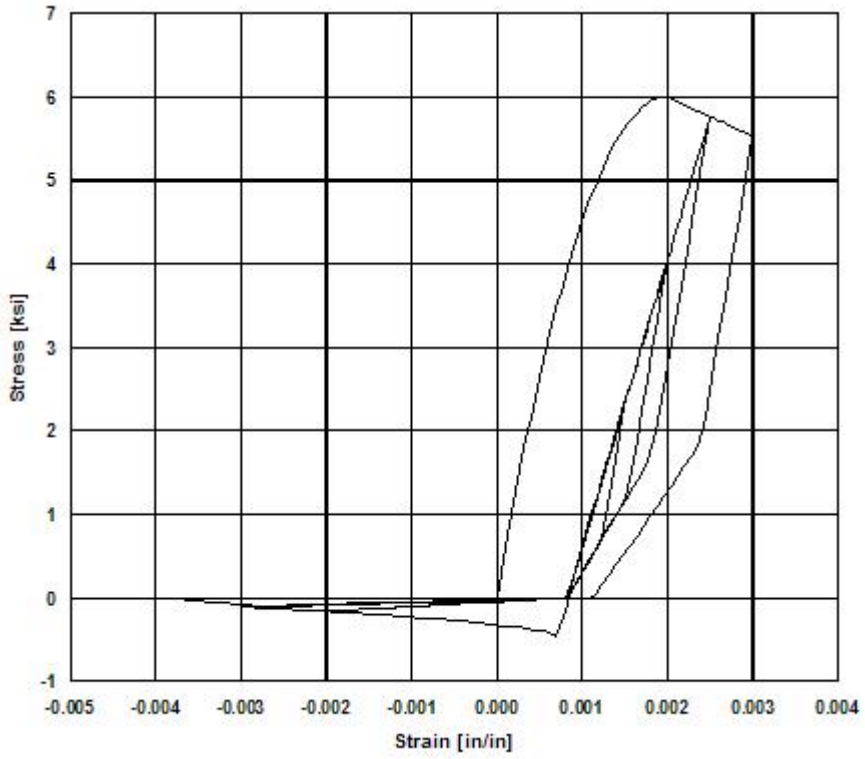

7
6
5
4
3
2
1
0
-1
Stress [ksi]
-0.005
-0.004
-0.003
-0.002
-0.001
0.000
0.001
0.002
0.003
0.004
Strain [in/in]

Figure 28: Hysteretic stress-strain relation of Concrete_3 model in Tension-Compression

Hysteretic stress-strain relation of Concrete_3 model in Tension-Compression

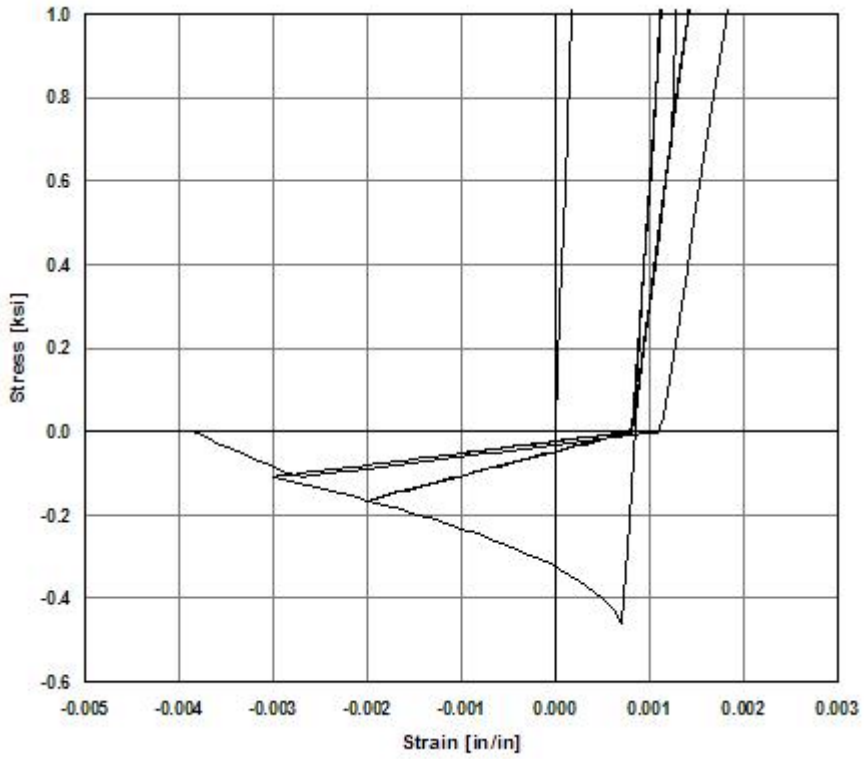


Figure 29: Hysteretic stress-strain relation of Concrete_3 model in Tension-Compression (Detail)

Hysteretic stress-strain relation of Concrete_3 model in Tension-Compression (Detail)

Steel02 Material -- Giuffré-Menegotto-Pinto Model with Isotropic Strain Hardening

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

uniaxialMaterial Steel02 $matTag $Fy $E $b $R0 $cR1 $cR2 $a1 $a2 $a3 $a4

$matTag	unique material object integer tag
$Fy	yield strength
$E	initial elastic tangent
$b	strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
$R0, $cR1, $cR2	control the transition from elastic to plastic branches. Recommended values: **$R0**=between 10 and 20, **$cR1**=0.925, **$cR2**=0.15
$a1, $a2, $a3, $a4	isotropic hardening parameters: (optional, default: no isotropic hardening) **$a1** isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of $a2*($Fy/E0). **$a2** isotropic hardening parameter (see explanation under $a1) **$a3** isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of $a4*($Fy/E0) **$a4** isotropic hardening parameter (see explanation under $a3)

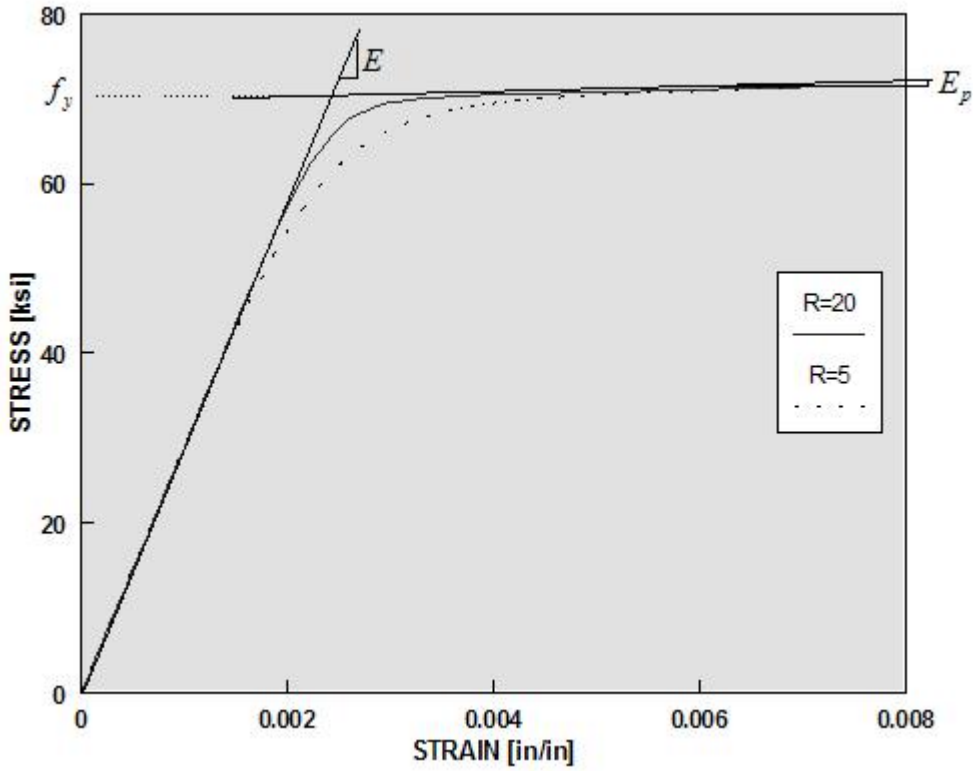

80
60
40
20
0
f_y
E
E_p
STRESS [ksi]
0
0.002
0.004
0.006
0.008
STRAIN [in/in]
R=20
R=5

Figure 30: Steel02 Material -- Material Parameters of Monotonic Envelope

Steel02 Material -- Material Parameters of Monotonic Envelope

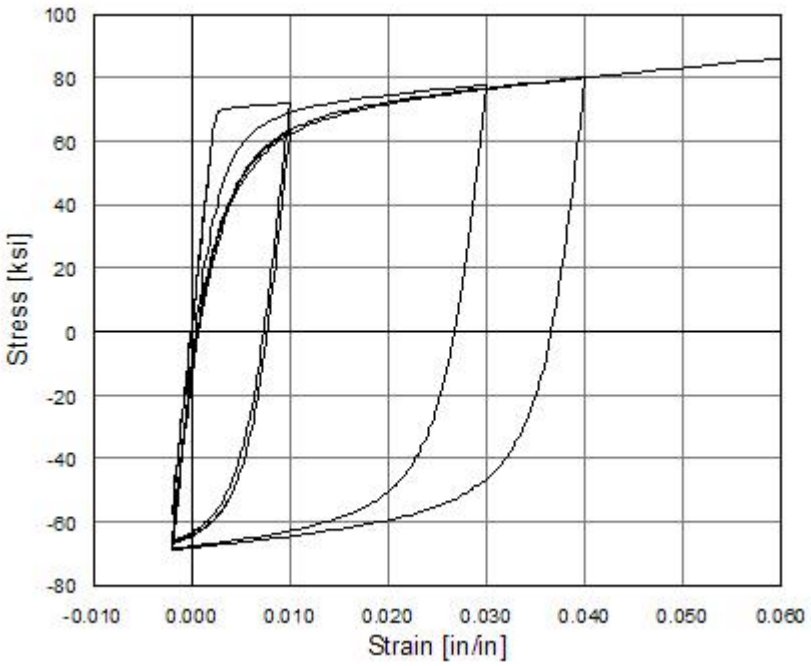


Figure 31: Steel02 Material -- Hysteretic Behavior of Model w/o Isotropic Hardening

Steel02 Material -- Hysteretic Behavior of Model w/o Isotropic Hardening

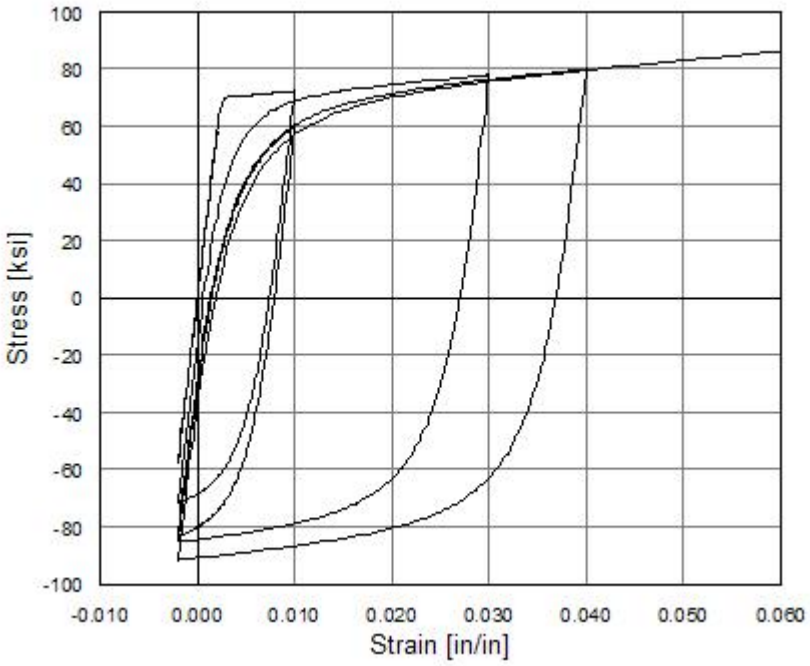

100
80
60
40
20
0
-20
-40
-60
-80
-100
Stress [ksi]
-0.010
0.000
0.010
0.020
0.030
0.040
0.050
0.060
Strain [in/in]

Figure 32: Steel02 Material -- Hysteretic Behavior of Model with Isotropic Hardening in Compression

Steel02 Material -- Hysteretic Behavior of Model with Isotropic Hardening in Compression

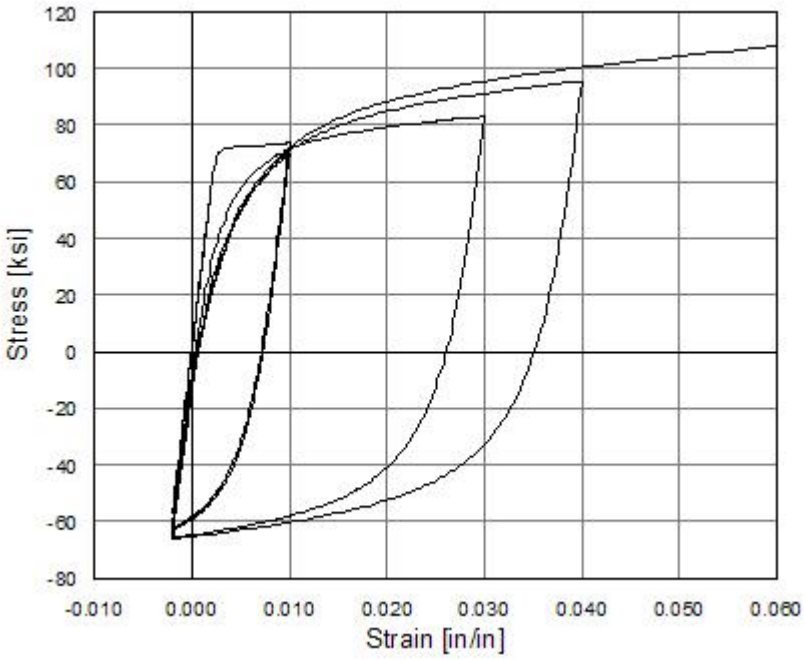


Figure 33: Steel02 Material -- Hysteretic Behavior of Model with Isotropic Hardening in Tension

Steel02 Material -- Hysteretic Behavior of Model with Isotropic Hardening in Tension

Bond01 Material

This command is used to construct an Eligehousen bond material object without damage.(REF: Fedeas).

uniaxialMaterial Bond01 $matTag $u1p $q1p $u2p $u3p $q3p $u1n $q1n $u2n $u3n $q3n $s0 $bb

Tensile bond-slip backbone parameter

$matTag		unique material object integer tag
$u1p	**$q1p**	slip and bond at first detachment -- tensile bond-slip backbone
$u2p		slip at start of degradation -- tensile bond-slip backbone
$u3p	**$q3p**	slip and bond at ultimate -- tensile bond-slip backbone
$u1n	**$q1n**	slip and bond at first detachment -- compressive bond-slip backbone*****
$u2n		slip at start of degradation -- compressive bond-slip backbone*****
$u3n	**$q3n**	slip and bond at ultimate -- compressive bond-slip backbone*****
$s0		unloading stiffness
$bb		exponent for the first branch of the backbone (prior to first detachment). i.e. $q=u^{bb}$

***NOTE:** Compressive concrete parameters should be input as negative values.

Bond02 Material

This command is used to construct an Eligehousen bond material object with damage.

uniaxiaMaterial Bond02 $matTag $u1p $q1p $u2p $u3p $q3p $u1n $q1n $u2n $u3n $q3n $s0 $bb $alp $aln

$matTag		unique material object integer tag
$u1p	**$q1p**	slip and bond at first detachment -- tensile bond-slip backbone

$u2p		slip at start of degradation -- tensile bond-slip backbone
$u3p	**$q3p**	slip and bond at ultimate -- tensile bond-slip backbone
$u1n	**$q1n**	slip and bond at first detachment -- compressive bond-slip backbone*****
$u2n		slip at start of degradation -- compressive bond-slip backbone*****
$u3n	**$q3n**	slip and bond at ultimate -- compressive bond-slip backbone*****
$s0		unloading stiffness
$bb		exponent for the first branch of the backbone (prior to first detachment). i.e. $q=u^{\$bb}$
$alp		damage factor for positive quadrant
$aln		damage factor for negative quadrant

***NOTE:** Compressive concrete parameters should be input as negative values.

Hyster_1: Bilinear Hysteretic Model with Damage

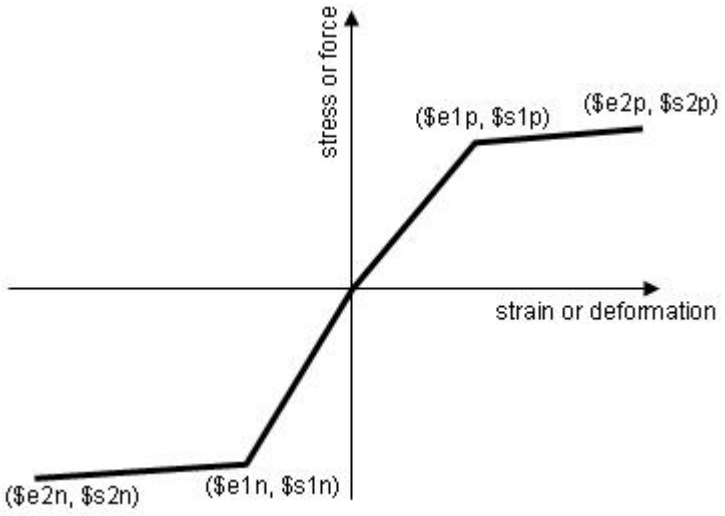
This command is used to construct a uniaxial bilinear hysteretic material object with pinching of force and deformation, damage due to ductility and energy.

uniaxialMaterial Hyster_1 $matTag $s1p $e1p $s2p $e2p $s1n $e1n $s2n $e2n $pinchX $pinchY $dDELTA $dE

$matTag		unique material object integer tag
$s1p	**$e1p**	stress and strain (or force & deformation) at first point of the envelope in the positive direction
$s2p	**$e2p**	stress and strain (or force & deformation) at second point of the envelope in the positive direction
$s1n	**$e1n**	stress and strain (or force & deformation) at first point of the envelope in the negative direction*****
$s2n	**$e2n**	stress and strain (or force & deformation) at second point of the envelope in the negative direction*****
$pinchX		pinching factor for strain (or deformation) during reloading
$pinchY		pinching factor for stress (or force) during reloading
$dDELTA		damage due to ductility: D_1(mu-1)

$dE damage due to energy: $D_2(E_{ii}/E_{ult})$

Figure 34: Hyster_1 Material -- Material Parameters of Monotonic Envelope



Hyster_1 Material -- Material Parameters of Monotonic Envelope

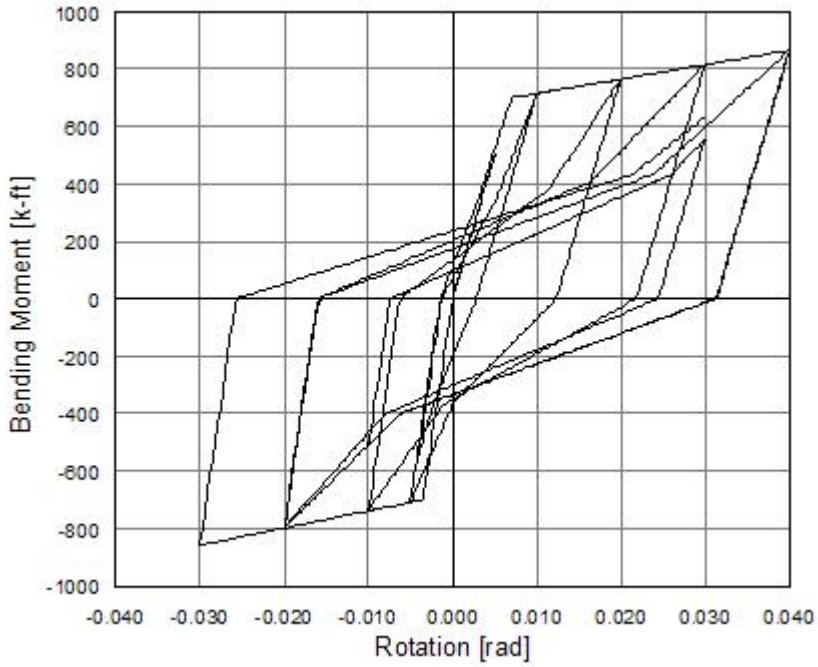

1000
800
600
400
200
0
-200
-400
-600
-800
-1000
Bending Moment [k-ft]
-0.040
-0.030
-0.020
-0.010
0.000
0.010
0.020
0.030
0.040
Rotation [rad]

Figure 35: Hyster_1 Material -- Hysteretic Behavior of Model (hardening; no damage)

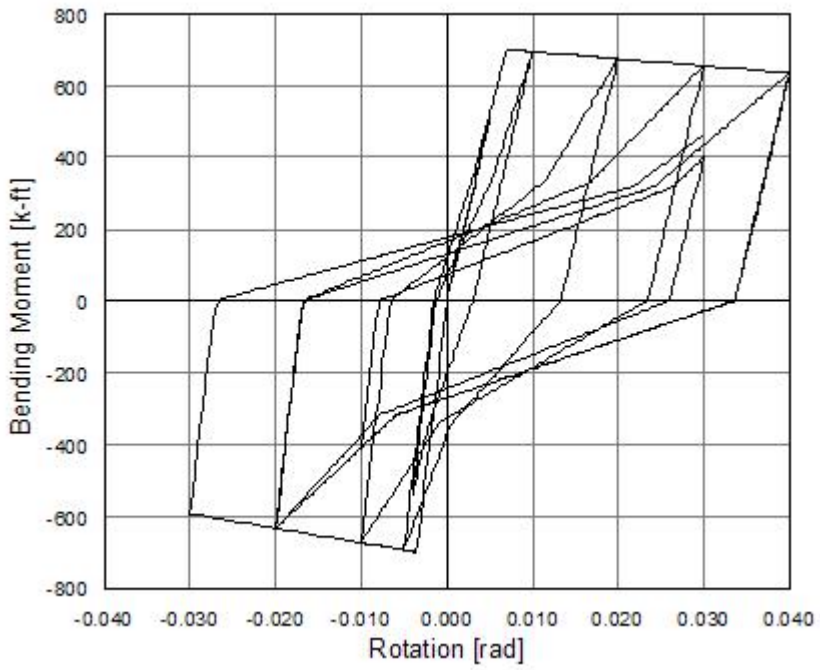


Figure 36: Hyster_1 Material -- Hysteretic Behavior of Model (softening; no damage)

Hyster_1 Material -- Hysteretic Behavior of Model (softening; no damage)

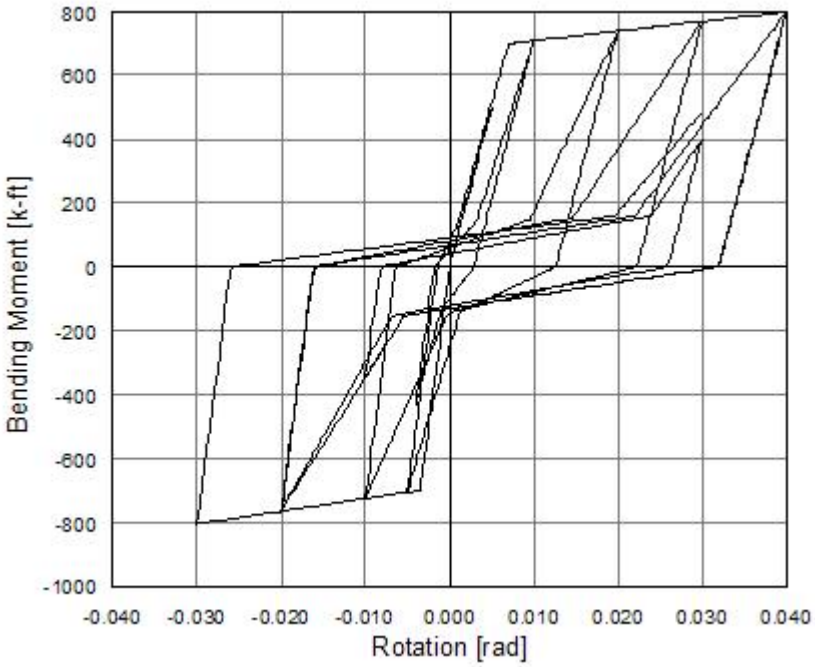

800
600
400
200
0
-200
-400
-600
-800
-1000
Bending Moment [k-ft]
-0.040
-0.030
-0.020
-0.010
0.000
0.010
0.020
0.030
0.040
Rotation [rad]

Figure 37: Hyster_1 Material -- Hysteretic Behavior of Model (no damage)(hardening and "pinching" pinchX=0.7 pinchY=0.2)

Hyster_1 Material -- Hysteretic Behavior of Model (no damage)(hardening and "pinching" pinchX=0.7 pinchY=0.2)

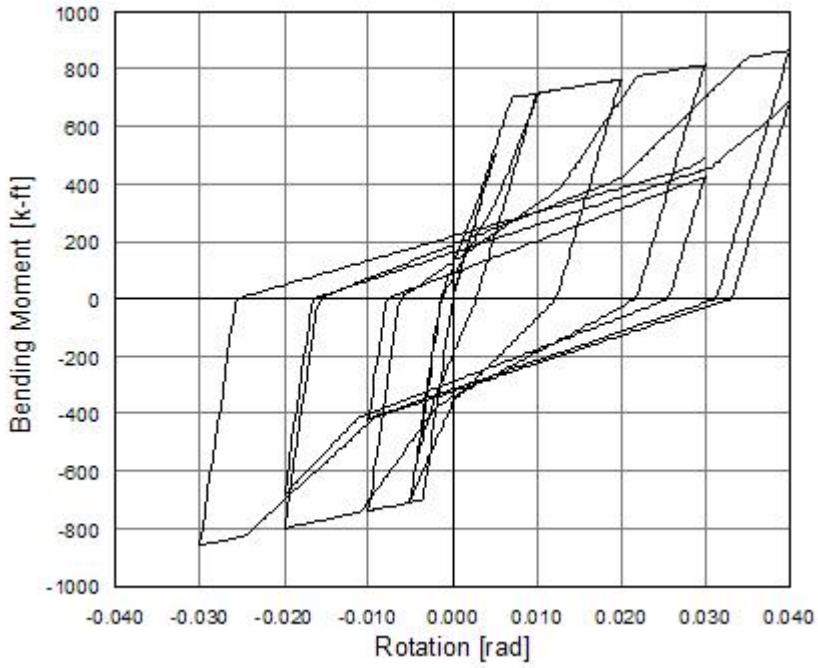


Figure 38: Hyster_1 Material -- Hysteretic Behavior of Model (hardening; displacement damage dDELTA=0.05)

Hyster_1 Material -- Hysteretic Behavior of Model (hardening; displacement damage dDELTA=0.05)

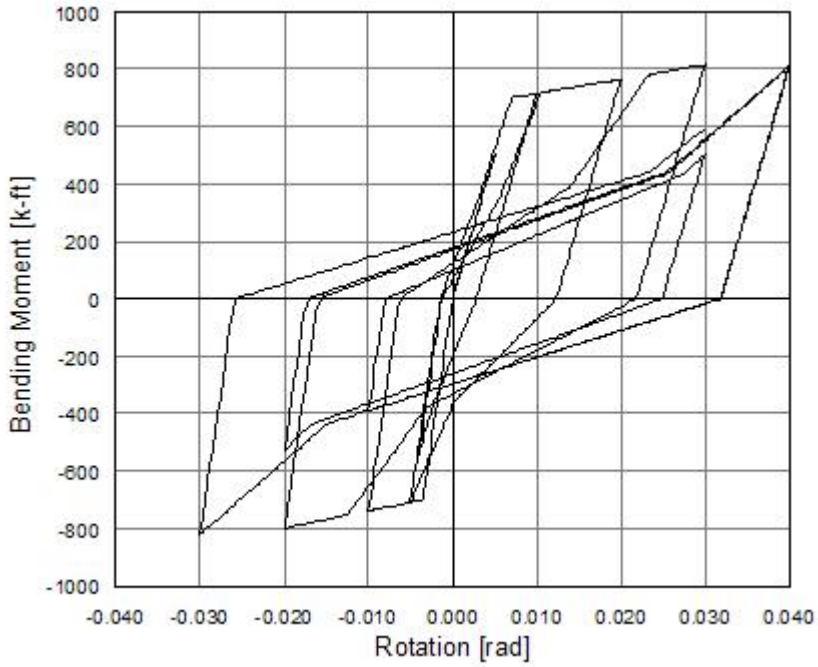

1000
800
600
400
200
0
-200
-400
-600
-800
-1000
Bending Moment [k-ft]
-0.040
-0.030
-0.020
-0.010
0.000
0.010
0.020
0.030
0.040
Rotation [rad]

Figure 39: Hyster_1 Material -- Hysteretic Behavior of Model (hardening; energy damage dE=0.05)

Hyster_1 Material -- Hysteretic Behavior of Model (hardening; energy damage dE=0.05)

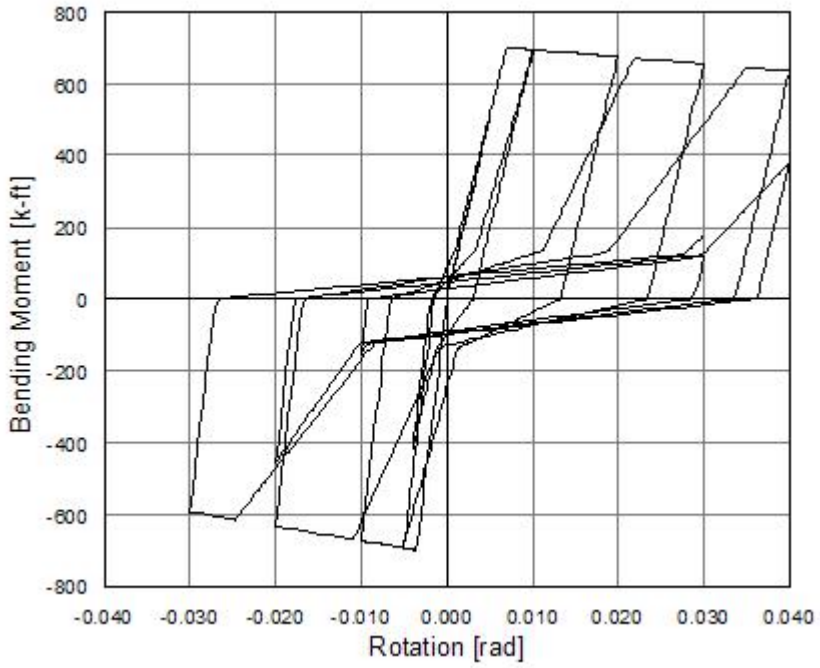


Figure 40: Hyster_1 Material -- Hysteretic Behavior of Model (softening; "pinching", displacement damage dDELTA=0.05)

Hyster_1 Material -- Hysteretic Behavior of Model (softening; "pinching", displacement damage dDELTA=0.05)

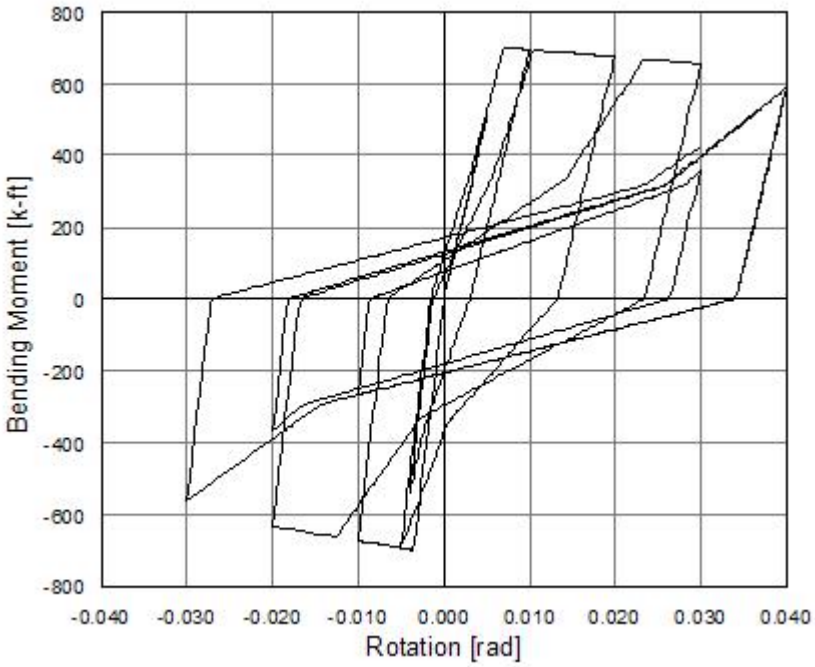

800
600
400
200
0
-200
-400
-600
-800
Bending Moment [k-ft]
-0.040
-0.030
-0.020
-0.010
0.000
0.010
0.020
0.030
0.040
Rotation [rad]

Figure 41: Hyster_1 Material -- Hysteretic Behavior of Model (softening; "pinching" energy damage dE=0.05)

Hyster_1 Material -- Hysteretic Behavior of Model (softening; "pinching" energy damage dE=0.05)

PyTzQz Uniaxial Materials

This section describes commands that are used to construct uniaxial materials for p-y, t-z and q-z elements for modeling soil-structure interaction developed at UC Davis.

There is available documentation on UCD COmpGeomech work within OpenSees framework at:

http://sokocalo.engr.ucdavis.edu/~jeremic/OpenSees/UCD_CG_OpenSees_Commands_2up.pdf
(http://sokocalo.engr.ucdavis.edu/~jeremic/OpenSees/UCD_CG_OpenSees_Commands_2up.pdf)

Also, there is a more in depth writeup (lecture notes) at:

http://sokocalo.engr.ucdavis.edu/~jeremic/CG/CompGeomechanicsLectureNotes.pdf
(http://sokocalo.engr.ucdavis.edu/~jeremic/CG/CompGeomechanicsLectureNotes.pdf)

PySimple1 Material

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

uniaxialMaterial PySimple1 $matTag $soilType $pult $Y50 $Cd <$c>.

$matTag	Unique material object integer tag.
$soilType	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	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	Displacement at which 50% of pult is mobilized in monotonic loading.
$Cd	Variable that sets the drag resistance within a fully-mobilized gap as Cd*pult.
$c	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: Full documentation of the PyLiq1 command is found in *PySimple1_documentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/PySimple1_documentation.PDF)

TzSimple1 Material

This command is used to construct a TzSimple1 uniaxial material object.

uniaxialMaterial TzSimple1 $matTag $tzType $tult $z50 <$c>.

$matTag	Unique material object integer tag.
$tzType	tzType = 1 Backbone of t-z curve approximates Reese and O’Neill (1987) relation. tzType = 2 Backbone of t-z curve approximates Mosher (1984) relation.
$tult	Ultimate capacity of the t-z material. Note that “t” or “tult” 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].

$z50	Displacement at which 50% of tult is mobilized in monotonic loading.
$c	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: Full documentation of the TzLiq1 command is included in:
TzSimple1_documentation.PDF
(http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/TzSimple1_documentation.PDF)

QzSimple1 Material

This command is used to construct a QzSimple1 uniaxial material object.

uniaxialMaterial PySimple1 $matTag $qzType $qult $Y50 <$suction $c>.

$matTag	Unique material object integer tag.
$soilType	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	Ultimate capacity of the q-z material. Note that "q" 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].
$z50	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.
$suction	Uplift resistance is equal to suction*qult. Default = 0.0. The value of suction must be 0.0 to 0.1.*
$c	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: Full documentation of the QzSimple1 command is found in: *QzSimple1_Documentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/QzSimple1_documentation.PDF)

***NOTE:** Optional variables suction and c must either both be omitted or both be included.

PyLiq1 Material

This command is used to construct a PyLiq1 uniaxial material object.

uniaxialMaterial PyLiq1 $matTag $soilType $pult $Y50 $Cd $c $pRes $solidElem1 $solidElem2.

$matTag	Unique material object integer tag.
$soilType	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	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	Displacement at which 50% of pult is mobilized in monotonic loading.
$Cd	Variable that sets the drag resistance within a fully-mobilized gap as Cd*pult.
$c	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.
$pRes	Minimum (or residual) p-y resistance that the material retains as the adjacent solid soil elements liquefy (i.e. at r_u = 1.0).
$solidElem1	Element object integer tag for a solid element from which PyLiq1 will obtain mean effective stresses and pore pressures.

$solidElem2 Element object integer tag for a solid element from which PyLiq1 will obtain mean effective stresses and pore pressures.

NOTE: Full documentation of the PyLiq1 command is found in: *PyLiq1_Documentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/PyLiq1_documentation.pdf)

NOTE: The implementation of PyLiq1 requires that the specified soil elements consist of FluidSolidPorousMaterials in FourNodeQuad elements. To model the effects of liquefaction with PyLiq1, it is necessary to use the material stage updating command:

updateMaterialStage –material $matNum –stage $sNum

where the argument matNum is the material number (for PyLiq1) and the argument sNum is the desired stage (valid values are 0 & 1). With sNum=0, the PyLiq1 behavior will be independent of any pore pressure in the specified solidElem's. When updateMaterialStage first sets sNum=1, PyLiq1 will obtain the average mean effective stress in the two solidElem's and treat it as the initial consolidation stress prior to undrained loading. Thereafter, the behavior of PyLiq1 will depend on the mean effective stresses (and hence excess pore pressures) in the solidElem's. The default value of sNum is 0 (i.e., sNum=0 if updateMaterialStage is not called). Note that the updateMaterialStage command is used with some soil material models, and that sNum=0 generally corresponds to the application of gravity loads (e.g., elastic behavior with no excess pore pressure development) and sNum=1 generally corresponds to undrained loading (e.g., plastic behavior with excess pore pressure development). The analysis for gravity loading cannot use the "algorithm Linear" command because the relevant soil materials do not currently work properly with this command. Instead, the "algorithm Newton" or some other option must be used.

TzLiq1 Material

This command is used to construct a PyLiq1 uniaxial material object.

uniaxialMaterial TzLiq1 $matTag $soilType $tult $z50 $c $solidElem1 $solidElem2.

$matTag Unique material object integer tag.

$tzType	tzType = 1 Backbone of t-z curve approximates Reese and O'Neill (1987) relation. tzType = 2 Backbone of t-z curve approximates Mosher (1984) relation.
$tult	Ultimate capacity of the t-z material. Note that "t" or "tult" 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].
$z50	Displacement at which 50% of tult is mobilized in monotonic loading.
$c	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.
$solidElem1	Element object integer tag for a solid element from which TzLiq1 will obtain mean effective stresses and pore pressures.
$solidElem2	Element object integer tag for a solid element from which TzLiq1 will obtain mean effective stresses and pore pressures.

NOTE: Full documentation of the TzLiq1 command is included in: *TzLiq1_Documentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/TzLiq1_documentation.pdf)

NOTE: The implementation of TzLiq1 requires that the specified soil elements consist of FluidSolidPorousMaterials in FourNodeQuad elements. To model the effects of liquefaction with TzLiq1, it is necessary to use the material stage updating command:

updateMaterialStage –material $matNum –stage $sNum

where the argument matNum is the material number (for TzLiq1) and the argument sNum is the desired stage (valid values are 0 & 1). With sNum=0, the TzLiq1 behavior will be independent of any pore pressure in the specified solidElem's. When updateMaterialStage first sets sNum=1, TzLiq1 will obtain the average mean effective stress in the two solidElem's and treat it as the initial consolidation stress prior to undrained loading. Thereafter, the behavior of TzLiq1 will depend on the mean effective stress (and hence excess pore pressures) in the solidElem's. The default value of sNum is 0 (i.e., sNum=0 if updateMaterialStage is not called). Note that the updateMaterialStage command is used with some soil material models, and that sNum=0 generally corresponds to the application of gravity loads (e.g., elastic behavior with no excess pore pressure development) and sNum=1 generally corresponds to undrained loading (e.g., plastic behavior with excess pore pressures development). The analysis for gravity loading cannot use the "algorithm Linear" command because the relevant soil materials do not currently work properly with this command. Instead, the "algorithm Newton" or some other option must be used.

PySimple1Gen Command

This command is used to construct output files containing material properties for PySimple1 uniaxial materials. The PySimple1Gen command constructs PySimple1 materials (Boulanger, 2003) for pre-defined zeroLength elements.

PySimple1Gen $file1 $file2 $file3 $file4 $file5 <$file6>

$file1	The name of an input file containing soil and pile properties required to define the PySimple1 materials.
$file2	The name of an input file containing information about nodes that define the mesh in the domain.

$file3	The name of an input file containing information about the zeroLength elements to be assigned PySimple1 materials (hereafter called p-y elements).
$file4	The name of an input file containing information about the beam column elements that are attached to p-y elements.
$file5	The name of the output file to which the PySimple1 materials are written.
$file6	The name of the output file to which the applied patterns are written (optional).

The command has been structured such that $File2, $File3, $File4, $File5 and $File6 can be sourced directly by OpenSees from within a master tcl file. Hence $File2, $File3 and $File4 serve two purposes:

1. They provide information to PySimple1Gen to create the PySimple1 materials.
2. They can be sourced directly in a master tcl file to define the nodes, zeroLength elements for p-y materials, and pile elements, respectively.

Furthermore, $File5 and $File 6 serve the following purpose:

1. They can be sourced by OpenSees from within a master tcl file to define the PySimple1 materials and the applied patterns, respectively.

The intended use of the files is demonstrated in an example problem in the Appendix: *PySimple1GenDocumentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/PySimple1GenDocumentation.pdf)

TzSimple1Gen Command

The TzSimple1Gen command constructs TzSimple1 materials (Boulanger, 2003) for pre-defined zeroLength elements.

TzSimple1Gen $file1 $file2 $file3 $file4 $file5 <$file6>

$file1	The name of an input file containing soil and pile properties required to define the TzSimple1 materials.
$file2	The name of an input file containing information about the nodes that define the mesh in the domain.

$file3	The name of an input file containing information about the zeroLength elements that are to be assigned TzSimple1 materials (hereafter called tz elements).
$file4	The name of an input file containing information about the beam column elements that are attached to tz elements.
$file5	The name of the output file to which the TzSimple1 materials are written.
$file6	The name of the output file to which the applied patterns are written (optional).

The command has been structured such that $File2, $File3, $File4, $File5 and $File6 can be sourced directly from within a master tcl file. Hence $File2, $File3 and $File4 serve two purposes:

1 They provide information to TzSimple1Gen to create the TzSimple1 materials.

2 They can be sourced directly in a master tcl file to define the nodes, zeroLength elements for tz materials, and pile elements, respectively.

Furthermore, $File5 and $File6 serve the following purpose:

1 They can be sourced directly in a master tcl file to define the TzSimple1 materials and the applied patterns.

The dual use of the files is demonstrated in an example problem in the Appendix: *TzSimple1GenDocumentation.pdf* (http://peer.berkeley.edu/~silvia/OpenSees/manual/documents/PyTzQzMats/TzSimple1GenDocumentation.pdf)

CHAPTER 9

nDMaterial Command

This command is used to construct an NDMaterial object which represents stress-strain relationships at the integration points of continuum and force-deformation elements.

The valid queries to any ND material when creating an *ElementRecorder* (page 224) are 'strain,' 'stress,' and 'tangent.'

In This Chapter

Elastic Isotropic Material

This command is used to construct an ElasticIsotropic material object.

nDMaterial ElasticIsotropic $matTag $E $v

$matTag	unique material object integer tag
$E	elastic Modulus
$v	Poisson's ratio

The material formulations for the ElasticIsotropic object are "ThreeDimensional," "PlaneStrain," "Plane Stress," "AxiSymmetric," and "PlateFiber." These are the valid strings that can be passed to the *continuum elements* (page 158, page 155, page 156, page 154, page 155, page 157) for the type parameter.

J2 Plasticity Material

This command is used to construct a J2Plasticity material object.

nDmaterial J2Plasticity $matTag $K $G $sig0 $sigInf $delta $H

$matTag	unique material object integer tag
$K	bulk Modulus
$G	shear Modulus
$sig0	initial yield stress
$sigInf	final saturation yield stress
$delta	exponential hardening parameter
$H	linear hardening parameter

Plane Stress Material

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.

nDMaterial PlaneStress $matTag $threeDtag

$matTag	unique material object integer tag
$threeDTag	material tag for a previously-defined three-dimensional material

Plate Fiber Material

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.

nDMaterial PlateFiber $matTag $threeDTag

$matTag	unique material object integer tag
$threeDTag	material tag for a previously-defined three-dimensional material

Template Elasto-Plastic Material

This command is used to construct the template elasto-plastic material object.

nDMaterial Template3Dep $matTag $ElmatTag -YS $ys -PS $ps -EPS $eps <-ELS1 $el> <-ELT1 $et>

$matTag	unique material object tag
$ElmatTag	previously defined elastic nDMaterial (such as *ElasticIsotropic3D* (page 108), PressureDependentElastic3D) tag
$ys	yield surface variable, previously defined in Yield Surface (page 111) object
$ps	potential surface variable, previously defined in Potential Surface (page 112) object
$eps	elasto-plastic state variable, previously defined in EPState (page 114) object
$el	scalar (isotropic) evolution law variable, previously defined in Evolution Law (page 113) object
$et	tensorial (kinematic) evolution law variable, previously defined in Evolution Law (page 113) object

Yield Surface

This command sets the yield surface variable ys to be the specified type. Currently these include: Drucker-Prager yield surface, Rounded Mohr-Coulomb (Willam-Warnke) yield surface, von Mises yield surface, Cam-Clay yield surface and Leon yield surface.

set ys "-YieldSurfaceType <parameter list>"

Valid strings for YieldSurfaceType are DP, VM and CC.

- ***For Drucker-Prager yield surface***

set ys "-DP"

- ***For von Mises yield surface***

set ys "-VM"

- ***For rounded Mohr-Coulomb (Willam-Warnke) yield surface***

set ys "-RMC01"

- ***For Cam-Clay yield surface***

set ys "-CC $M"

$M	Slope of the critical state line in p-q space

- ***For Leon yield surface***

set ys "-Leon $fc $ft $e $c"

$fc	compressive strength
$ft	tensile strength
$e	excentricity of yield surface (usually 0.6-0.7)
$c	cohesion

Potential Surface

This command is used to set the potential surface variable $ps to the specific surface (or directly to the flow directions). Currently included are: Drucker-Prager potential surface, Rounded Mohr-Coulomb (Willam-Warnke) potential surface, von Mises potential surface, Cam-Clay potential surface and Leon potential surface.

```
set ps "-PotentialSurfaceType <parameter list>"
```

Valid strings for PotentialSurfaceType are DP, VM and CC.

- ***For the Drucker-Prager potential surface***

```
set ps "-DP"
```

- ***For the von Mises potential surface***

```
set ps "-VM"
```

- ***For rounded Mohr-Coulomb (Willam-Warnke) potential surface***

```
set ps "-RMC01"
```

- ***For the Cam-Clay potential surface***

```
set ps "-CC $M"
```

$M Slope of the critical state line in p-q space

- ***For Leon potential surface***

```
set  ps "-Leon $fc $ft $e $c"
```

$fc compressive strength

$ft	tensile strength
$e	excentricity of yield surface (usually 0.6-0.7)
$c	cohesion

Evolution Law

This command is used to set the evolution law variable el to the specified type. There are two types of evolutions laws implemented: scalar (isotropic) evolution and tensorial (kinemartic) evolution. For scalar evolution law, there are linear scalar evolution law and nonlinear scalar evolution law. For tensorial evolution law, there are linear tensorial evolution law and nonlinear tensorial evolution law.

```
set el "-EvolutionLawType <parameter list>"
```

Valid strings for EvolutionLawType are Leq, NLp, LEij, and NLEij

- ***For linear scalar evolution law***

```
set el "-Leq $a"
```

- ***For Cam-Clay type nonlinear scalar evolution law***

```
set el "-NLp $e_o $lambda $k
```

- ***For linear tensorial evolution law***

```
 set et "-LEij $a1"
```

- ***For Armstrong-Frederick type nonlinear tensorial evolution law***

```
set et "-NLEij $h_a $C_r"
```

$a	linear hardening coefficient
$e_o	initial void ratio
$lambda	nonlinear evolution law constant (Cam-Clay type)

$k	nonlinear evolution law constant (Cam-Clay type)
$a1	linear tensorial evolution law constant
$h_a	nonlinear tensorial evolution law constant (Armstrong-Frederick type)
$C_r	nonlinear tensorial evolution law constant (Armstrong-Frederick type)

EPState

This command is used to set the Elasto-Plastic State, which includes two states.

- ***To set the initial stress tensor to variable sts, if –stressp $sts is used in eps:***

```
set sts "$sigma_xx $sigma_xy $sigma_xz $sigma_yx $sigma_yy $sigma_yz
         $sigma_zx $sigma_zy $sigma_zz"
```

- ***To assign to the Elasto-Plastic state variable eps the specified state parameters***

```
set eps "<-NOD $nt> -NOS $ns $scalar1 $scalar2 ... <-stressp $sts>"
```

$sigma_xx $sigma_xy $sigma_xz $sigma_yx $sigma_yy $sigma_yz $sigma_zx $sigma_zy $sigma_zz	Initial stress tensor components (Default = 0.0),
$nt	number of tensorial internal variables
$ns	number of scalar internal variables
$scalar1 $scalar2 ...	corresponding initial values of scalar internal variables
$sts	initial stresses

FluidSolidPorousMaterial Material

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* (page 108). This material is developed to simulate the response of saturated porous media under fully undrained condition.

nDMaterial FluidSolidPorousMaterial $tag $nd $soilMatTag $combinedBulkModul

$tag	unique material object integer tag
$nd	Number of dimensions, 2 for plane-strain, and 3 for general 3D analysis.
$soilMatTag	The material number for the solid phase material (previously defined).
$combinedBulkModul	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 (2.2×10^6 kPa for water typically), and n the initial porosity.

NOTE:

1. Buoyant unit weight (total unit weight - fluid unit weight) should be used in definition of the finite elements composed of a FluidSolidPorousMaterial.

2. During the application of gravity (elastic) load, the fluid phase does not contribute to the material response.

OUTPUT INTERFACE:

The following information may be extracted for this material at given integration point, using the OpenSees Element Recorder facility (McKenna and Fenves 2001): "**stress**", "**strain**", "**tangent**", or "**pressure**". The "**pressure**" option records excess pore pressure and excess pore pressure ratio at a given material integration point.

updateMaterialStage

This command is used to update a *PressureDependMultiYield* (page 122), a *PressureIndependMultiYield* (page 117), or a *FluidSolidPorous* (page 115) material. To conduct a seismic analysis, two stages should be followed. First, during the application of gravity load (and static loads if any), set material stage to 0, and material behavior is linear elastic (with G_r and B_r as elastic moduli). A *FluidSolidPorous* (page 115) material does not contribute to the material response if its stage is set to 0. After the application of gravity load, set material stage to 1 or 2. In case of stage 2, all the elastic material properties are then internally determined at the current effective confinement, and remain constant thereafter. In the subsequent dynamic (fast) loading phase(s), the deviatoric stress-strain response is elastic-plastic (stage 1) or linear-elastic (stage 2), and the volumetric response remains linear-elastic.

updateMaterialStage -material $tag -stage $sNum

$tag	previously-defined material object integer tag
$sNum	desired stage: 0 – linear elastic, 1 – plastic, 2 – Linear elastic, with elasticity constants (shear modulus and bulk modulus) as a function of initial effective confinement.

PressureIndependMultiYield Material

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.

During the application of gravity load (and static loads if any), material behavior is linear elastic. In the subsequent dynamic (fast) loading phase(s), the stress-strain response is elastic-plastic (see *updateMaterialStage* (page 116) command). Plasticity is formulated based on the multi-surface (nested surfaces) concept, with an associative flow rule. The yield surfaces are of the Von Mises type.

nDmaterial PressureIndependMultiYield $tag $nd $rho $refShearModul $refBulkModul $cohesi $peakShearStra <$frictionAng $refPress $pressDependCoe $noYieldSurf <$gamma1 $Gs1 ...> >

$tag	unique material object integer tag
$nd	Number of dimensions, 2 for plane-strain, and 3 for 3D analysis
$rho	Saturated soil mass density.
$refShearModul ($G_r$)	Reference low-strain shear modulus, specified at a reference mean effective confining pressure refPress of p'_r (see below).
$refBulkModul ($B_r$)	Reference bulk modulus, specified at a reference mean effective confining pressure refPress of p'_r (see below).
$cohesi (c)	Apparent cohesion at zero effective confinement.
$peakShearStra ($\gamma_{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).
$frictionAng ($\phi$)	Friction angle at peak shear strength in degrees, optional (default is 0.0).
$refPress ($p'_r$)	Reference mean effective confining pressure at which G_r, B_r, and γ_{max} are defined, optional (default is 100.).
$pressDependCoe (d)	An optional non-negative constant defining variations of G and B as a function of initial effective confinement p'_i (default is 0.0):

$$G = G_r\left(\frac{p'_i}{p'_r}\right)^d \qquad B = B_r\left(\frac{p'_i}{p'_r}\right)^d$$

$noYieldSurf 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.

$gamma1 $Gs1 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 (gamma) and modulus ratio (Gs) values. For example, to define 10 surfaces:

… -10 gamma1 Gs1 … gamma10 Gs10 …

See Note 3 below for some important notes.

NOTE:

1. The friction angle ϕ and cohesion c define the variation of peak (octahedral) shear strength τ_f as a function of initial effective confinement p'_i:

$$\tau_f = \frac{2\sqrt{2}\sin\phi}{3-\sin\phi} p'_i + \frac{2\sqrt{2}}{3} c$$

2. Automatic surface generation: at a constant confinement p', the shear stress τ (octahedral) - shear strain γ (octahedral) nonlinearity is defined by a hyperbolic curve (backbone curve):

$$\tau = \frac{G\gamma}{1+\gamma/\gamma_r}$$

where γ_r satisfies the following equation at p'_r:

$$\tau_f = \frac{2\sqrt{2}\sin\phi}{3-\sin\phi} p'_r + \frac{2\sqrt{2}}{3} c = \frac{G_r \gamma_{max}}{1+\gamma_{max}/\gamma_r}$$

3. (User defined surfaces) If the user specifies ϕ=0, cohesion c will be ignored. Instead, c is defined by c=sqrt(3)*σ_m/2, where σ_m is the product of the last modulus and strain pair in the modulus reduction curve. Therefore, it is important to adjust the backbone curve so as to render an appropriate c.

If the user specifies ϕ>0, this ϕ will be ignored. Instead, ϕ is defined as follows:

$$\sin\phi = \frac{3(\sqrt{3}\,\sigma_m - 2c)/p'_r}{6+(\sqrt{3}\,\sigma_m - 2c)/p'_r}$$

If the resulting ϕ<0, we set ϕ=0 and c=sqrt(3)*σ_m/2.

Also remember that improper modulus reduction curves can result in strain softening response (negative tangent shear modulus), which is not allowed in the current model formulation. Finally, note that the backbone curve varies with confinement, although the variation is small within commonly interested confinement ranges. Backbone curves at different confinements can be obtained using the OpenSees element recorder facility (see OUTPUT INTERFACE below).

SUGGESTED PARAMETER VALUES

For user convenience, a table is provided below as a quick reference for selecting parameter values. However, use of this table should be of great caution, and other information should be incorporated wherever possible.

	Soft Clay	Medium Clay	Stiff Clay
rho (ton/m^3)	1.3	1.5	1.8
refShearModul (kPa)	$1.3x10^4$	$6.0x10^4$	$1.5x10^5$
refBulkModu (kPa)	$6.5x10^4$	$3.0x10^5$	$7.5x10^5$
cohesi (kPa)	18	37	75
peakShearStra	0.1	0.1	0.1
frictionAng	0	0	0
pressDependCoe	0	0	0

OUTPUT INTERFACE:

The following information may be extracted for this material at a given integration point, using the OpenSees Element Recorder facility (McKenna and Fenves 2001): "**stress**", "**strain**", "**backbone**", or "**tangent**".

For 2D problems, the stress output follows this order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, η_r, where η_r is the ratio between the shear (deviatoric) stress and peak shear strength at the current confinement ($0<= \eta_r <=1.0$). The strain output follows this order: ε_{xx}, ε_{yy}, γ_{xy}.

For 3D problems, the stress output follows this order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{zx}, η_r, and the strain output follows this order: ε_{xx}, ε_{yy}, ε_{zz}, γ_{xy}, γ_{yz}, γ_{zx}.

The **"backbone"** option records (secant) shear modulus reduction curves at one or more given confinements. The specific recorder command is as follows:

recorder Element $eleNum -file $fName -dT $deltaT material $GaussNum backbone $p1 <$p2 …>

where p1, p2, … are the confinements at which modulus reduction curves are recorded. In the output file, corresponding to each given confinement there are two columns: shear strain γ and secant modulus G_s. The number of rows equals the number of yield surfaces.

updateMaterialStage

This command is used to update a *PressureDependMultiYield* (page 122), a *PressureIndependMultiYield* (page 117), or a *FluidSolidPorous* (page 115) material. To conduct a seismic analysis, two stages should be followed. First, during the application of gravity load (and static loads if any), set material stage to 0, and material behavior is linear elastic (with G_r and B_r as elastic moduli). A *FluidSolidPorous* (page 115) material does not contribute to the material response if its stage is set to 0. After the application of gravity load, set material stage to 1 or 2. In case of stage 2, all the elastic material properties are then internally determined at the current effective confinement, and remain constant thereafter. In the subsequent dynamic (fast) loading phase(s), the deviatoric stress-strain response is elastic-plastic (stage 1) or linear-elastic (stage 2), and the volumetric response remains linear-elastic.

updateMaterialStage -material $tag -stage $sNum

$tag previously-defined material object integer tag

$sNum desired stage:

0 – linear elastic,

1 – plastic,

2 – Linear elastic, with elasticity constants (shear modulus and bulk modulus) as a function of initial effective confinement.

updateParameter

This command is used to update material parameters of *PressureDependMultiYield* (page 122) or *PressureIndependMultiYield* (page 117) material. Currently, two material parameters, reference low-strain shear modulus G_r and reference bulk modulus B_r, can be modified during an analysis.

To update G_r:

```
updateParameter -material $tag -refG $newVal
```

To update B_r:

```
updateParameter -material $tag -refB $newVal
```

$tag previously-defined material object integer tag

$newVal New parameter value

PressureDependMultiYield Material

PressureDependMultiYield material is an elastic-plastic material used 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.

When this material is employed in regular solid elements (e.g., *FourNodeQuad* (page 155, page 156, page 154, page 155), *Brick* (page 158, page 157)), it simulates drained soil response. To simulate soil response under fully undrained condition, this material may be either embedded in a *FluidSolidPorousMaterial* (page 115), or used with the *FourNodeQuadUP* (page 165) element with very low permeability. To simulate partially drained soil response, this material should be used with the *FourNodeQuadUP* (page 165) element with proper permeability values.

During the application of gravity load (and static loads if any), material behavior is linear elastic. In the subsequent dynamic (fast) loading phase(s), the stress-strain response is elastic-plastic (see *updateMaterialStage* (page 116) command). Plasticity is formulated based on the multi-surface (nested surfaces) concept, with a non-associative flow rule to reproduce dilatancy effect. The yield surfaces are of the Drucker-Prager type.

```
nDMaterial PressureDependMultiYield $tag $nd $rho $refShearModul
        $refBulkModul $frictionAng $peakShearStra $refPress
        $pressDependCoe $PTAng $contrac $dilat1 $dilat2 $liquefac1
        $liquefac2 $liquefac3 <$noYieldSurf <$gamma1 $Gs1 ...> $e $cs1
        $cs2 $cs3 $pa>
```

$tag	unique material object integer tag
$nd	Number of dimensions, 2 for plane-strain, and 3 for 3D analysis.
$rho	Saturated soil mass density.
$refShearModul ($G_r$)	Reference low-strain shear modulus, specified at a reference mean effective confining pressure refPress of p'_r (see below).
$refBulkModul ($B_r$)	Reference bulk modulus, specified at a reference mean effective confining pressure refPress of p'_r (see below).
$frictionAng ($\phi$)	Friction angle at peak shear strength, in degrees.

$peakShearStra ($\gamma_{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).

Octahedral shear strain is defined as:

$$\gamma = \frac{2}{3}\left[(\varepsilon_{xx}-\varepsilon_{yy})^2+(\varepsilon_{yy}-\varepsilon_{zz})^2+(\varepsilon_{xx}-\varepsilon_{zz})^2+6\varepsilon_{xy}^2+6\varepsilon_{yz}^2+6\varepsilon_{xz}^2\right]^{1/2}$$

$refPress ($p'_r$)

Reference mean effective confining pressure at which G_r, B_r, and γ_{max} are defined.

$pressDependCoe (d)

A positive constant defining variations of G and B as a function of instantaneous effective confinement p':

$$G = G_r\left(\frac{p'}{p'_r}\right)^d \qquad B = B_r\left(\frac{p'}{p'_r}\right)^d$$

$PTAng ($\phi_{PT}$)

Phase transformation angle, in degrees.

$contrac

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.

$dilat1, $dilat2

Non-negative constants defining the rate of shear-induced volume increase (dilation). Larger values correspond to stronger dilation rate.

$liquefac1, $liquefac2, $liquefac3

Parameters controlling the mechanism of liquefaction-induced perfectly plastic shear strain accumulation, i.e., cyclic mobility. **Set liquefac1 = 0 to deactivate this mechanism altogether.**

liquefac1 defines the effective confining pressure (e.g., 10 kPa) below which the mechanism is in effect. Smaller values should be assigned to denser sands.

liquefac2 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.

liquefac3 defines the maximum amount of biased perfectly plastic shear strain γ_b accumulated at each loading phase under biased shear loading conditions, as γ_b =liquefac2 x liquefac3.

Typically, **liquefac3** 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

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.

$gamma1 $Gs1 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 (gamma) and modulus ratio (Gs) values. For example, to define 10 surfaces:

… -10 gamma1 Gs1 … gamma10 Gs10 …

See Note 3 below for some important notes.

$e Initial void ratio, optional (default is 0.6).

$cs1, $cs2, $cs3, $pa Parameters defining a straight critical-state line e_c in e-p' space.

If cs3=0,

$$e_c = cs1 - cs2 \log(p'/p_a)$$

else (Li and Wang, JGGE, 124(12)),

$$e_c = cs1 - cs2(p'/p_a)^{cs3}$$

where p_a is atmospheric pressure for normalization (typically 101 kPa in SI units). All four constants are optional (default values: cs1=0.9, cs2=0.02, cs3=0.7, pa=101).

NOTE:

1. The friction angle ϕ defines the variation of peak (octahedral) shear strength τ_f as a function of current effective confinement p':

$$\tau_f = \frac{2\sqrt{2}\sin\phi}{3-\sin\phi}p'$$

Octahedral shear stress is defined as:

$$\tau = \frac{1}{3}\left[(\sigma_{xx}-\sigma_{yy})^2 + (\sigma_{yy}-\sigma_{zz})^2 + (\sigma_{xx}-\sigma_{zz})^2 + 6\sigma_{xy}^2 + 6\sigma_{yz}^2 + 6\sigma_{xz}^2\right]^{1/2}$$

2. (Automatic surface generation) At a constant confinement p', the shear stress τ (octahedral) - shear strain γ (octahedral) nonlinearity is defined by a hyperbolic curve (backbone curve):

$$\tau = \frac{G\gamma}{1+\gamma/\gamma_r}$$

where γ_r satisfies the following equation at p'_r:

$$\tau_f = \frac{2\sqrt{2}\sin\phi}{3-\sin\phi} p'_r = \frac{G_r \gamma_{max}}{1+\gamma_{max}/\gamma_r}$$

3. (User defined surfaces) The user specified friction angle ϕ is ignored. Instead, ϕ is defined as follows:

$$\sin\phi = \frac{3\sqrt{3}\,\sigma_m / p'_r}{6+\sqrt{3}\,\sigma_m / p'_r}$$

where σ_m is the product of the last modulus and strain pair in the modulus reduction curve. Therefore, it is important to adjust the backbone curve so as to render an appropriate ϕ. If the resulting ϕ is smaller than the phase transformation angle ϕ_{PT}, ϕ_{PT} is set equal to ϕ.

Also remember that improper modulus reduction curves can result in strain softening response (negative tangent shear modulus), which is not allowed in the current model formulation. Finally, note that the backbone curve varies with confinement, although the variations are small within commonly interested confinement ranges. Backbone curves at different confinements can be obtained using the OpenSees element recorder facility (see OUTPUT INTERFACE below).

The last five optional parameters are needed when critical-state response (flow liquefaction) is anticipated. Upon reaching the critical-state line, material dilatancy is set to zero.

SUGGESTED PARAMETER VALUES

For user convenience, a table is provided below as a quick reference for selecting parameter values. However, use of this table should be of great caution, and other information should be incorporated wherever possible.

	Loose Sand (15%-35%)	Medium Sand (35%-65%)	Medium-dense Sand (65%-85%)	Dense Sand (85%-100%)
rho (ton/m3)	1.7	1.9	2.0	2.1
refShearModul (kPa, at p'_r=80 kPa)	5.5×10^4	7.5×10^4	1.0×10^5	1.3×10^5
refBulkModu (kPa, at p'_r=80 kPa)	1.5×10^5	2.0×10^5	3.0×10^5	3.9×10^5
frictionAng	29	33	37	40

peakShearStra (at p'_r=80 kPa)	0.1	0.1	0.1	0.1
pressDependCoe	0.5	0.5	0.5	0.5
PTAng	27	23	20	16
contrac	0.21	0.07	0.05	0.03
dilat1	0.	0.4	0.6	0.8
dilat2	0	2	3	5
liquefac1 (kPa)	10	10	5	0
liquefac2	0.02	0.01	0.003	0
liquefac3	1	1	1	0
e	0.85	0.7	0.55	0.45

OUTPUT INTERFACE:

The following information may be extracted for this material at a given integration point, using the OpenSees *Element Recorder* (page 221) facility (McKenna and Fenves 2001): "**stress**", "**strain**", "**backbone**", or "**tangent**".

For 2D problems, the stress output follows this order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, η_r, where η_r is the ratio between the shear (deviatoric) stress and peak shear strength at the current confinement ($0 <= \eta_r <= 1.0$). The strain output follows this order: ε_{xx}, ε_{yy}, γ_{xy}.

For 3D problems, the stress output follows this order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{zx}, η_r, and the strain output follows this order: ε_{xx}, ε_{yy}, ε_{zz}, γ_{xy}, γ_{yz}, γ_{zx}.

The **"backbone"** option records (secant) shear modulus reduction curves at one or more given confinements. The specific recorder command is as follows:

recorder Element $eleNum -file $fName -dT $deltaT material $GaussNum backbone $p1 <$p2 …>

where p1, p2, … are the confinements at which modulus reduction curves are recorded. In the output file, corresponding to each given confinement there are two columns: shear strain γ and secant modulus G_s. The number of rows equals the number of yield surfaces.

updateMaterialStage

This command is used to update a *PressureDependMultiYield* (page 122), a *PressureIndependMultiYield* (page 117), or a *FluidSolidPorous* (page 115) material. To conduct a seismic analysis, two stages should be followed. First, during the application of gravity load (and static loads if any), set material stage to 0, and material behavior is linear elastic (with G_r and B_r as elastic moduli). A *FluidSolidPorous* (page 115) material does not contribute to the material response if its stage is set to 0. After the application of gravity load, set material stage to 1 or 2. In case of stage 2, all the elastic material properties are then internally determined at the current effective confinement, and remain constant thereafter. In the subsequent dynamic (fast) loading phase(s), the deviatoric stress-strain response is elastic-plastic (stage 1) or linear-elastic (stage 2), and the volumetric response remains linear-elastic.

updateMaterialStage -material $tag -stage $sNum

$tag	previously-defined material object integer tag
$sNum	desired stage: 0 – linear elastic, 1 – plastic, 2 – Linear elastic, with elasticity constants (shear modulus and bulk modulus) as a function of initial effective confinement.

updateParameter

This command is used to update material parameters of *PressureDependMultiYield* (page 122) or *PressureIndependMultiYield* (page 117) material. Currently, two material parameters, reference low-strain shear modulus G_r and reference bulk modulus B_r, can be modified during an analysis.

To update G_r:

```
updateParameter -material $tag -refG $newVal
```

To update B_r:

```
updateParameter -material $tag -refB $newVal
```

$tag	previously-defined material object integer tag
$newVal	New parameter value

CHAPTER 10

section Command

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.

➢ *What is a section?*

- A section defines the stress resultant force-deformation response at a cross section of a beam-column or plate element
- Types of sections:

• **Elastic** – defined by material and geometric constants

• **Resultant** – general nonlinear description of force-deformation response, e.g. moment-curvature

• **Fiber** – section is discretized into smaller regions for which the material stress-strain response is integrated to give resultant behavior, e.g. reinforced concrete

The valid queries to any section when creating an *ElementRecorder* (page 224) are 'force' and 'deformation.'

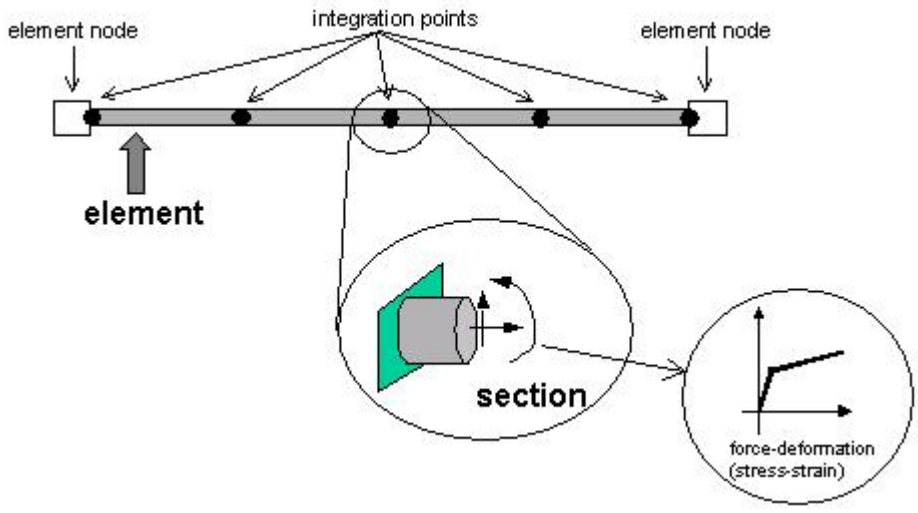


Figure 42: Section Representation

In This Chapter

Elastic Section

This command is used to construct an ElasticSection object.

section Elastic $secTag $E $A $Iz <$Iy $G $J>

$secTag	unique section object tag
$E	Young's Modulus
$A	cross-sectional area of section
$Iz	second moment of area about the local z-axis
$Iy	second moment of area about the local y-axis (optional, used for 3D analysis)
$G	Shear Modulus (optional, used for 3D analysis)
$J	torsional moment of inertia of section (optional, used for 3D analysis)

This command is useful for patch tests of the *nonlinear beam-column elements* (page 149). It also allows nonlinear beam-column elements to be used for elastic analysis.

EXAMPLE:

section Elastic 1 29000 100 100000 80000 20000 100000; # create elastic section with IDtag 1

Uniaxial Section

This command is used to construct a UniaxialSection object which uses a previously-defined *UniaxialMaterial* (page 35) object to represent a single section force-deformation response quantity. (Formerly known as Generic1d section, which is still accepted by OpenSees)

section Uniaxial $secTag $matTag $string

$secTag		unique section object tag
$matTag		previously-defined *UniaxialMaterial* (page 35) object
$string		the force-deformation quantity to be modeled by this section object. One of the following strings is 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

EXAMPLE:

section Uniaxial 1 1 Mz; # create sectionID-tag 1 from UniaxialMaterialID-tag 1 for the moment-curvature about section local z-axis.

Fiber Section

The FiberSection object is composed of Fiber objects.

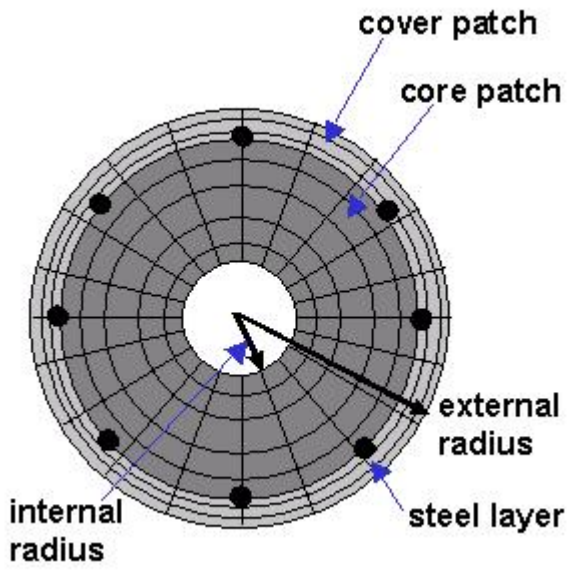
A fiber section has a general geometric configuration formed by subregions of simpler, regular shapes (e.g. quadrilateral, circular and triangular regions) called patches. In addition, layers of reinforcement bars can be specified. The subcommands *patch* (page 134) and *layer* (page 139, page 138) are used to define the discretization of the section into fibers. Individual fibers, however, can also be defined using the *fiber* (page 133) command (During generation, the Fiber objects are associated with *uniaxialMaterial* (page 35) objects, which enforce Bernoulli beam assumptions.

The geometric parameters are defined with respect to a planar local coordinate system (y,z). See figures.

```
section Fiber $secTag {

        fiber <fiber arguments>

        patch <patch arguments>

        layer <layer arguments>

}
```

An example fiber section is shown in the Figure.

Figure 43: Fiber Section



Fiber Command

This command is used to construct a UniaxialFiber object and add it to the section.

fiber $yLoc $zLoc $A $matTag

$yLoc y coordinate of the fiber in the section (local coordinate system)

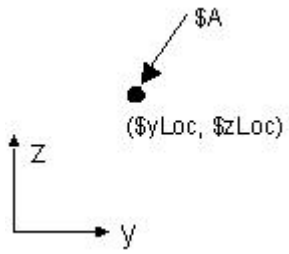
$zLoc	z coordinate of the fiber in the section (local coordinate system)
$A	area of fiber
$matTag	material tag of the pre-defined *UniaxialMaterial* (page 35) object used to represent the stress-strain for the area of the fiber

NOTE: in *2D* (page 26) bending is about the local z-axis

EXAMPLE:

fiber 0.0 0.0 1.0 1; # create a single fiber of area 1.0 at the origin (0,0) of the section, using materialIDtag 1

Figure 44: Fiber Command



Quadrilateral Patch Command

This command is used to construct a Patch object with a quadrilateral shape. The geometry of the patch is defined by four vertices: I J K L, as illustrated in the Figure. The coordinates of each of the four vertices is specified in sequence -- counter-clockwise.

patch quad $matTag $numSubdivIJ $numSubdivJK $yI $zI $yJ $zJ $yK $zK $yL $zL

$matTag	material integer tag of the previously-defined *UniaxialMaterial* (page 35) object used to represent the stress-strain for the area of the fiber

$numSubdivIJ		number of subdivisions (fibers) in the IJ direction.
$numSubdivJK		number of subdivisions (fibers) in the JK direction.
$yI	**$zI**	y & z-coordinates of vertex I (local coordinate system)
$yJ	**$zJ**	y & z-coordinates of vertex J (local coordinate system)
$yK	**$zK**	y & z-coordinates of vertex K (local coordinate system)
$yL	**$zL**	y & z-coordinates of vertex L (local coordinate system)

NOTE: in *2D* (page 26) bending is about the local z-axis

EXAMPLE:

patch quad $coreMatTag 8 8 -$b -$h $b -$h $b $h -$b $h; # define core patch with 8 subdivisions within a rectange of width 2b and depth 2h

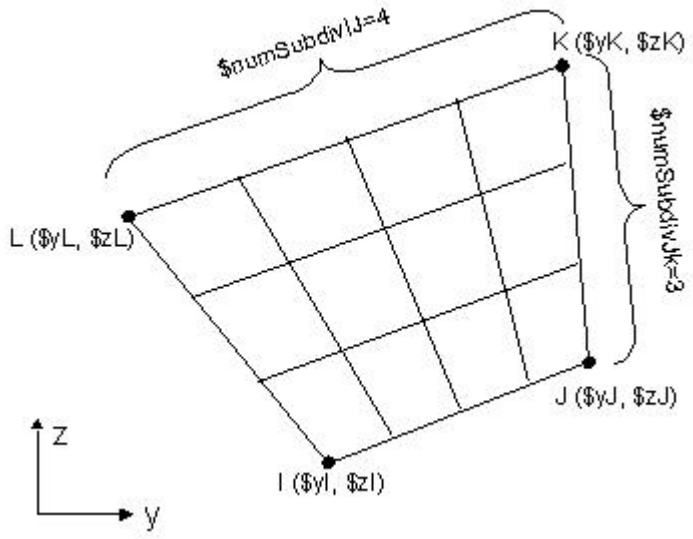


Figure 45:
Quadrilateral Patch

Circular Patch Command

This command is used to construct a Patch object with a circular shape.

patch circ $matTag $numSubdivCirc $numSubdivRad $yCenter $zCenter $intRad $extRad <$startAng $endAng>

$matTag	material integer tag of the previously-defined *UniaxialMaterial* (page 35) object used to represent the stress-strain for the area of the fiber
$numSubdivCirc	number of subdivisions (fibers) in the circumferential direction.
$numSubdivRad	number of subdivisions (fibers) in the radial direction.
$yCenter $zCenter	y & z-coordinates of the center of the circle
$intRad	internal radius
$extRad	external radius
$startAng	starting angle (optional. default=0.0)
$endAng	ending angle (optional. default=360.0)

NOTE: in *2D* (page 26) bending is about the local z-axis

EXAMPLE:

patch circ $coreMatTag 8 8 0.0 0.0 0.0 $h; # define core patch with 8 subdivisions within a whole circle of diameter 2h

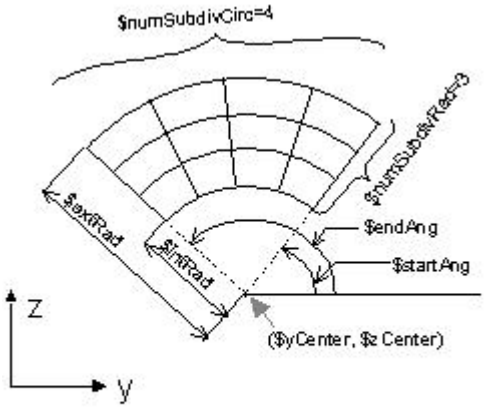


Figure 46: Circular Patch

Straight Layer Command

This command is used to construct a straight layer of reinforcing bars.

layer straight $matTag $numBars $areaBar $yStart $zStart $yEnd $zEnd

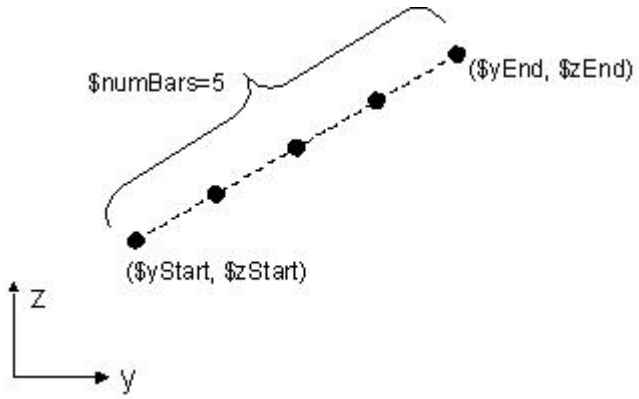
$matTag		material integer tag of the previously-defined *UniaxialMaterial* (page 35) object used to represent the stress-strain for the area of the fiber
$numBars		number of reinforcing bars along layer
$areaBar		area of individual reinforcing bar
$yStart	**$zStart**	y and z-coordinates of starting point of reinforcing layer (local coordinate system)
$yEnd	**$zEnd**	y and z-coordinates of ending point of reinforcing layer (local coordinate system)

NOTE: in *2D* (page 26) bending is about the local z-axis

EXAMPLE:

layer straight $steelMatTag 10 0.11 -b -h b -h; # define steel layer of 10 bars with area 0.11 at bottom of section of width 2b by 2h

Figure 47: Straight Layer



Circular Layer Command

This command is used to construct a circular layer of reinforcing bars.

layer circ $matTag $numBar $areaBar $yCenter $zCenter $radius <$startAng $endAng>

$matTag	material integer tag of the previously-defined *UniaxialMaterial* (page 35) object used to represent the stress-strain for the area of the fiber
$numBar	number of reinforcing bars along layer
$areaBar	area of individual reinforcing bar

$yCenter	**$zCenter**	y and z-coordinates of center of reinforcing layer (local coordinate system)
$radius		radius of reinforcing layer
$startAng	**$endAng**	starting and ending angle of reinforcing layer, respectively. (Optional, Default: a full circle is assumed 0-360)

NOTE: in *2D* (page 26) bending is about the local z-axis

EXAMPLE:

layer circ $steelMatTag 10 0.11 0.0 0.0 $h 0 360; # define circular steel layer of 10 bars with area 0.11 uniformly distributed along circumference of circle of diameter 2h

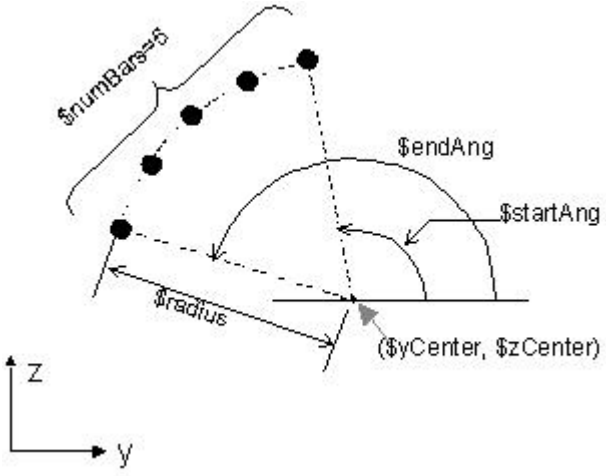


Figure 48: Circular Reinforcing Layer

Section Aggregator

This command is used to construct a SectionAggregator object which groups previously-defined *UniaxialMaterial* (page 35) objects into a single section force-deformation model.

section Aggregator $secTag $matTag1 $string1 $matTag2 $string2 ..... <-section $sectionTag>

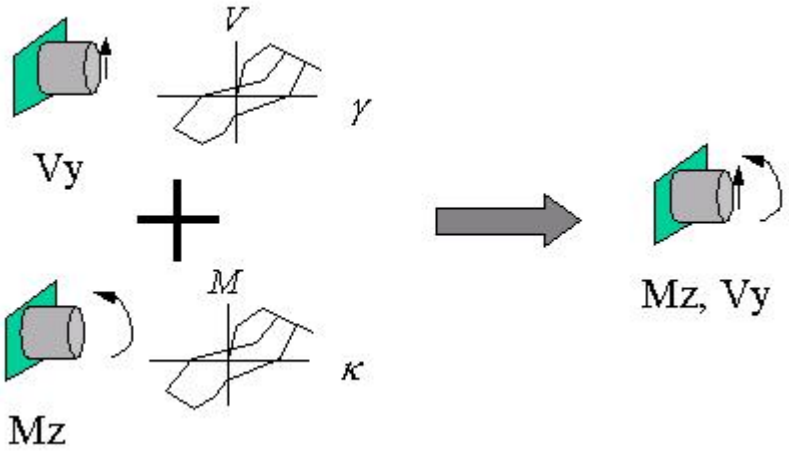
$secTag		unique section object integer tag
$matTag1, $matTag2 ...		previously-defined *UniaxialMaterial* (page 35) objects
$string1, $string2		the force-deformation quantities corresponding to each section object. One of the following strings is used:
	P	Axial force-deformation
	Mz	Moment-curvature about 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
<-section $sectionTag>		specifies a previously-defined *Section* (page 129) object (identified by the argument $sectionTag) to which these *UniaxialMaterial* (page 35) objects may be added to recursively define a new *Section* (page 129) object

NOTE: The *UniaxialMaterial* (page 35) objects aggregated in this *Section* (page 129) object are uncoupled from each other as well as from the *Section* (page 129) object represented by **$sectionTag**, if present.

There are two main tasks that can be performed using the Section Aggregator:

- ***1. Group previously defined uniaxial materials to describe stress resultant section behavior***

Figure 49: Section Aggregator 1

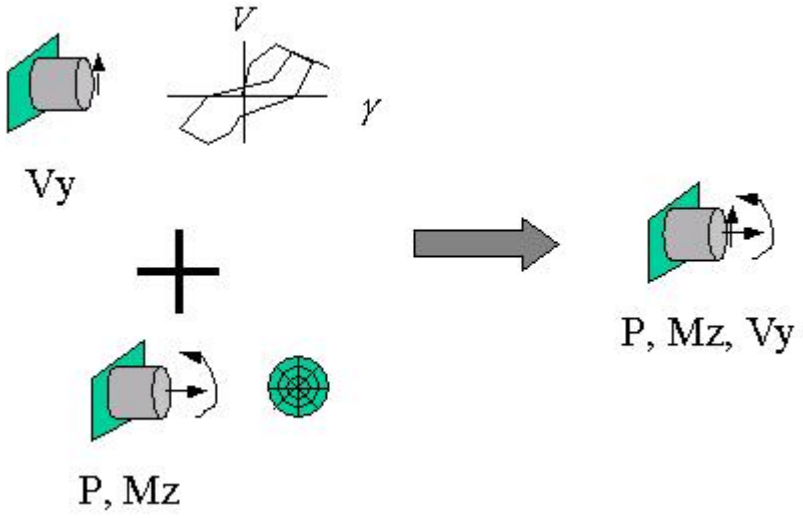


EXAMPLE:

section Aggregator 1 2 Vy 5 Mz; #create new section with IDtag 1, taking the existing material tag 2 to represent the shear and the existing material tag 5 to represent the moment.

- ***2. Add to an existing section***

Figure 50: Section Aggregator 2



EXAMPLE:

section Aggregator 2 2 Vy -section 4; # create new section with IDtag 2, taking the existing material tag 2 to represent the shear and adding it to the existing section tag 4, which may be a fiber section where the interaction betweeen axial force and flexure is already considered.

Elastic Membrane Plate Section

This command is used to construct an ElasticMembranePlateSection object, which is an isotropic section appropriate for plate and shell analysis.

section ElasticMembranePlateSection $secTag $E $nu $h $rho

$secTag	unique section object tag
$E	Elastic Modulus
$nu	Poisson's Ratio
$h	thickness of the plate section
$rho	mass density of the material (per unit volume)

Plate Fiber Section

The plate fiber section takes any *plate fiber material* (page 110) and, by thickness integration, creates a plate section appropriate for shell analysis.

section PlateFiber $secTag $fiberTag $h

$secTag	unique section object tag for section being constructed
$fiberTag	material tag for a previously-defined *plate fiber material* (page 110)
$h	thickness of the plate section

Bidirectional Section

This command is used to construct a Bidirectional section object which is the two-dimensional generalization of a one-dimensional elasto-plastic model with linear hardening.

section Bidirectional $matTag $E $sigY $H_iso $H_kin

$matTag	unique section object integer tag
$E	Elastic Modulus
$sigY	yield stress
$H_iso	isotropic hardening Modulus
$H_kin	kinematic hardening Modulus

then why do we have kinematic and isotripic hardening parameters??????**********

CHAPTER 11

element Command

This command is used to construct an Element object.

In This Chapter

Truss Element

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

One way is to specify an area and a *UniaxialMaterial* (page 35) identifier:

element truss $eleTag $iNode $jNode $A $matTag

the other is to specify a *Section* (page 129) identifier:

element truss $eleTag $iNode $jNode $secTag

$eleTag	unique element object tag
$iNode $jNode	end nodes
$A	cross-sectional area of element
$matTag	tag associated with previously-defined *UniaxialMaterial* (page 35)
$secTag	tag associated with previously-defined *Section* (page 129)

When constructed with a *UniaxialMaterial* (page 35) object, the truss element considers strain-rate effects, and is thus suitable for use as a damping element.

The valid queries to a truss element when creating an *ElementRecorder* (page 224) object are 'axialForce,' 'stiff,' material matArg1 matArg2...,' 'section sectArg1 sectArg2...' There will be more queries after the interface for the methods involved have been developed further.

Corotational Truss Element

This command is used to construct a Corotational Truss (CorotTruss) element object. A corotational formulation adopts a set of corotational axes which rotate with the element, thus taking into account an exact geometric transformation between local and global frames of reference.

There are two ways to construct a Corotational Truss element object:

One way is to specify an area and a *UniaxialMaterial* (page 35) identifier:

```
element corotTruss $eleTag $iNode $jNode $A $matTag
```

the other is to specify a *Section* (page 129) identifier:

```
element corotTruss $eleTag $iNode $jNode $secTag
```

$eleTag	unique element object tag
$iNode $jNode	end nodes
$A	cross-sectional area of element
$matTag	tag associated with previously-defined *UniaxialMaterial* (page 35) object
$secTag	tag associated with previously-defined *Section* (page 129) object

NOTE: When constructed with a *UniaxialMaterial* (page 35) object, the truss element considers strain-rate effects, and is thus suitable for use as a damping element.

The valid queries to a corotational truss element when creating an *ElementRecorder* (page 224) object are 'axialForce,' 'stiff,' 'material $matNum matArg1 matArg2...,' 'section $secNum sectArg1 sectArg2...'

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* (page 22):

For a two-dimensional problem:

```
element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag
```

For a three-dimensional problem:

```
element elasticBeamColumn $eleTag $iNode $jNode $A $E $G $J $Iy $Iz
        $transfTag
```

$eleTag		unique element object tag
$iNode	**$jNode**	end nodes
$A		cross-sectional area of element
$E		Young's Modulus
$G		Shear Modulus
$J		torsional moment of inertia of cross section
$Iz		second moment of area about the local z-axis
$Iy		second moment of area about the local y-axis
$transfTag		identifier for previously-defined *coordinate-transformation* (page 200) (CrdTransf) object

The valid queries to an elastic beam-column element when creating an *ElementRecorder* (page 224) object are 'stiffness' and 'force.'

NonLinear Beam-Column Elements

There are basically two types of Nonlinear Beam-Column Elements

- ***Force based elements***
- Distributed plasticity (*nonlinearBeamColumn* (page 149))
- Concentrated plasticity with elastic interior (*beamWithHinges* (page 150))
- ***Displacement based element***
- Distributed plasticity with linear curvature distribution (*dispBeamColumn* (page 151))

**************NEED FIGURE BY MHS**

Nonlinear Beam Column Element

This command is used to construct a nonlinearBeamColumn element object, which is based on the non-iterative (or iterative) force formulation, and considers the spread of plasticity along the element.

element nonlinearBeamColumn $eleTag $iNode $jNode $numIntgrPts $secTag $transfTag <-mass $massDens> <-iter $maxIters $tol>

$eleTag	unique element object tag
$iNode $jNode	end nodes
$numIntgrPts	number of integration points along the element.
$secTag	identifier for previously-defined *section* (page 129) object
$transfTag	identifier for previously-defined *coordinate-transformation* (page 200) (CrdTransf) object
$massDens	element mass density (per unit length), from which a lumped-mass matrix is formed (optional, default=0.0)
$maxIters	maximum number of iterations to undertake to satisfy element compatibility (optional, default=1)
$tol	tolerance for satisfaction of element compatibility (optional, default=10^{-16})

The integration along the element is based on Gauss-Lobatto quadrature rule (two integration points at the element ends).

The element is prismatic, i.e. the beam is represented by the *section* (page 129) model identified by **$secTag** at each integration point.

The **-iter** switch enables the iterative form of the flexibility formulation. Note that the iterative form can improve the rate of global convergence at the expense of more local element computation.

The valid queries to a nonlinear beam-column element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' and 'section $secNum secArg1 secArg2...' Where **$secNum** refers to the integration point whose data is to be output.

Beam With Hinges Element

This command is used to construct a beamWithHinges element object, which is based on the non-iterative (or iterative) flexibility formulation, and considers plasticity to be concentrated over specified hinge lengths at the element ends.

The arguments for the construction of the element depend on the dimension of the problem, *ndm* (page 22).

For a two-dimensional problem:

```
element beamWithHinges $eleTag $iNode $jNode $secTagI $HingeLengthI
        $secTagJ $HingeLengthJ $E $A $Iz $transfTag <-mass $massDens> <-
        iter $maxIters $tol>
```

For a three-dimensional problem:

```
element beamWithHinges $eleTag $iNode $jNode $secTagI $HingeLengthI
        $secTagJ $HingeLengthJ $E $A $Iz $Iy $G $J $transfTag <-mass
        $massDens> <-iter $maxIters $tol>
```

$eleTag	unique element object tag
$iNode $jNode	end nodes
$secTagI	identifier for previously-defined *section* (page 129) object corresponding to node I
$HingeLengthI	ratio of hinge length to total element length at node I
$secTagJ	identifier for previously-defined *section* (page 129) object corresponding to node J
$HingeLengthJ	ratio of hinge length to total element length at node J

$E	Young's Modulus
$A	area of element cross-section
$Iz	section moment of inertia about the section local z-axis
$Iy	section moment of inertia about the section local y-axis
$G	Shear Modulus
$J	torsional moment of inertia of cross section
$transfTag	identifier for previously-defined *coordinate-transformation* (page 200) (CrdTransf) object
$massDens	element mass density (per unit length), from which a lumped-mass matrix is formed (optional, default=0.0)
$maxIters	maximum number of iterations to undertake to satisfy element compatibility (optional, default=1)
$tol	tolerance for satisfaction of element compatibility (optional, default=10^{-16})

The **-iter** switch enables the iterative form of the flexibility formulation. Note that the iterative form can improve the rate of global convergence at the expense of more local element computation.

NOTE: The elastic properties are integrated only over the beam interior, which is considered to be linear-elastic. Forces and deformations of the inelastic regions are sampled at the hinge midpoints, using mid-point integration.

The valid queries to a beamWithHinges element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' 'rotation' (hinge rotation), or 'section $secNum secArg1 secArg2...' Where **$secNum** refers to the integration point whose data is to be output.

Displacement-Based Beam-Column Element

This command is used to construct a dispBeamColumn element object, which is a distributed-plasticity, displacement-based beam-column element.

element dispBeamColumn $eleTag $iNode $jNode $numIntgrPts $secTag $transfTag <-mass $massDens>

$eleTag	unique element object tag
$iNode $jNode	end nodes
$numIntgrPts	number of integration points along the element.
$secTag	identifier for previously-defined *section* (page 129) object

$transfTag	identifier for previously-defined *coordinate-transformation* (page 200) (CrdTransf) object
$massDens	element mass density (per unit length), from which a lumped-mass matrix is formed (optional, default=0.0)

The integration along the element is based on the Gauss-Legendre quadrature rule (REF???).

The element is prismatic, i.e. the beam is represented by the section model identified by $secTag at each integration point.

The valid queries to a displacement-based beam-column element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' and 'section $secNum secArg1 secArg2...' Where **$secNum** refers to the integration point whose data is to be output.

Zero-Length Elements

Zero-length elements connect two points at the same coordinate.

Zero-Length Element

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* (page 35) objects to represent the force-deformation relationship for the element.

element zeroLength $eleTag $iNode $jNode -mat $matTag1 $matTag2 ... -dir $dir1 $dir2 ... <-orient $x1 $x2 $x3 $yp1 $yp2 $yp3>

$eleTag	unique element object tag
$iNode $jNode	end nodes
$matTag1 $matTag2 ...	tags associated with previously-defined *UniaxialMaterials* (page 35)
$dir1 $dir2 ...	material directions:

1,2,3	translation along local x,y,z axes, respectively
4,5,6	rotation about local x,y,z axes, respectively

the orientation vectors can be specified for the element (optional):

$x1 $x2 $x3	vector components in global coordinates defining local x-axis (vector x)

$yp1 $yp2 $yp3 vector components in global coordinates defining vector yp which lies in the local x-y plane for the element:

the local z-axis is defined by the cross product between the vectors x and yp

If the optional orientation vectors are not specified, the local element axes coincide with the global axes.

The valid queries to a zero-length element when creating an *ElementRecorder* (page 224) object are 'force,' 'deformation,' 'stiffness,' and 'material $matNum matArg1 matArg2 ...' Where **$matNum** is the tag associated with the material whose data is to be output.

Zero-Length Section Element

This command is used to construct a zeroLengthSection element object, which is defined by two nodes at the same location. The nodes are connected by a single *SectionForceDeformation* (page 129) object to represent the force-deformation relationship for the element.

element zeroLengthSection $eleTag $iNode $jNode $secTag <-orient $x1 $x2 $x3 $yp1 $yp2 $yp3>

$eleTag unique element object tag

$iNode $jNode end nodes

$secTag tag associated with previously-defined *Section* (page 129) object

the orientation vectors can be specified for the element (optional):

$x1 $x2 $x3 vector components in global coordinates defining local x-axis (vector x)

$yp1 $yp2 $yp3 vector components in global coordinates defining vector yp which lies in the local x-y plane for the element:

the local z-axis is defined by the cross product between the vectors x and yp

If the optional orientation vectors are not specified, the local element axes coincide with the global axes.

The *section* (page 129) force-deformation response represented by section string P acts along the element local x-axis, and the response for code Vy along the local y-axis. The other modes of section response follow from this orientation.

The valid queries to a zero-length element when creating an *ElementRecorder* (page 224) object are 'force,' 'deformation,' 'stiffness,' and 'section secArg1 secArg2 ....'

Quadrilateral Elements

Quad Element

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

element quad $eleTag $iNode $jNode $kNode $lNode $thick $type $matTag <$pressure $rho $b1 $b2>

$eleTag	unique element object tag
$iNode $jNode $kNode $lNode	four nodes defining element boundaries, input in counter-clockwise order around the element.
$thick	element thickness (constant)
$type	string representing material behavior. Valid options depend on the *NDMaterial* (page 108) object and its available material formulations. The type parameter can be either "PlaneStrain" or "PlaneStress."
$matTag	tag associated with previously-defined *NDMaterial* (page 108) object
$pressure	surface pressure (???? sign convention????****)
$rho	element mass density (per unit volume) from which a lumped element mass matrix is computed (optional, default=0.0)
$b1 $b2	constant body forces defined in the isoparametric domain (optional, default=0.0)

Consistent nodal loads are computed from the pressure and body forces.

The valid queries to a Quad element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' 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.

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 $iNode $jNode $kNode $lNode $secTag

$eleTag	unique element object tag
$iNode $jNode $kNode $lNode	four nodes defining element boundaries, input in counter-clockwise order around the element.
$secTag	tag associated with previously-defined *SectionForceDeformation* (page 129) object. Typically, corresponds to some *PlateFiberSection* (page 144), elastic or otherwise

Should the element be required to compute a mass matrix, a consistent translational element mass matrix is computed. Rotational-inertia terms are ignored.

The valid queries to a shell element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' and 'material matArg1 matArg2 ...'

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 $iNode $jNode $kNode $lNode $matTag

$eleTag	unique element object tag
$iNode $jNode $kNode $lNode	four nodes defining element boundaries, input in counter-clockwise order around the element
$matTag	tag associated with previously-defined *NDMaterial* (page 108) object

Should the element be required to compute a mass matrix, a consistent translational element mass matrix is computed. Rotational-inertia terms are ignored.

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 $iNode $jNode $kNode $lNode type $matTag

$eleTag	unique element object tag
$iNode $jNode $kNode $lNode	four nodes defining element boundaries, input in counter-clockwise order around the element.
type	string representing material behavior. Valid options depend on the *NDMaterial* (page 108) object and its available material formulations. The type parameter can be either "PlaneStrain" or "PlaneStress."
$matTag	tag associated with previously-defined *NDMaterial* (page 108) object

Should the element be required to compute a mass matrix, a consistent translational element mass matrix is computed. Rotational-inertia terms are ignored.

The valid queries to a zero-length element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' and 'material matArg1 matArg2 ...'

Brick Elements

Standard Brick Element

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

element stdBrick $eleTag $node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8 $matTag

$eleTag	unique element object tag
$node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8	eight nodes defining element boundaries, input order is shown in the figure
$matTag	tag associated with previously-defined *NDMaterial* (page 108) object

Should the element be required to compute a mass matrix, a consistent translational element mass matrix is computed. Rotational-inertia terms are ignored.

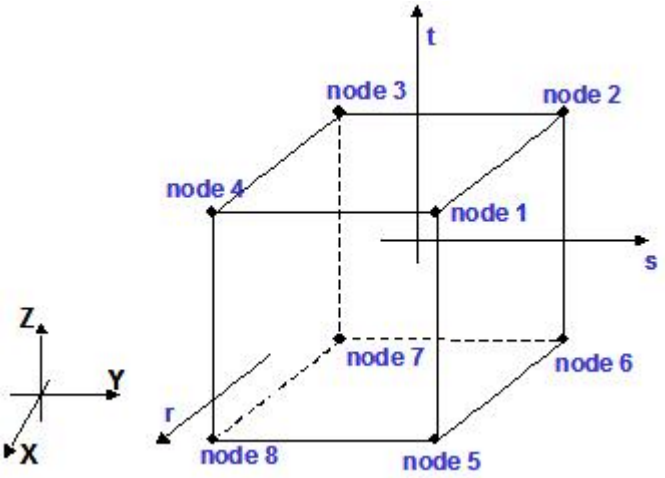


Figure 51: Node Numbering for Eight-Node Three-Dimensional Element

Bbar Brick Element

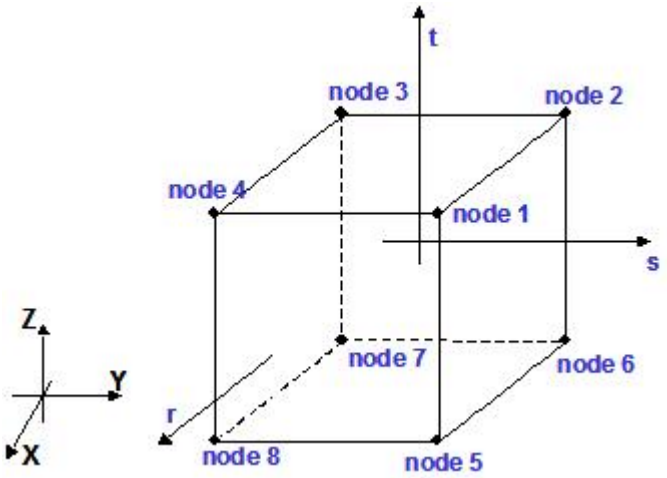
This command is used to construct an eight-node mixed volume/pressure brick element object, which uses a trilinear isoparametric formulation.

element bbarBrick $eleTag $node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8 $matTag

$eleTag	unique element object tag
$node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8	eight nodes defining element boundaries, input order is shown in the figure
$matTag	tag associated with previously-defined *NDMaterial* (page 108) object

Should the element be required to compute a mass matrix, a consistent translational element mass matrix is computed. Rotational-inertia terms are ignored

Figure 52: Node Numbering for Eight-Node Three-Dimensional Element



Eight Node Brick Element

The command is used to construct an eight-node three dimensional brick element object, which is based on tensor operation.

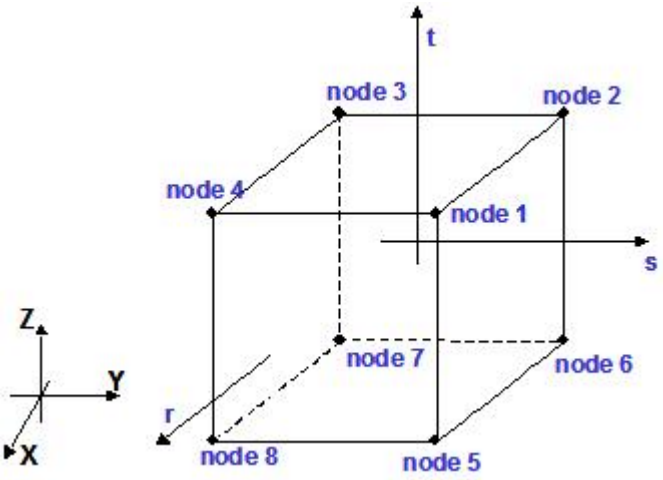
element Brick8N $eletag $node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8 $matTag $bf1 $bf2 $bf3 $massDens

$eletag	unique element object tag
$node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8	eight node coordinates, input order is shown in the figure
$matTag	material tag associated with previsouly-defined NDMaterial object
$bf1 $bf2 $bf3	body force in the direction of global coordinates x, y and z
$massDens	mass density (mass/volume)

The valid queries to a Brick8N element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' stress', 'gausspoint' or 'plastic'. The output is given as follows:

'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
'gausspoint'	the coordinates of all Gauss points are printed out
'plastic'	the equivalent deviatoric plastic strain from each Gauss point is output in the same order as the coordinates are printed

Figure 53: Node Numbering for Eight-Node Three-Dimensional Element



Twenty Node Brick Element

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

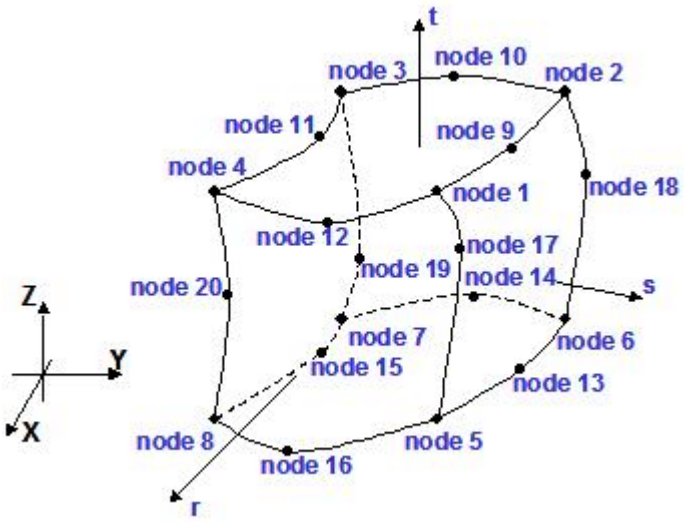
element Brick20N $eletag $node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8 $node9 $node10 $node11 $node12 $node13 $node14 $node15 $node16 $node17 $node18 $node19 $node20 $matTag $bf1 $bf2 $bf3 $massDen

$eletag	unique element object tag
$node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8 $node9 $node10 $node11 $node12 $node13 $node14 $node15 $node16 $node17 $node18 $node19 $node20	twenty node coordinates, input order is shown in the figure
$matTag	material tag associated with previsouly-defined *NDMaterial* (page 108) object
$bf1 $bf2 $bf3	body force in the direction of global coordinates x, y and z
$massDen	mass density (mass/volume)

The valid queries to a Brick20N element when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' stress', 'gausspoint' or 'plastic'. The output is given as follows:

'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
'gausspoint'	the coordinates of all Gauss points are printed out
'plastic'	the equivalent deviatoric plastic strain from each Gauss point is output in the same order as the coordinates are printed

Figure 54: Node Numbering for Twenty-Node Three-Dimensional Element



u-p-U element

This command is used to construct a u-p-U element object, which include two types: eight node element and twenty node element.

- For eight-node element:

```
element Brick8N_u_p_U $eleTag $node1 $node2 $node3 $node4 $node5
        $node6 $node7 $node8 $matTag $bf1 $bf2 $bf3 $n $alpha $soildDens
        $fluidDens $k1  $k2 $k3 $K_fluid $P
```

- For twenty-node element:

```
element Brick20N_u_p_U $eleTag $node1 $node2 $node3 $node4 $node5
        $node6 $node7 $node8 $node9 $node10 $node11 $node12 $node13
        $node14 $node15 $node16 $node17 $node18 $node19 $node20
        $matTag $bf1 $bf2 $bf3 $n $alpha $soildDens $fluidDens $k1  $k2 $k3
        $K_fluid $P
```

$eleTag	unique element object tag
$node1 $node2 $node3 $node4 $node5 $node6 $node7 $node8	node coordinate (either eight or twenty), input order is shown in the figure
$matTag	material tag associated with previsouly-difined NDMaterial object
$bf1 $bf2 $bf3	body force in the direction of global coordinates x, y and z
$n	porosity
$alpha	1-Ks/Kt (ratio of void space =1 for soils, =0.6 for concrete...)
$soildDens	solid density
$fluidDens	fluid density
$k1 $k1 $k3	coefficient of permeability in the direction of x, y and z
$K_fluid	fluid bulk modulus
$P	pressure... not used currently (set to 0.0)

The valid queries to a Brick8N_u_p_U and Brick20N_u_p_U elements when creating an *ElementRecorder* (page 224) object are 'force,' 'stiffness,' stress', 'gausspoint' or 'plastic'. The output is given as follows:

'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
'gausspoint'	the coordinates of all Gauss points are printed out
'plastic'	the equivalent deviatoric plastic strain from each Gauss point is output in the same order as the coordinates are printed

FourNodeQuadUP 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 quadUP $eleTag $iNode $jNode $kNode $lNode $thick $type $matTag $bulk $fmass $hPerm $vPerm <$b1 $b2 $t>

$eleTag	unique element object tag
$iNode, $jNode, $kNode, $lNode	Four element node (previously defined) numbers in counter-clockwise order around the element
$thick	Element thickness
$type	The string "PlaneStrain"
$matTag	Tag of an NDMaterial object (previously defined) of which the element is composed
$bulk	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 (2.2×10^6 kPa for water), and n the initial porosity.
$fmass	Fluid mass density
$hPerm	Permeability coefficient in horizontal direction
$vPerm	Permeability coefficient in vertical direction

$b1, $b2	Optional body forces in horizontal and vertical directions respectively (defaults are 0.0)
$t	Optional uniform element normal traction, positive in tension (default is 0.0)

TYPICAL RANGE OF PERMEABILITY COEFFICIENT (m/s)

Gravel	Sand	Silty Sand	Silt	Clay
$>1.0\times10^{-3}$	$1.0\times10^{-5} \sim 1.0\times10^{-3}$	$1.0\times10^{-7} \sim 1.0\times10^{-5}$	$1.0\times10^{-9} \sim 1.0\times10^{-7}$	$<1.0\times10^{-9}$

OUTPUT INTERFACE:

Pore pressure can be recorded at an element node using OpenSees *Node Recorder* (page 221):

recorder Node <-file $fileName> <-time> <-node ($node1 $node2 ...)> -dof 3 vel

Note: dof 3 is for pore pressure output.

The valid queries to a quadUP element when creating an *ElementRecorder* (page 224) are 'force', 'stiffness', or 'material matNum matArg1 matArg2 ...', where matNum represents the material object at the corresponding integration point.

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 $Nd1 $Nd2 $Nd3 $Nd4 $Mat1 $Mat2 $Mat3 $Mat4 $Mat5 $Mat6 $Mat7 $Mat8 $Mat9 $Mat10 $Mat11 $Mat12 $Mat13 <$eleHeightFac $eleWidthFac>

$eleTag	an integer identifying the element tag in the domain
$Nd1,$Nd2,$Nd3,$Nd4	tag associated with previously defined nodes
$Mat1	uniaxial material tag for left bar-slip spring at node 1
$Mat2	uniaxial material tag for right bar-slip spring at node 1

$Mat3	uniaxial material tag for interface-shear spring at node 1
$Mat4	uniaxial material tag for lower bar-slip spring at node 2
$Mat5	uniaxial material tag for upper bar-slip spring at node 2
$Mat6	uniaxial material tag for interface-shear spring at node 2
$Mat7	uniaxial material tag for left bar-slip spring at node 3
$Mat8	uniaxial material tag for right bar-slip spring at node 3
$Mat9	uniaxial material tag for interface-shear spring at node 3
$Mat10	uniaxial material tag for lower bar-slip spring at node 4
$Mat11	uniaxial material tag for upper bar-slip spring at node 4
$Mat12	uniaxial material tag for interface-shear spring at node 4
$Mat13	uniaxial material tag for shear-panel
$eleHeightFac	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	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)

NOTE:

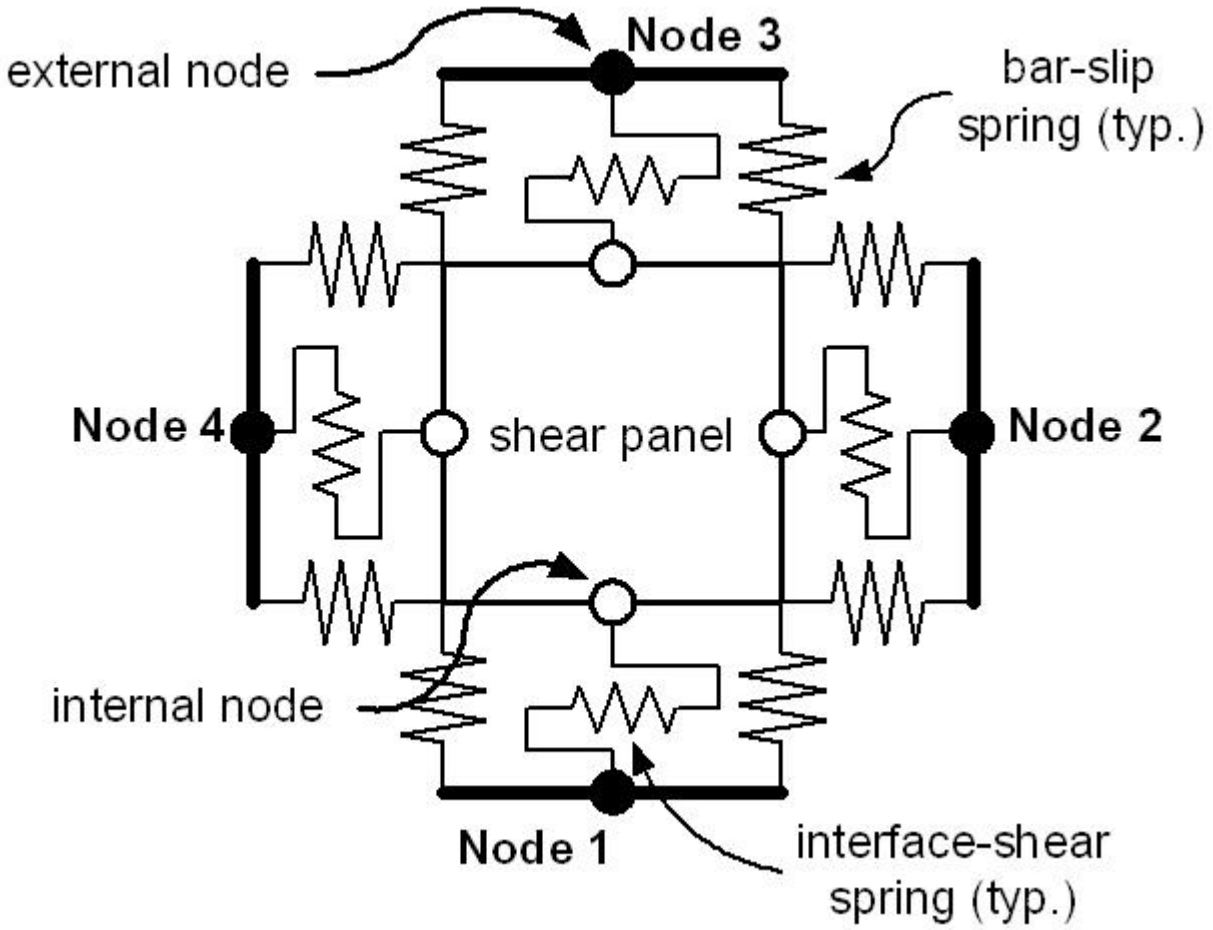

external node
Node 3
bar-slip
spring (typ.)
Node 4
shear panel
Node 2
internal node
Node 1
interface-shear
spring (typ.)

Figure 55: BeamColumnJoint Element

The valid queries to a BeamColumnJoint element when creating an *ElementRecorder* (page 224) are as follows:

'internalDisplacement' - returns the displacements of the internal joint nodes.

'externalDisplacement' - returns the displacement of the external joint nodes.

'deformation' - generates a four-column matrix in which the first column is the contribution to the total joint shear deformation of all of the bar-slip components of the joint, the second is the deformation contribution of the interface shear springs, the third is the deformation contribution of the shear-panel and the fourth is the total shear deformation of the joint.

'node1BarSlipL' - returns the load-deformation response history of the Bar-Slip spring on the Left at node 1.

'node1BarSlipR' - returns the load-deformation response history of the Bar-Slip spring on the Right at node 1.

'node1InterfaceShear'- returns the load-deformation response history of the Interface-Shear spring at node 1.

'node2BarSlipB' - returns the load-deformation response history of the Bar-Slip spring on the Bottom at node 2.

'node2BarSlipT' - returns the load-deformation response history of the Bar-Slip spring on the Top at node 2.

'node2InterfaceShear'- returns the load-deformation response history of the Interface Shear spring at node 1.

'node3BarSlipL' - returns the load-deformation response history of the Bar-Slip spring on the Left at node 3.

'node3BarSlipR' - returns the load-deformation response history of the Bar-Slip spring on the Right at node 3.

'node3InterfaceShear'- returns the load-deformation response history of the Interface-Shear spring at node 3.

'node4BarSlipB' - returns the load-deformation response history of the Bar-Slip spring on the Bottom at node 4.

'node4BarSlipT' - returns the load-deformation response history of the Bar-Slip spring on the Top at node 4.

‘node4InterfaceShear’- returns the load-deformation response history of the Interface Shear spring at node 4.

‘shearPanel’ - returns the load-deformation response history of the Shear-Panel spring.

➢ *EXAMPLE:*

main input file:

- *PR1.tcl* (page 176)

supporting files:

- *procMKPC.tcl* (page 186)
- *procUniaxialPinching.tcl* (page 70)
- *procRC.tcl* (page 188)

Beam-Column Joint Element Discussion

Beam-Column Joint Element Discussion

The example files (*PR1.tcl* (page 176), *procMKPC.tcl* (page 186), *procUniaxialPinching.tcl* (page 70), *procRC.tcl* (page 188)) create a model of a RC beam column sub-assemblage (Figure 1). The cruciform is subjected to constant gravity load at nodes 4 and 7 and pseudo-static cyclic lateral load under displacement control at node 10. The beam-column-joint region (element number 7) is represented using a beamColumnJoint element (Figure 2), and the beams and columns (element numbers 1 through 6) are modeled using the nonlinearBeamColumn element. The beam-column joint consists of 13 components that may have different material constitutive models; in this example 9 of the 13 components utilize the nonlinear material model – Pinching4. Figure 3 shows the displacement history for node 10.

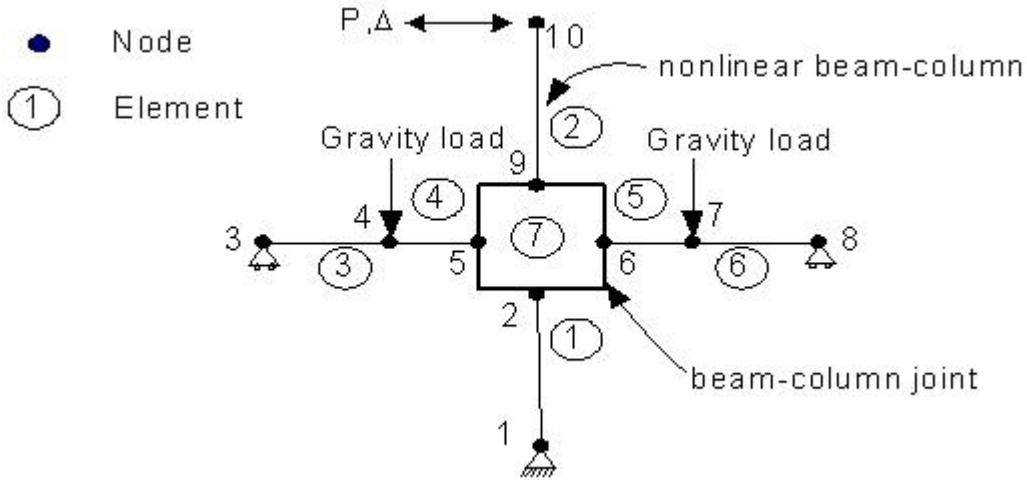


Figure 56: Cruciform model

Figure 1: Cruciform model

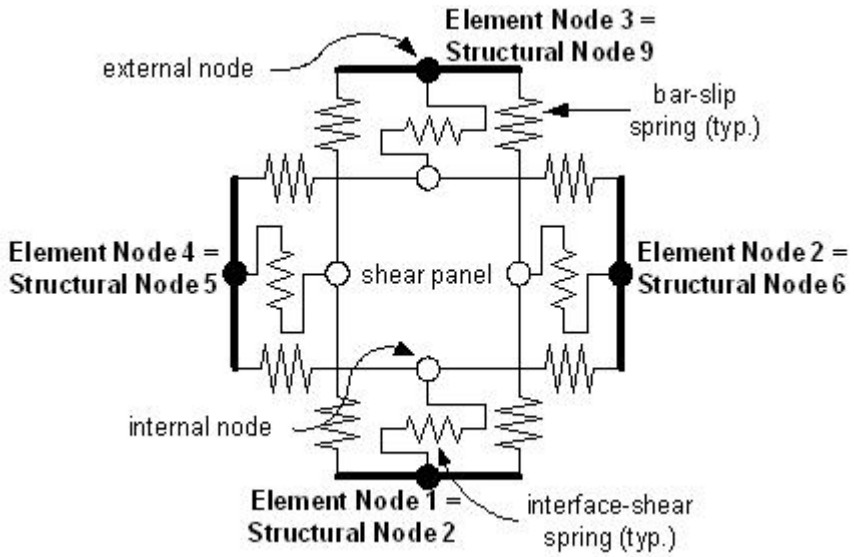


Figure 57: Beam Column Joint Element

Figure 2: Beam Column Joint Element

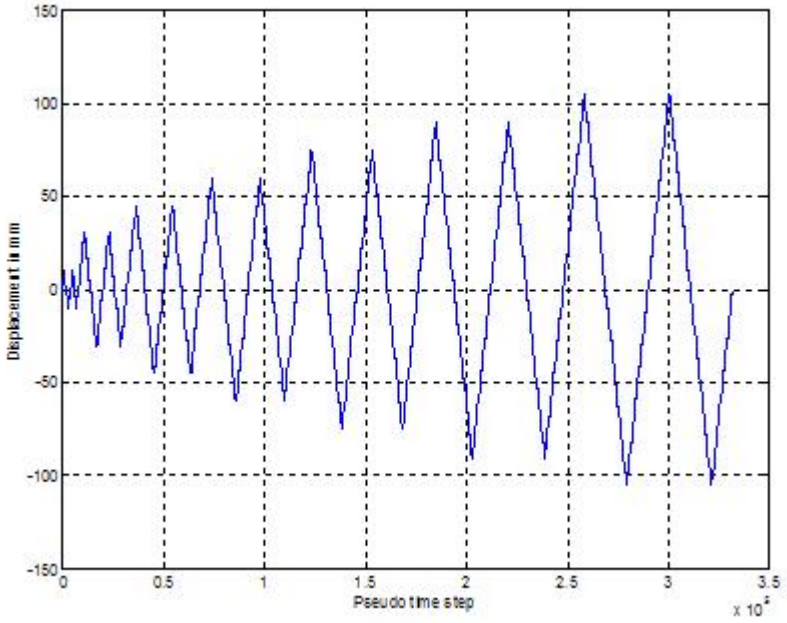

150
100
50
0
-50
-100
-150
Displacement in mm
0
0.5
1
1.5
2
2.5
3
3.5
Pseudo time step
x 10^4

Figure 58:
Displacement history
for Node 10

Figure 3: Displacement history for Node 10

Tcl Scripts:

The following tcl script files are used to run the examples:

PR1.tcl (page 176)

procMKPC.tcl (page 186)

procUniaxialPinching.tcl (page 70)

procRC.tcl (page 188)

The p-delta response of cruciform, along with the response of each of the nonlinear joint components is shown below. The shear panel response shows the moment-curvature relationships whereas the bar slips at the beam top and bottom are represented by the force-slip plots (Figure 4).

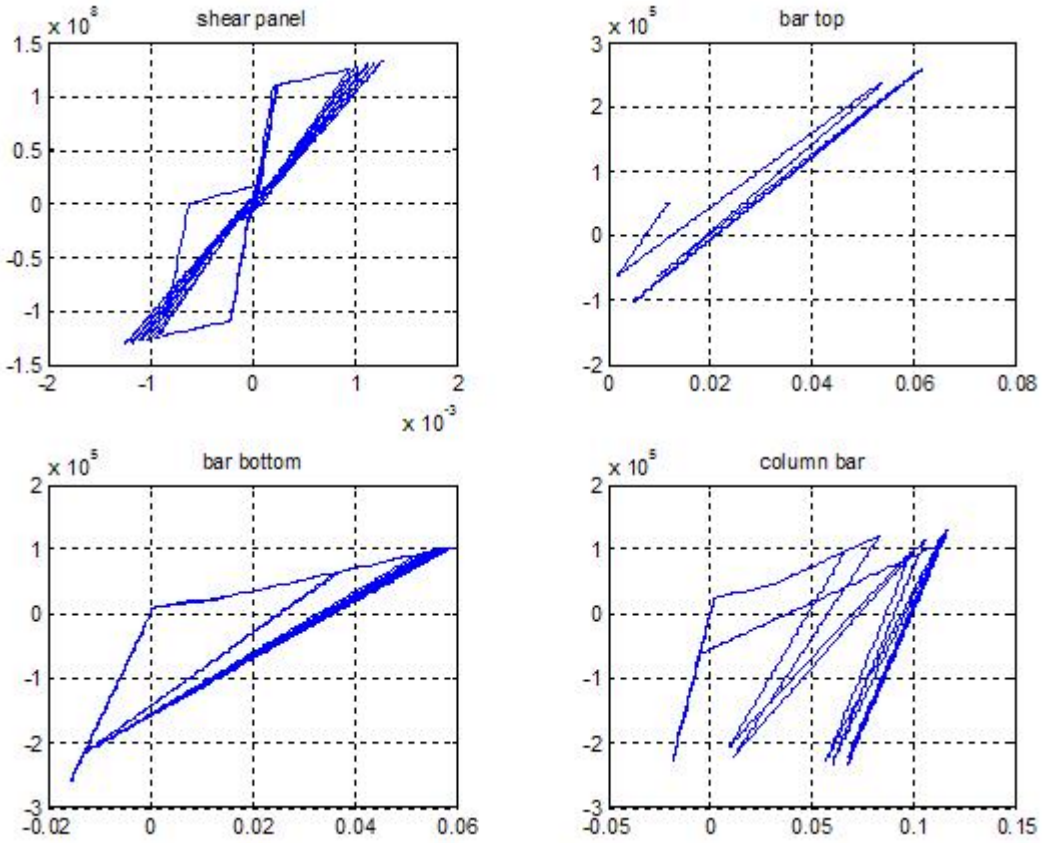

shear panel
x 10^6
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1
0
1
2
x 10^-3
bar top
x 10^5
3
2
1
0
-1
-2
0
0.02
0.04
0.06
0.08
bar bottom
x 10^5
2
1
0
-1
-2
-3
-0.02
0
0.02
0.04
0.06
column bar
x 10^5
2
1
0
-1
-2
-3
-0.05
0
0.05
0.1
0.15

Figure 59: Nonlinear Joint component response

Figure 4: Nonlinear Joint component response

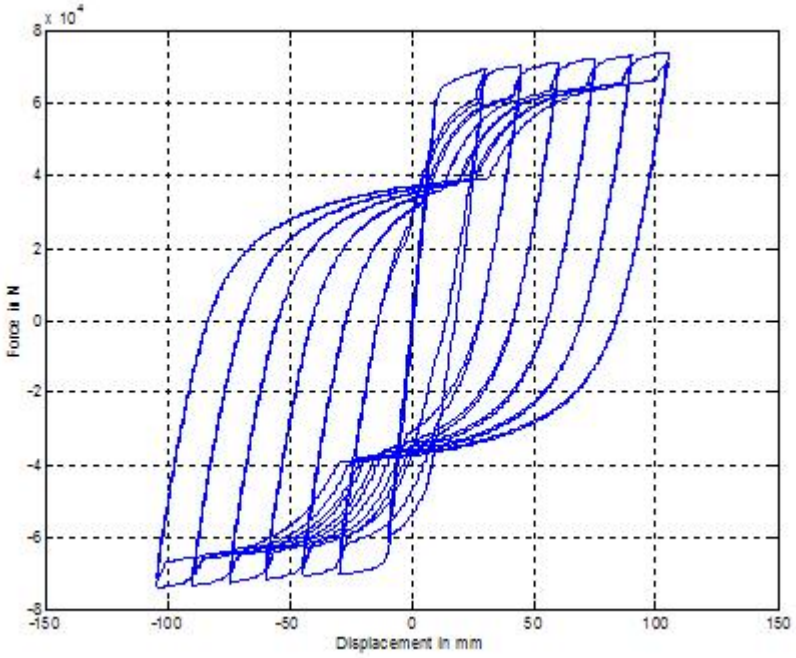


Figure 60: P-delta response of the cruciform

Figure 5. P-delta response of the cruciform

PR1.tcl

```
################################################################################
# Test example for BEAM COLUMN ELEMENT JOINT  ------- PARK RUITONG TEST SPECIMEN Unit 1
# Written: N.Mitra
# Description: 4 noded 12 dof element having 12 springs and a shear panel
# Date: Feb 16 2003
## Model consisting of a crucifix with beams and columns and a joint
```

```
## File Name: PR1.tcl
# refer to Beam-Column-Joint Element.doc for full explanation of the parameters
################################################################################

#create the ModelBuilder object
model BasicBuilder -ndm 2 -ndf 3

## unit name ---- PR1
set fName "PR1";

source procMKPC.tcl
source procUniaxialPinching.tcl
source procRC.tcl

# all dimensions are in here as MPa (conversion factors are required in certain places)
set Strfactor 145; set Lenfactor [expr 1/25.4];

## Y taken as the inplane dim. against which the bending takes place
set colY 406; set colZ 305;
set bmY 457; set bmZ 229;

# covers
set colCov 43; set bmCov1 42; set bmCov2 33; set bmCov $bmCov1;

# y,z,x dimension of the joint respectively
set JointWidth [expr $colY];  set JointHeight [expr $bmY]; set JointDepth  $colZ ;
set BeamLengthIn 645; set BeamLengthOut 1271; set ColumnLengthClear 1008;

set JointVolume [expr $JointWidth*$JointHeight*$JointDepth];

#################################################################
################## material properties of column section
#################################################################
set CUnconfFc -45.9; set CUnconfEc -0.002;
set CTSspace 60; set CTSlength 1853.53; set CTSFy 282; set CTSarea 28.3;
set CFy 498.0; set CEs 196600.0; set CsHratio 0.004216; set CAs 201.06;

procMKPC $CUnconfFc $CUnconfEc $colY $colZ $colCov $CTSspace $CTSlength $CTSFy $CTSarea $Strfactor
$Lenfactor
```

```
set CUnconfFcu [lindex $concreteProp 2]; set CUnconfEcu [lindex $concreteProp 3];
set CConfFc [lindex $concreteProp 4]; set CConfEc [lindex $concreteProp 5];
set CConfFcu [lindex $concreteProp 6]; set CConfEcu [lindex $concreteProp 7];

############################################################
############################ material properties of beam section
############################################################
set BUnconfFc -45.9; set BUnconfEc -0.002;
set BTSspace 80; set BTSlength 1036; set BTSFy 282; set BTSarea 28.3;
set BFy 294.0; set BEs 210400.0; set BAs 201.06; set BsHratio 0.002322;

procMKPC $BUnconfFc $BUnconfEc $bmY $bmZ $bmCov $BTSspace $BTSlength $BTSFy $BTSarea $Strfactor
$Lenfactor

set BUnconfFcu [lindex $concreteProp 2]; set BUnconfEcu [lindex $concreteProp 3];
set BConfFc [lindex $concreteProp 4]; set BConfEc [lindex $concreteProp 5];
set BConfFcu [lindex $concreteProp 6]; set BConfEcu [lindex $concreteProp 7];

#########################################################################
########################### details for the material models of bar slip of the beam
#####################################
set bs_fc [expr -$BUnconfFc]; set bs_fs $BFy; set bs_es $BEs; set bs_fsu 434; set bs_dbar 16; set bs_esh [expr
$BsHratio*$BEs];
set bs_wid $colZ; set bs_dep $bmY;
set bsT_nbars 5; set bsB_nbars 2;
set bs_ljoint $colY;

#########################################################################################
############################ details for the material models of bar slip of the column
#####################################
set cs_fc [expr -$CUnconfFc]; set cs_fs $CFy; set cs_es $CEs; set cs_fsu 660; set cs_dbar 16; set cs_esh [expr
$CsHratio*$CEs];
set cs_wid $colZ; set cs_dep $colY;
set cs_nbars 3;
set cs_ljoint $bmY;

#########################################################################
################ add nodes - command: node nodeId xCrd yCrd
############################################################
```

```
node 1   0.0                                        0.0
node 2   0.0                                        $ColumnLengthClear
node 3   [expr -$BeamLengthOut-$BeamLengthIn-$JointWidth/2]     [expr $ColumnLengthClear+$JointHeight/2]
node 4   [expr -$BeamLengthIn-$JointWidth/2]                    [expr $ColumnLengthClear+$JointHeight/2]
node 5   [expr -$JointWidth/2]                                  [expr $ColumnLengthClear+$JointHeight/2]
node 6   [expr $JointWidth/2]                                   [expr $ColumnLengthClear+$JointHeight/2]
node 7   [expr $BeamLengthIn+$JointWidth/2]                     [expr $ColumnLengthClear+$JointHeight/2]
node 8   [expr $BeamLengthOut+$BeamLengthIn+$JointWidth/2]      [expr $ColumnLengthClear+$JointHeight/2]
node 9   0.0                                         [expr $ColumnLengthClear+$JointHeight]
node 10  0.0                                         [expr 2*$ColumnLengthClear+$JointHeight]

# add material Properties - command: uniaxialMaterial matType matTag ...
#command: uniaxialMaterial Elastic tag? E?
uniaxialMaterial Elastic 1 10000000000.0

#########################################################################
############## inelastic beam column elements
#########################################################################
uniaxialMaterial Concrete01 10 $BUnconfFc $BUnconfEc $BUnconfFcu $BUnconfEcu

uniaxialMaterial Concrete01 20 $BConfFc $BConfEc $BConfFcu $BConfEcu

uniaxialMaterial Steel02 30 $BFy $BEs $BsHratio 18.5 0.925 0.15 0.0 0.4 0.0 0.5

uniaxialMaterial Concrete01 40 $CUnconfFc $CUnconfEc $CUnconfFcu $CUnconfEcu

uniaxialMaterial Concrete01 50 $CConfFc $CConfEc $CConfFcu $CConfEcu

uniaxialMaterial Steel02 60 $CFy $CEs $CsHratio 18.5 0.925 0.15 0.0 0.4 0.0 0.5

########### for columns ////////////////////////////////////////////////////////////////////////
set z [expr $colZ/2.0]; set y [expr $colY/2.0];

section Fiber 1 {
patch rect 50 8 1    [expr $colCov-$y]      [expr $colCov-$z]   [expr $y-$colCov]      [expr $z-$colCov]
patch rect 40 2 1    [expr -$y]             [expr $colCov-$z]   [expr $colCov-$y]      [expr $z-$colCov]
patch rect 40 2 1    [expr $y-$colCov]      [expr $colCov-$z]   [expr $y]              [expr $z-$colCov]
```

```
patch rect 40 8 1    [expr -$y]          [expr -$z]         [expr $y]          [expr $colCov-$z]
patch rect 40 8 1    [expr -$y]          [expr $z-$colCov]  [expr $y]          [expr $z]

layer straight 60 3 $CAs    [expr $y-$colCov]       [expr $colCov-$z]  [expr $y-$colCov]       [expr $z-$colCov]
layer straight 60 2 $CAs    0.0                     [expr $colCov-$z]  0.0                     [expr $z-$colCov]
layer straight 60 3 $CAs    [expr $colCov-$y]       [expr $colCov-$z]  [expr $colCov-$y]       [expr $z-$colCov]
}

##################### for beams ////////////////////////////////////////////////////////////////////
set z [expr $bmZ/2.0]; set y [expr $bmY/2.0];

section Fiber 2 {
patch rect 20 8 1    [expr $bmCov1-$y]       [expr $bmCov1-$z]    [expr $y-$bmCov1]     [expr $z-$bmCov1]
patch rect 10 2 1    [expr -$y]              [expr $bmCov1-$z]    [expr $bmCov1-$y]     [expr $z-$bmCov1]

patch rect 10 2 1    [expr $y-$bmCov1]       [expr $bmCov1-$z]    [expr $y]             [expr $z-$bmCov1]

patch rect 10 8 1    [expr -$y]              [expr -$z]           [expr $y]             [expr $bmCov1-$z]
patch rect 10 8 1    [expr -$y]              [expr $z-$bmCov1]    [expr $y]             [expr $z]

layer straight 30 3 $BAs   [expr $y-$bmCov1]           [expr $bmCov1-$z]    [expr $y-$bmCov1]           [expr $z -
$bmCov1]
layer straight 30 2 $BAs   [expr $y-$bmCov1-$bmCov2]   [expr $bmCov1-$z]    [expr $y-$bmCov1-$bmCov2]
[expr $z - $bmCov1]
layer straight 30 2 $BAs   [expr $bmCov1-$y]           [expr $bmCov1-$z]    [expr $bmCov1-$y]           [expr $z -
$bmCov1]
}

###############////////////////////////////////////////////////////////////////////////////////////

## add geometric transformation -command: geomTransf transfType ...

## geomTransf Linear tag?
geomTransf Linear 1
geomTransf Linear 2

element nonlinearBeamColumn 1 1 2 5 1 2
element nonlinearBeamColumn 2 9 10 5 1 2
```

```
element nonlinearBeamColumn 3 3 4 3 2 1
element nonlinearBeamColumn 4 4 5 2 2 1
element nonlinearBeamColumn 5 6 7 2 2 1
element nonlinearBeamColumn 6 7 8 3 2 1

##### end element formation as well as material defination for beams and columns ######################
#############################################################################################



# for beam bottom
set matID1 21
set matID2 22

uniaxialMaterial BarSlip $matID1 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsB_nbars $bs_wid $bs_dep strong beamBot
uniaxialMaterial BarSlip $matID2 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsB_nbars $bs_wid $bs_dep strong beamBot

## %%%%%%%%%%%%%%% equivalent statement can be made in other way
#uniaxialMaterial BarSlip $matID1 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsB_nbars $bs_wid $bs_dep 1.0 strong beamBot damage
#uniaxialMaterial BarSlip $matID2 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsB_nbars $bs_wid $bs_dep 1.0 strong beamBot damage

# for beam top
set matID3 31
set matID4 32

uniaxialMaterial BarSlip $matID3 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsT_nbars $bs_wid $bs_dep strong beamTop
uniaxialMaterial BarSlip $matID4 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsT_nbars $bs_wid $bs_dep strong beamTop

## %%%%%%%%%%%%%%% equivalent statement can be made in other way
#uniaxialMaterial BarSlip $matID3 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsT_nbars $bs_wid $bs_dep 1.0 strong beamTop damage
#uniaxialMaterial BarSlip $matID4 $bs_fc $bs_fs $bs_es $bs_fsu $bs_esh $bs_dbar $bs_ljoint $bsT_nbars $bs_wid $bs_dep 1.0 strong beamTop damage

# for columns
set matID5 41
```

```
set matID6 42

set matID7 43

set matID8 44


uniaxialMaterial BarSlip $matID5 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep strong column

uniaxialMaterial BarSlip $matID6 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep strong column

uniaxialMaterial BarSlip $matID7 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep strong column

uniaxialMaterial BarSlip $matID8 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep strong column


## %%%%%%%%%%%%%%% equivalent statement can be made in other way

#uniaxialMaterial BarSlip $matID5 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep 1.0 strong column damage

#uniaxialMaterial BarSlip $matID6 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep 1.0 strong column damage

#uniaxialMaterial BarSlip $matID7 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep 1.0 strong column damage

#uniaxialMaterial BarSlip $matID8 $cs_fc $cs_fs $cs_es $cs_fsu $cs_esh $cs_dbar $cs_ljoint $cs_nbars $cs_wid $cs_dep 1.0 strong column damage


###################### end material formation for bar slip
#############################################################




############################## material for shear panel
#####################################################################

## Positive/Negative envelope Stress

set p1 2.1932; set p2 4.0872; set p3 4.4862; set p4 [expr $p3*1e-3];


##                  stress1          stress2          stress3          stress4

set pEnvStrsp [list   [expr $p1*$JointVolume]   [expr $p2*$JointVolume]   [expr $p3*$JointVolume]   [expr $p4*$JointVolume]]

set nEnvStrsp [list   [expr -$p1*$JointVolume]   [expr -$p2*$JointVolume]   [expr -$p3*$JointVolume]   [expr -$p4*$JointVolume]]


## Positive/Negative envelope Strain

##                strain1  strain2  strain3  strain4

set pEnvStnsp [list  0.0002   0.004465   0.0131    0.0269]

set nEnvStnsp [list  -0.0002  -0.004465   -0.0131   -0.0269]
```

```
## Ratio of maximum deformation at which reloading begins
##                Pos_env.   Neg_env.
set rDispsp [list   0.25      0.25]



## Ratio of envelope force (corresponding to maximum deformation) at which reloading begins
###                Pos_env.   Neg_env.
set rForcesp [list 0.15       0.15]

## Ratio of monotonic strength developed upon unloading
###             Pos_env.   Neg_env.
set uForcesp [list 0.0    0.0]

## Coefficients for Unloading Stiffness degradation
##               gammaK1   gammaK2  gammaK3  gammaK4   gammaKLimit
set gammaKsp [list   1.13364492409642    0.0    0.10111033064469   0.0       0.91652498468618]
#set gammaKsp [list   0.0    0.0    0.0   0.0       0.0]

#### Coefficients for Reloading Stiffness degradation
###               gammaD1   gammaD2  gammaD3  gammaD4   gammaDLimit
set gammaDsp [list    0.12     0.0      0.23     0.0       0.95]
#set gammaDsp [list    0.0     0.0      0.0     0.0       0.0]

#### Coefficients for Strength degradation
###               gammaF1   gammaF2  gammaF3  gammaF4   gammaFLimit
set gammaFsp [list   1.11      0.0      0.319    0.0       0.125]
#set gammaFsp [list   0.0      0.0      0.0    0.0       0.0]



set gammaEsp 10.0



uniaxialMaterial Pinching4 5 [lindex $pEnvStrsp 0] [lindex $pEnvStnsp 0] \
 [lindex $pEnvStrsp 1] [lindex $pEnvStnsp 1] [lindex $pEnvStrsp 2] \
 [lindex $pEnvStnsp 2] [lindex $pEnvStrsp 3] [lindex $pEnvStnsp 3] \
 [lindex $nEnvStrsp 0] [lindex $nEnvStnsp 0] \
```

```
 [lindex $nEnvStrsp 1] [lindex $nEnvStnsp 1] [lindex $nEnvStrsp 2] \
 [lindex $nEnvStnsp 2] [lindex $nEnvStrsp 3] [lindex $nEnvStnsp 3] \
 [lindex $rDispsp 0] [lindex $rForcesp 0] [lindex $uForcesp 0] \
 [lindex $rDispsp 1] [lindex $rForcesp 1] [lindex $uForcesp 1] \
 [lindex $gammaKsp 0] [lindex $gammaKsp 1] [lindex $gammaKsp 2] [lindex $gammaKsp 3] [lindex $gammaKsp 4] \
 [lindex $gammaDsp 0] [lindex $gammaDsp 1] [lindex $gammaDsp 2] [lindex $gammaDsp 3] [lindex $gammaDsp 4] \
 [lindex $gammaFsp 0] [lindex $gammaFsp 1] [lindex $gammaFsp 2] [lindex $gammaFsp 3] [lindex $gammaFsp 4] \
 $gammaEsp energy

##################### end material formation for shear panel
################################################

##element BeamColumnJoint tag? iNode? jNode? kNode? lNode? matTag1? matTag2? matTag3? matTag4?
##              matTag5? matTag6? matTag7? matTag8? matTag9? matTag10? matTag11? matTag12? matTag13?
##              <element Height factor?> <element Width factor?>
## please note: the four nodes are in anticlockwise direction around the element
##          requires material tags for all 13 different components within the element.

##          the first 12 being that of spring and the last of the shear panel

element beamColumnJoint 7 2 6 9 5 41 42 1 21 31 1 43 44 1 22 32 1 5
#element beamColumnJoint 7 2 6 9 5 1 1 1 1 1 1 1 1 1 1 1 1 1

## %%%%%%%%%%%%%%% equivalent statement can be made in other way
#element beamColumnJoint 7 2 6 9 5 41 42 1 21 31 1 43 44 1 22 32 1 5 1.0 1.0
#element beamColumnJoint 7 2 6 9 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1.0 1.0



# set the boundary conditions - command: fix nodeID xResrnt? yRestrnt?
fix 1 1 1 0
fix 2 0 0 0
fix 3 0 1 0
fix 4 0 0 0
fix 5 0 0 0
fix 6 0 0 0
fix 7 0 0 0
fix 8 0 1 0
fix 9 0 0 0
```

```
fix 10 0 0 0

pattern Plain 2 Linear {
        load 4 0 -55000 0 -const
        load 7 0 -55000 0 -const
}

system ProfileSPD
constraints Plain
integrator LoadControl 0 1 0 0
test NormDispIncr 1e-8 150
algorithm Newton
numberer RCM
analysis Static
analyze 1
loadConst -time 0.0

pattern Plain 1 Linear {
        #load nd? Fx? Fy? Mz?
        load 10 1 0 0
}


set rbbt "_RBbt"; set rbtp "_RBtp"; set dlcbr "_DLCbr"; set sp "_Sp"; set jdf "_Jdf";
set lbbt "_LBbt"; set lbtp "_LBtp"; set drcbr "_DRCbr"; set ulcbr "_ULCbr"; set urcbr "_URCbr";
set RBbt [concat $fName$rbbt]; set RBtp [concat $fName$rbtp]; set Sp [concat $fName$sp];
set LBbt [concat $fName$lbbt]; set LBtp [concat $fName$lbtp]; set DLCbr [concat $fName$dlcbr];
set DRCbr [concat $fName$drcbr]; set URCbr [concat $fName$urcbr]; set ULCbr [concat $fName$ulcbr];
set Jdf [concat $fName$jdf];

recorder Node $fName.out disp -load -node 10 -dof 1
recorder Element 7 -file $RBbt.out node2BarSlipB stressStrain
recorder Element 7 -file $RBtp.out node2BarSlipT stressStrain
recorder Element 7 -file $LBbt.out node4BarSlipB stressStrain
recorder Element 7 -file $LBtp.out node4BarSlipT stressStrain
recorder Element 7 -file $DLCbr.out node1BarSlipL stressStrain
recorder Element 7 -file $DRCbr.out node1BarSlipR stressStrain
recorder Element 7 -file $ULCbr.out node3BarSlipL stressStrain
```

```
recorder Element 7 -file $URCbr.out node3BarSlipR stressStrain
recorder Element 7 -file $Sp.out shearpanel stressStrain
recorder Element 7 -file $Jdf.out deformation

set peakpts [list 0.1 10 10 30 30 45 45 60 60 75 75 90 90 105 105]
set increment 10
set nodeTag 10
set dofTag 1
procRC $increment $nodeTag $dofTag $peakpts

# print the results at node and at all elements
print node
#print element
```

procMKPC.tcl

```
##########################################################################################################
##########
#
#                              procMKPC.tcl
## procedure for evaluating the confined concrete material envelope points based upon the modified
## kent park procedure. The procedure takes in the unconfined concrete and confining steel properties.
## created : NM (nmitra@u.washington.edu) dated: Dec. 2002
##########################################################################################################
##########

proc procMKPC { CUnconfFc CUnconfEc Y Z Cov TSspace TSlength TSFy TSarea Strfactor Lenfactor } {

        set CUnconfEcu -0.004;
        set SecWid [expr $Lenfactor*$Z]; set SecDep [expr $Lenfactor*$Y]; set cover [expr $Lenfactor*$Cov];
        set UFc [expr -$Strfactor*$CUnconfFc]; set Ue0 [expr -$CUnconfEc]; set Uecu [expr -$CUnconfEcu];
        set hoopSpc [expr $Lenfactor*$TSspace]; set hoopLngth [expr $Lenfactor*$TSlength];
        set hoopFy [expr $Strfactor*$TSFy]; set hoopArea [expr $TSarea*$Lenfactor*$Lenfactor];

# ratio of volume of rectangular steel hoops to volumne of concrete core measured to outside of peripheral hoops
        set rhoS [expr ($hoopLngth*$hoopArea)/(($SecWid-2*$cover)*($SecDep-2*$cover)*$hoopSpc)];
# width of concrete core measured to outside of peripheral hoop
        set b [expr $SecWid - 2*$cover];
```

```
        set temp [expr $b/$hoopSpc]
        set e50u [expr (3+0.002*$UFc)/($UFc - 1000)]; set e50h [expr 3*$rhoS*pow($temp,0.5)/4];
        set Zm [expr 0.5*($UFc-1000)/(3+0.002*$UFc)]; set Z [expr 0.5/($e50u + $e50h - $Ue0)];
        set K [expr (1 + $rhoS*$hoopFy/$UFc)];

# unconfined ultimate compressive strength
        set UFcu [expr -$UFc*(1-$Zm*($Uecu-$Ue0))/$Strfactor];
#cracking strain in confined concrete
        set Ce0 [expr -$K*$Ue0];
# cracking stress in confined concrete
        set CFc [expr -$K*$UFc/$Strfactor];
# ultimate stress in confined concrete
        set CFcu [expr 0.2*$CFc];
# ultimate strain in confined concrete
        set Cecu [expr -(0.8/$Z - $Ce0)];

        global concreteProp;

        set concreteProp [list $CUnconfFc $CUnconfEc $UFcu $CUnconfEcu $CFc $Ce0 $CFcu $Cecu];

        #puts [lindex $concreteProp 0]

        return $concreteProp;
}
```

procUniaxialPinching.tcl

```
##########################################################################################################
########
#                                                                                                        #
#                          procUniaxialPinching.tcl                                                      #
# procedure for activating the pinching material given its parameters in the form of list                #
# created NM (nmitra@u.washington.edu) dated : Feb 2002                                                  #
##########################################################################################################
########
proc procUniaxialPinching { materialTag pEnvelopeStress nEnvelopeStress pEnvelopeStrain nEnvelopeStrain rDisp
rForce uForce gammaK gammaD gammaF gammaE damage} {



 # add material - command: uniaxialMaterial ..... paramaters as shown
 #uniaxialMaterial Pinching4 tag
```

```
####          stress1P strain1P stress2P strain2P stress3P strain3P stress4P strain4P
####          stress1N strain1N stress2N strain2N stress3N strain3N stress4N strain4N
####          rDispP rForceP uForceP rDispN rForceN uForceN
####          gammaK1 gammaK2 gammaK3 gammaK4 gammaKLimit
####          gammaD1 gammaD2 gammaD3 gammaD4 gammaDLimit
####          gammaF1 gammaF2 gammaF3 gammaF4 gammaFLimit gammaE $damage


uniaxialMaterial Pinching4 $materialTag [lindex $pEnvelopeStress 0] [lindex $pEnvelopeStrain 0] \
 [lindex $pEnvelopeStress 1] [lindex $pEnvelopeStrain 1] [lindex $pEnvelopeStress 2] \
 [lindex $pEnvelopeStrain 2] [lindex $pEnvelopeStress 3] [lindex $pEnvelopeStrain 3] \
 [lindex $nEnvelopeStress 0] [lindex $nEnvelopeStrain 0] \
 [lindex $nEnvelopeStress 1] [lindex $nEnvelopeStrain 1] [lindex $nEnvelopeStress 2] \
 [lindex $nEnvelopeStrain 2] [lindex $nEnvelopeStress 3] [lindex $nEnvelopeStrain 3] \
 [lindex $rDisp 0] [lindex $rForce 0] [lindex $uForce 0] \
 [lindex $rDisp 1] [lindex $rForce 1] [lindex $uForce 1] \
 [lindex $gammaK 0] [lindex $gammaK 1] [lindex $gammaK 2] [lindex $gammaK 3] [lindex $gammaK 4] \
 [lindex $gammaD 0] [lindex $gammaD 1] [lindex $gammaD 2] [lindex $gammaD 3] [lindex $gammaD 4] \
 [lindex $gammaF 0] [lindex $gammaF 1] [lindex $gammaF 2] [lindex $gammaF 3] [lindex $gammaF 4] \
 $gammaE $damage


}
```

procRC.tcl

```
##########################################################################################################
###########
#
#                              procRC.tcl
## procedure for setting up a reversed cycle loading scheme. The input are mainly the
## peak points for the loading.
## The procedure primarily uses Displacement control for loading, if it fails uses ArcLength control
## created : NM (nmitra@u.washington.edu) dated: Sep 2002
##########################################################################################################
###########



proc procRC { incre nodeTag dofTag peakpts } {
```

```
set displayTag 0;
set numTimes 150;

        set x [lindex $peakpts 0];
        set dU [expr $x/$incre];
        #set dU0 [expr $dU/1000];
        set dU0 [expr $dU/10000];
        integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU $dU
        analysis Static
        analyze $incre

        integrator DisplacementControl $nodeTag $dofTag 0.0 1 [expr -$dU] [expr -$dU]
        analyze [expr 2*$incre]

        integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU $dU
        analyze $incre

        ## end the first peak pt start for others

        for {set j 1} {$j < [llength $peakpts]} {incr j 1} {

                set y [lindex $peakpts $j]
                set dSt [expr $y/$dU]
                set dS [expr int($dSt)]

                test NormDispIncr 1e-8 $numTimes $displayTag
                algorithm Newton

############## start loading cycle ###################
                set t 0;

                while {$t != $dS} {
                        integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU $dU
                        set ok [analyze 1]
                        incr t 1;

                        if {$ok != 0} {
#                               if {$t == $dS} {break};
```

```
                                puts "Displacement control failed ..... trying Arc-Length control"
                                set currentDisp [nodeDisp $nodeTag $dofTag]
                                puts "Current Displacement is $currentDisp"
#                               algorithm Linear
                                test NormDispIncr 1e-6 $numTimes $displayTag
                                #algorithm ModifiedNewton
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU0 $dU0
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 10 $dU0 $dU0
                                integrator ArcLength [expr $dU0] 1.0
#                               set ok [analyze 1]
                                analyze 1
                        }
#                       puts "that worked ..... back to regular Newton "
                        test NormDispIncr 1e-8 $numTimes $displayTag
#                       algorithm Newton
                }

################### end of loading cycle, start unloading cycle ########

                set t 0;

                while {$t != [expr 2*$dS]} {
                        integrator DisplacementControl $nodeTag $dofTag 0.0 1 [expr -$dU] [expr -$dU]
                        set ok [analyze 1]
                        incr t 1;

                        if {$ok != 0} {
#                               if {$t == [expr 2*$dS]} {break};
                                puts "Displacement control failed ..... trying Arc-Length control"
                                set currentDisp [nodeDisp $nodeTag $dofTag]
                                puts "Current Displacement is $currentDisp"
#                               algorithm Linear

                                test NormDispIncr 1e-6 $numTimes $displayTag
                                #algorithm ModifiedNewton
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 1 [expr -$dU0] [expr -
$dU0]
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 10 [expr -$dU0] [expr -
$dU0]
```

```
                                integrator ArcLength [expr $dU0] 1.0
#                               set ok [analyze 1]
                                analyze 1
                        }
#                       puts "that worked .... back to regular Newton "
                        test NormDispIncr 1e-8 $numTimes $displayTag
#                       algorithm Newton
                }

############## end of unloading cycle, start reloading cycle ############

                set t 0;

                while {$t != $dS} {
                        integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU $dU
                        set ok [analyze 1]
                        incr t 1;

                        if {$ok != 0} {
#                               if {$t == $dS} {break};
                                puts "Displacement control failed ..... trying Arc-Length control"
                                set currentDisp [nodeDisp $nodeTag $dofTag]
                                puts "Current Displacement is $currentDisp"
#                               algorithm Linear
                                test NormDispIncr 1e-6 $numTimes $displayTag
                                #algorithm ModifiedNewton
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 1 $dU0 $dU0
#                               integrator DisplacementControl $nodeTag $dofTag 0.0 10 $dU0 $dU0
                                integrator ArcLength [expr $dU0] 1.0
#                               set ok [analyze 1]
                                analyze 1
                        }
#                       puts "that worked .... back to regular Newton "
                        test NormDispIncr 1e-8 $numTimes $displayTag
#                       algorithm Newton
                }

######## reloading cycle completed ###############################
```

```
                if {$ok == 0} {
                        puts "analysis succesful at $y mm displacement";
                } else {
                        puts "analysis could not proceed fine beyond $y mm displacement";
                }

        }
}
```

CHAPTER 12

block Command

The block command is used to generate meshes of quadrilateral or brick finite element.

The *block2D* (page 194) command generates meshes of quadrilateral elements in two or three dimensions. In three dimensions, a two-dimensional surface appropriate for shell analysis is generated.

The *block3D* (page 196) command generates three-dimensional meshes of eight-node brick solid element.

In This Chapter

block2D Command

The block2D command generates meshes of quadrilateral elements in two or three dimensions. In three dimensions, a two-dimensional surface appropriate for shell analysis is generated.

```
block2d $nx $ny $e1 $n1 element (element arguments) {
        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>
}
```

$nx	**$ny**	number of elements in the local x and y directions of the block, respectively
$e1	$n1	starting element and node number for generation, respectively
element		string defining which *quadrilateral element* (page 155, page 156, page 154, page 155) is being used
(element arguments)		list of data parameters for element being used. This list may include, but is not limited to, a $matTag number
{$x1, ... $x9} {$y1, ... $y9}		coordinates of the block elements in two dimensions
{$z1, ... $z9}		coordinate of the block elements in third dimension (optional, default=0.0)

Only the first four nodes (1-4) are required. Nodes 5-9 are used to generate curved meshes. The user may specify any combination of nodes 5-9, omitting some of them if desired.

NOTE: this command only recognizes variable substitutions when the command arguments are placed in quotes rather than braces

EXAMPLE:

block2d $nx $ny $e1 $n1 element (element arguments) {

1 $x1 $y1 <$z1>

2 $x2 $y2 <$z2>

3 $x3 $y3 <$z3>

4 $x4 $y4 <$z4>

<5> <$x5> <$y5> <$z5>

<6> <$x6> <$y6> <$z6>

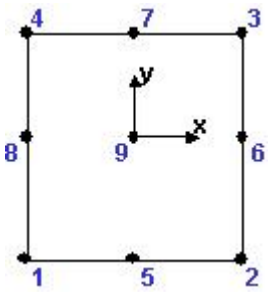
<7> <$x7> <$y7> <$z7>

<8> <$x8> <$y8> <$z8>

<9> <$x9> <$y9> <$z9>

}

Figure 61: Node Numbering for Nine-Node block2D



block3D Command

The block3D command generates three-dimensional meshes of eight-node brick solid element.

```
block3d $nx $ny $nz $e1 $n1 element elementArgs {
        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>
        ...
        <27> <$x27> <$y27> <$z27>
}
```

$nx $ny $nz	number of elements in the local x,y and z-direction of the block
$e1	starting element number for generation
$n1	starting node number for generation
element	define which *brick element* (page 158, page 157) is being used
elementArgs	list of data parameters for element being used. This list may include, but is not limited to, a $matTag number
{$x1, ... $x27} {$y1, ... $y27} {$z1, ... $z27}	coordinates of the block elements in three dimensions.

NOTE: this command only recognizes variable substitutions when the command

arguments are placed in quotes rather than braces

Only the first eight nodes (1-8) are required. Nodes 9-27 are used to generate curved meshes. The user may specify any combination of nodes 9-27, omitting some of them if desired.

CHAPTER 13

region Command

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.

region $regTag <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-ele all> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-node all> <-rayleigh $alphaM $betaK $betaKinit $betaKcomm>

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. If nodes are specified, the region includes these nodes and all elements whose external nodes are prescribed.

$regTag	unique integer tag
$ele1 $ele2 ...	tags of elements -- selected elements in domain (optional, default: omitted)
$startEle $endEle	tag for start and end elements -- range of selected elements in domain (optional, default: all)
all	all elements in domain (optional & default)
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero)
OR:	
$regTag	unique integer tag
$node1 $node2 ...	node tags -- select nodes in domain (optional, default: all)
$startNode $endNode	tag for start and end nodes -- range of nodes in domain (optional, default: all)
all	all nodes in domain (optional & default)
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero)

The damping matrix D is specified as a combination of stiffness and mass-proportional damping matrices:

D = $alphaM * M + $betaK * Kcurrent +$betaKinit * Kinit + $betaKcomm * KlastCommit

The mass and stiffness matrices are defined as:

M	mass matrix used to calculate Rayleigh Damping
Kcurrent	stiffness matrix at current state determination used to calculate Rayleigh Damping
Kinit	stiffness matrix at initial state determination used to calculate Rayleigh Damping
KlastCommit	stiffness matrix at last-committed state determination used to calculate Rayleigh Damping

NOTE: a region is defined by either nodes or elements, not both.

EXAMPLE:

region 1 -ele 1 5 -eleRange 10 15

region 2 -node 2 4 6 -nodeRange 9 12

CHAPTER 14

Geometric Transformation Command

The geometric-transformation command (geomTransf) 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. Each type is outlined below.

In This Chapter

Linear Transformation

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.

For a two-dimensional problem:

```
geomTransf Linear $transfTag <-jntOffset $dXi $dYi $dXj $dYj>
```

For a three-dimensional problem:

```
geomTransf Linear $transfTag $vecxzX $vecxzY $vecxzZ <-jntOffset $dXi $dYi
        $dZi $dXj $dYj $dZj>
```

$transfTag unique identifier for CrdTransf object

$vecxzX $vecxzY $vecxzZ	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 x-axis and the vecxz vector. 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.
$dXi $dYi $dZi	joint offset values -- absolute 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) (optional)
$dXj $dYj $dZj	joint offset values -- absolute 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) (optional)

The element coordinate system is specified as follows:

The x-axis is the axis connecting the two element nodes; the y- and z-axes are then defined using a vector that lies on a plane parallel to the local x-z plane -- vecxz. The y-axis is defined by taking the cross product of the x-axis and the vecxz vector. The section is attached to the element such that the y-z coordinate system used to specify the section corresponds to the y-z axes of the element.

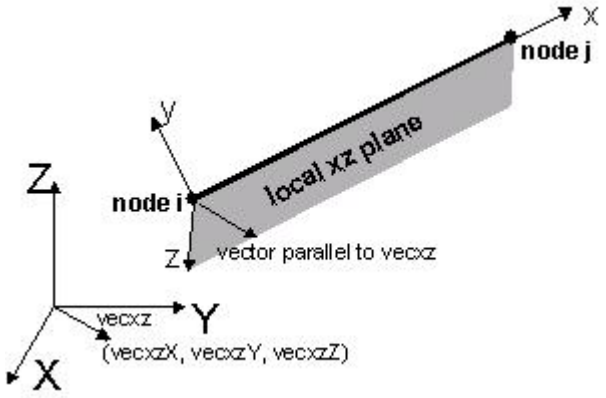


Figure 62: Definition of the Local Coordinate System

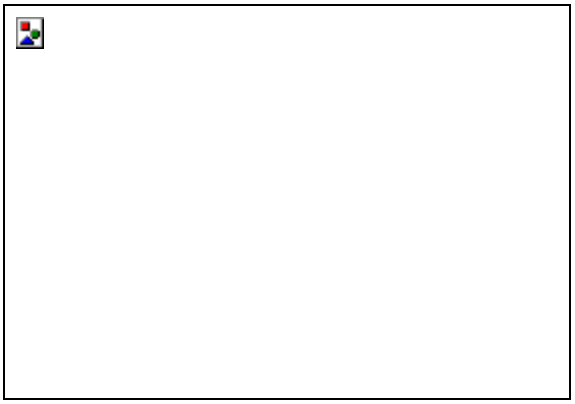
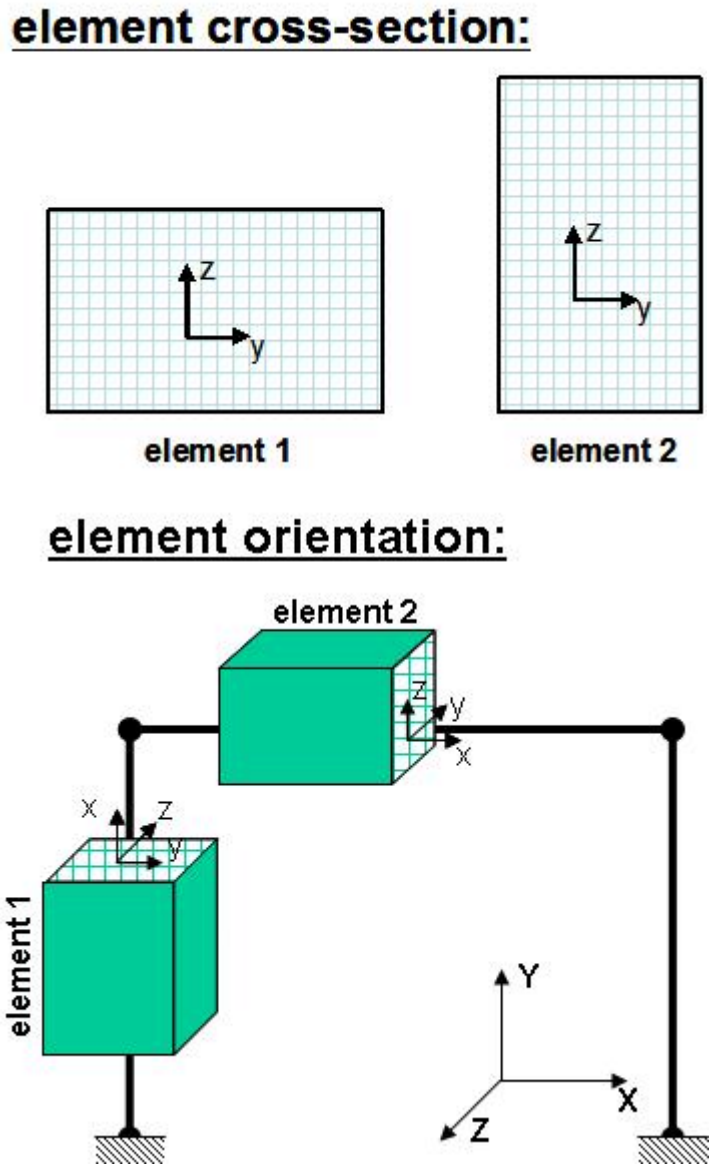
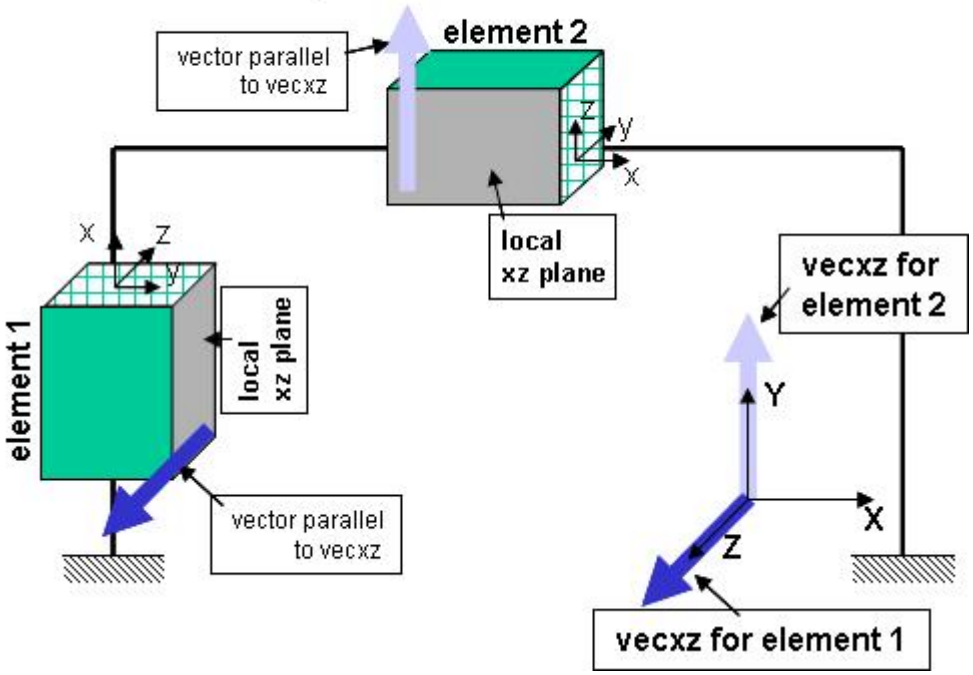


Figure 63: Definition of Rigid Joint Offset (note: check sign of dXi,etc components)

The following figures should aid in understanding the vector vecxz definition:



element xz plane and vectors:



linear transformation command:

element 1:
vecxz = z axis, coords: (0 0 1)
geomTransf Linear $transfTag 0 0 1
element 2:
vecxz = y axis, coords: (0 1 0)
geomTransf Linear $transfTag 0 1 0

P-Delta Transformation

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.

For a two-dimensional problem:

```
geomTransf PDelta $transfTag <-jntOffset $dXi $dYi $dXj $dYj>
```

For a three-dimensional problem:

```
geomTransf PDelta $transfTag $vecxzX $vecxzY $vecxzZ <-jntOffset $dXi $dYi
        $dZi $dXj $dYj $dZj>
```

The element coordinate system and joint offset values are specified as in the *Linear transformation* (page 200).

$transfTag	unique identifier for CrdTransf object
$vecxzX $vecxzY $vecxzZ	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. (These items need to be specified for the three-dimensional problem.) 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.
$dXi $dYi $dZi	joint offset values -- absolute 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* (page 26)) (optional)
$dXj $dYj $dZj	joint offset values -- absolute 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* (page 26)) (optional)

Corotational Transformation

This command is used to construct the Corotational Coordinate Transformation (CorotCrdTransf) object, which performs an exact geometric transformation of beam stiffness and resisting force from the basic system to the global coordinate system.

For a two-dimensional problem:

geomTransf Corotational $transfTag <-jntOffset $dXi $dYi $dXj $dYj>

For a three-dimensional problem:

geomTransf Corotational $transfTag $vecxzX $vecxzY $vecxzZ <-jntOffset $dXi $dYi $dZi $dXj $dYj $dZj>

NOTE: The Corotational transformation is only available with the Win32 version of *OpenSees* (http://opensees.berkeley.edu/OpenSees/binaries.html).

The element coordinate system and joint offset values are specified as in the *Linear transformation* (page 200).

$transfTag	unique identifier for CrdTransf object
$vecxzX $vecxzY $vecxzZ	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. (These items need to be specified for the three-dimensional problem.) 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.
$dXi $dYi $dZi	joint offset values -- absolute 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* (page 26)) (optional)
$dXj $dYj $dZj	joint offset values -- absolute 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* (page 26)) (optional)

CHAPTER 15

Time Series

While there is no timeSeries command in the language, a number of commands take as the argument a list of items which defines the TimeSeries object to be constructed as part of the command, such as the *LoadPattern* (page 214) and groundMotion (page 218) objects.

Time series act differently depending on what type of object they are applied to:

LoadPattern (page 214) **object:**

Load factors are applied to the loads and constraints

groundMotion (page 218) **object:**

Load factors are applied at the DOF in a ground motion

The type of TimeSeries objects available are presented in this chapter.

NOTE: The TimeSeries objects are handled by the Tcl interpreter as lists. Therefore, they can be defined a-priori within quotes "" and given a variable name. EXAMPLE:

set Gaccel "Series -dt $dt -filePath $outFile -factor $GMfatt"; # time series information

pattern UniformExcitation 2 1 -accel $Gaccel; # create uniform excitation

In This Chapter

Constant Time Series

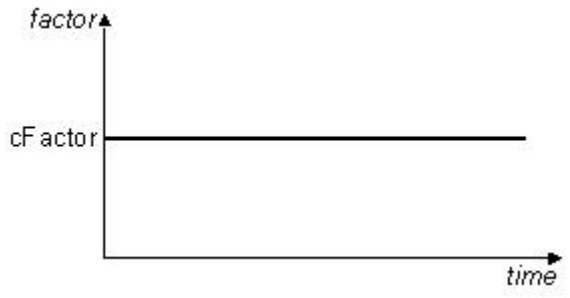
This command creates a ConstantSeries *TimeSeries* (page 208) object and associates it to the *LoadPattern* (page 214) object being constructed.

Constant <-factor $cFactor>

$cFactor load-factor coefficient. (optional. default = 1.0)

The load factor to be applied to the loads and constraints in the LoadPattern object is constant and equal to **$cFactor.**

Figure 64: Constant Time Series



Linear Time Series

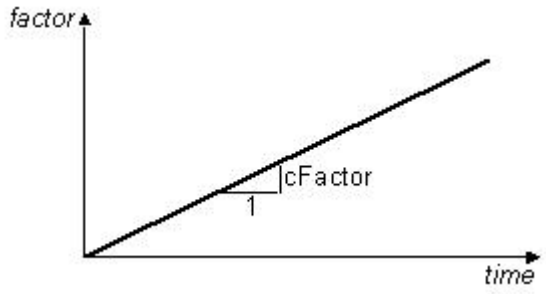
This command creates a LinearSeries *TimeSeries* (page 208) object and associates it to the *LoadPattern* (page 214) or *groundMotion* (page 218) object being constructed.

Linear <-factor $cFactor>

$cFactor load-factor coefficient. (optional. default = 1.0)

The load factor to be applied to the loads and constraints in the LoadPattern or groundMotion object is equal to **$cFactor* time**

Figure 65: Linear Time Series



Rectangular Time Series

This command creates a RectangularSeries *TimeSeries* (page 208) object and associates it to the *LoadPattern* (page 214) object being constructed.

Rectangular $tStart $tFinish <-factor $cFactor>

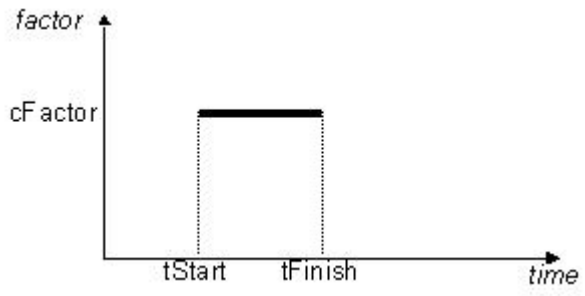
$tStart start time when the load factor is applied

$tFinish end time when the load factor is applied

$cFactor load-factor coefficient. (optional. default = 1.0)

The load factor to be applied to the loads and constraints in the LoadPattern object is constant and equal to **$cFactor** during the domain time from **$tStart** to **$tFinish**

Figure 66: Rectangular Time Series



Sine Time Series

This command creates a TrigSeries *TimeSeries* (page 208) object and associates it to the *LoadPattern* (page 214) object being constructed.

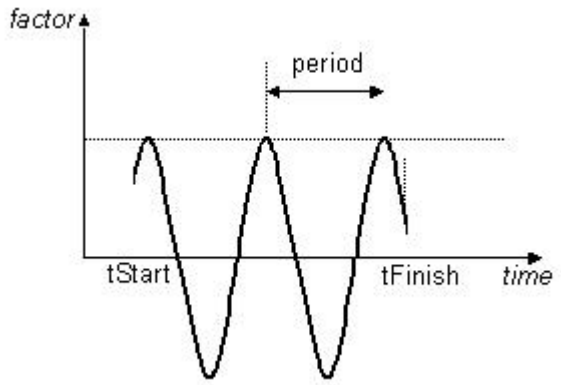
Sine $tStart $tFinish $period <-shift $shift> <-factor $cFactor>

$tStart	start time when the load factor is applied
$tFinish	end time when the load factor is applied
$period	characteristic period of sine wave
$shift	phase shift (radians) (optional. default = 0.0)
$cFactor	load-factor coefficient. (optional. default = 1.0)

The load factor applied to the loads and constraints in the LoadPattern object is equal to:

$$cFactor \cdot \sin\left[\frac{2 \cdot \pi \cdot (time - tStart)}{period} + shift\right]$$

Figure 67: Sine Time Series



Path Time Series

This command associates a *TimeSeries* (page 208) object of the type PathSeries or PathTimeSeries (if the increment is not constant) to a *LoadPattern* (page 214) object.

There are many ways to specify the load path.

For a load path where the values are specified at constant time intervals:

Series -dt $dt -values {list_of_values} <-factor $cFactor>

where the values are specified in a list included in the command

Series -dt $dt -filePath $fileName <-factor $cFactor>

where the values are taken from a file specified by **$fileName**

For a load path where the values are specified at non-constant time intervals:

Series -time {list_of_times} -values {list_of_values} <-factor $cFactor>

where both time and values are specified in a list included in the command

Series -fileTime $fileName1 -filePath $fileName2 <-factor $cFactor>

where both time and values are taken from a file specified by **$fileName1** (for the time data) and **$fileName2** (for the values data)

$cFactor load-factor coefficient. (optional. default = 1.0)

The load factor to be applied to the loads and constraints in the LoadPattern object is equal to **$cFactor*(user-defined series)**

CHAPTER 16

pattern Command

The pattern command is used to construct a LoadPattern object, its associated with the *TimeSeries* (page 208) object and the *Load* (page 215) and *Constraint* (page 232) objects for the pattern.

In This Chapter

plain Pattern

This command is used to construct an ordinary *LoadPattern* (page 214) object in the *Domain* (page 23).

```
pattern Plain $patternTag (TimeSeriesType arguments) {

        load (load-command arguments)

        sp (sp-command arguments)

        eleLoad (eleLoad-command arguments)

}
```

$patternTag	unique pattern object tag
TimeSeriesType arguments	list which is parsed to construct the *TimeSeries* (page 208) object associated with the LoadPattern object.
load ...	list of commands to construct nodal loads -- the *NodalLoad* (page 215) object
sp ...	list of commands to construct single-point constraints -- the *SP_Constraint* (page 216) object
eleLoad ...	list of commands to construct element loads -- the *eleLoad* (page 216) object

NOTE: The TimeSeries object is handled by the Tcl interpreter as a list and can be defined a-priori and given a variable name.

EXAMPLE

```
pattern Plain 1 Linear  {  ;          # define LoadPattern 1. impose load in a linear manner
        load 3 100 0. 0. 0. 0. 20.;          # apply force and moment at node 3
}
```

load Command

This command is used to construct a NodalLoad object.

load $nodeTag (ndf $LoadValues)

The nodal load is added to the LoadPattern being defined in the enclosing scope of the pattern command.

$nodeTag	node on which loads act
$LoadValues	load values that are to be applied to the node. Valid range is from 1 through *ndf* (page 26), the number of nodal degrees-of-freedom.

EXAMPLE

```
load 3 100 0. 0. 0. 0. 20.;      # apply force Fx=100 and moment Mz=20 at node 3
```

sp Command

This command is used to construct a single-point non-homogeneous constraint (SP_Constraint) object.

sp $nodeTag $DOFtag $DOFvalue

$nodeTag	node on which the single-point constraint acts
$DOFtag	degree-of-freedom at the node being constrained. Valid range is from 1 through *ndf* (page 26), the number of nodal degrees-of-freedom.
$DOFvalue	reference value of the constraint to be applied to the DOF at the node.

EXAMPLE

sp 3 1 0.1; # impose displacement Dx=0.1 at node 3

eleLoad Command

i NEED information from Michael!!

UniformExcitation Pattern

This command is used to construct a UniformExcitation load pattern object.

pattern UniformExcitation $patternTag $dir -accel (TimeSeriesType arguments) <-vel0 $ver0>

$patternTag	unique pattern object tag
$dir	direction of excitation (1, 2, or 3) used in formulating the inertial loads for the transient analysis.

TimeSeriesType arguments	*TimeSeries* (page 208) object associated with the acceleration record used in determining the inertial loads.
$vel0	initial velocity to be assigned to each node (optional, default: zero)

NOTE: The TimeSeries object is handled by the Tcl interpreter as a list and can be defined a-priori and given a variable name.

EXAMPLE:

set Gaccel "Series -dt $dt -filePath $outFile -factor $GMfatt"; # time series information

pattern UniformExcitation 2 1 -accel $Gaccel; # create uniform excitation with IDtag 2 in direction 1

MultipleSupport Pattern

This command is used to construct a MultipleSupportExcitation load pattern object.

```
pattern MultipleSupport $patternTag {

        groundMotion (groundMotion-command arguments)

        imposedMotion (imposedMotion-command arguments)

}
```

$patternTag	unique pattern object tag
groundMotion ...	list of commands to construct the *GroundMotions* (page 218) object that is then added to the object to define the multiple-support excitation that is being imposed on the model
imposedMotion ...	list of commands to construct the *ImposedSupportSP* (page 219) constraint object that is then added to the object to define the multiple-support excitation that is being imposed on the model

groundMotion Command

The groundMotion command is used to construct a GroundMotion object used by the *ImposedMotionSP* (page 219) constraints in a *MultipleSupportExcitation* (page 217) object.

Plain GroundMotion

This command is used to construct a plain GroundMotion object. Each GroundMotion object is associated with a number of *TimeSeries* (page 208) objects, which define the acceleration, velocity and displacement records.

groundMotion $gMotionTag Plain <-accel (accelSeriesType accelArgs)> <-vel (velSeriesType velArgs)> <-disp (dispSeriesType dispArgs)> <-int (IntegratorType intArgs)>

$gMotionTag unique *GroundMotion* (page 218) object tag

<-accel (accelSeriesType accelArgs)>

TimeSeries (page 208) objects defining the acceleration record (optional).

<-vel (velSeriesType velArgs)>

TimeSeries (page 208) objects defining the velocity record (optional)

<-disp (dispSeriesType dispArgs)>

TimeSeries (page 208) objects defining the displacement record (optional)

<-int (IntegratorType intArgs)>

If only the acceleration record is specified, the user has the option of specifying the *TimeSeriesIntegrator* (page 249) that is to be used to integrate the acceleration record to determine the velocity and displacement record (optional, default: Trapezoidal)

NOTE: The TimeSeries object is handled by the Tcl interpreter as a list and can be defined a-priori and given a variable name.

NOTE: Any combination of the acceleration, velocity and displacement time-series can be specified.

Interpolated GroundMotion

This command is used to construct an InterpolatedGroundMotion object.

groundMotion $gMotionTag Interpolated $gmTag1 $gmTag2 ... -fact $fact1 $fact2 ...

$gMotionTag	unique *GroundMotion* (page 218) object tag
$gmTag1 $gmTag2 ...	ground motions which have already been added to the *MultipleSupportExcitation* (page 217) object.
$fact1 $fact2 ...	factors that are used in the interpolation of the ground motions to determine the ground motion for this InterpolatedGroundMotion object.

imposedMotion Command

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* (page 218) object associated with the constraint.

imposedMotion $nodeTag $dirn $gMotionTag

$nodeTag	node where response is enforced
$dirn	dof of enforced response Valid range is from 1 through *ndf* (page 26), the number of nodal degrees-of-freedom.
$gMotionTag	pre-defined *GroundMotion* (page 218) object tag

NOTE: The *GroundMotion* (page 218) object must be added to the *MultipeSupportExcitation* (page 217) pattern before the ImposedMotionSP constraint.

FMK: ADD TIME-SERIES INTEGRATORS

CHAPTER 17

Recorder Objects

The recorder commands are used to construct a Recorder object, which is used to monitor items of interest to the analyst at each commit().

In This Chapter

Node Recorder

The Node type records the displacement, velocity, acceleration and incremental displacement at the nodes (translational & rotational)

recorder Node <-file $fileName> <-time> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-region $RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType

$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)
-time	this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)
$node1 $node2 ...	tags nodes where response is being recorded -- select nodes in domain (optional, default: all)
$startNode $endNode	tag for start and end nodes where response is being recorded -- range of nodes in domain (optional, default: all)

$RegionTag	tag for previously-defined selection of nodes defined using the Region command. (optional)
all	where response is being recorded -- all nodes in domain (optional & default)
$dof1 $dof2 ...	degrees of freedom of response being recorded. Valid range is from 1 through *ndf* (page 26) , the number of nodal degrees-of-freedom.
$respType	defines response type to be recorded. The following response types are available:

disp	displacement
vel	velocity
accel	acceleration
incrDisp	incremental displacement
eigen	eigenvector

NOTE: $respType must be the last argument in this command.

Example:

recorder Node -file node.out –time –node 1 5 -nodeRange 10 25 -dof 2 disp

recorder Node -file node34.eig -time -node 3 4 -dof 1 2 3 "eigen 2"

EnvelopeNode Recorder

The Node type records the envelope of displacement, velocity, acceleration and incremental displacement at the nodes (translational & rotational). The envelope consists of the following: minimum, maximum and maximum absolute value of specified response type.

recorder EnvelopeNode <-file $fileName> <-time> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-region $RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType

$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)
-time	this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)

$node1 $node2 ...	where response is being recorded -- select nodes (optional, default: all)
$startNode $endNode	tag for start and end where response is being recorded (optional, default: all)
$RegionTag	tag for previously-defined selection of nodes defined using the Region command. (optional)
all	where response is being recorded -- all nodes in domain (optional & default)
$dof1 $dof2 ...	degrees of freedom of response being recorded. Valid range is from 1 through *ndf* (page 26) , the number of nodal degrees-of-freedom.
$respType	defines response type to be recorded. The following response types are available:

disp	displacement
vel	velocity
accel	acceleration
incrDisp	incremental displacement

NOTE: $respType must be the last argument in this command.

Example:

recorder EnvelopeNode -file EnvelopeNode.out –time –node 1 5 -nodeRange 10 25 -dof 2 disp

Drift Recorder

The Drift type records the displacement drift between two nodes. The drift is taken as the ratio between the prescribed relative displacement and the specified distance between the nodes.

recorder Drift $fileName <-time> $node1 $node2 $dof $perpDirn

$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain
-time	this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)
$node1 $node2	the two for which drift is recorded

$dof	nodal degree of freedom to monitor for drift Valid range is from 1 through , the number of nodal degrees-of-freedom. ??? does rotation count???
$perpDirn	perpendicular global direction from which length is determined to compute drift (1 = X, 2 = Y, 3 = Z) ???????

Example:

recorder Drift drift.out –time 2 4 1 2

Element Recorder

The Element type records the response of a number of elements. The response recorded is element-dependent and depends on the arguments which are passed to the setResponse() element method.

recorder Element <-file $fileName> <-time> <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-region $regTag> <-ele all> ($arg1 $arg2 ...)

$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)
-time	this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)
$ele1 $ele2 ...	tags of elements whose response is being recorded -- selected elements in domain (optional, default: omitted)
$startEle $endEle	tag for start and end elements whose response is being recorded -- range of selected elements in domain (optional, default: all)
$regTag	previously-defined tag of region of elements whose response is being recorded -- region of elements in domain (optional)
all	elements whose response is being recorded -- all elements in domain (optional & default)
$arg1 $arg2 ...	arguments which are passed to the setResponse() element method

The setResponse() element method is dependent on the element type, and is described with the *element Command* (page 146).

- **Beam-Column Elements *(page 150, page 151, page 148, page 149) :***

Common to all beam-column elements:

globalForce – element resisting force in global coordinates (does not include inertial forces)

example:

recorder Element -file ele1global.out -time -ele 1 globalForce

localForce – element resisting force in local coordinates (does not include inertial forces)

example:

recorder Element -file ele1local.out -time -ele 1 localForce

- **Sections: *(page 129)***

section $secNum – request response quantities from a specific section along the element length,

$secNum refers to the integration point whose data is to be output

force – section forces

example: recorder Element -file ele1sec1Force.out –time -ele 1 section 1 force

deformation – section deformations

example: recorder Element -file ele1sec1Force.out –time -ele 1 section 1 deformation

stiffness – section stiffness

example: recorder Element -file ele1sec1Force.out –time -ele 1 section 1 stiffness

stressStrain – record stress-strain response.

example: recorder Element -file ele1sec1Force.out –time -ele 1 section 1 fiber $y $z stressStrain

$y local y coordinate of fiber to be monitored*

$z local z coordinate of fiber to be monitored*

***NOTE:** The recorder object will search for the fiber closest to the location ($y,$z) on the section and record its stress-strain response

EnvelopeElement Recorder

The Element type records the response of a number of elements. The response recorded is element-dependent and depends on the arguments which are passed to the setResponse() element method. The envelope consists of the following: **minimum**, **maximum** and **maximum absolute value** of specified response type.

recorder EnvelopeElement <-file $fileName> <-time> <-ele ($ele1 $ele2 ...)> <-eleRange $startele $endele> <-ele all> <-region $regTag> ($arg1 $arg2 ...)

$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)
-time	this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)
$eleID1 $eleID2 ...	tags of elements whose response is being recorded -- selected elements in *domain* (page 23) (optional, default: omitted)
$startele $endele	tag for start and end elements whose response is being recorded -- selected elements in *domain* (page 23) (optional, default: all)
all	elements whose response is being recorded -- all elements in *domain* (page 23) (optional & default)
$regTag	tag of region of elements whose response is being recorded -- region of elements in *domain* (page 23) (optional)
$arg1 $arg2 ...	arguments which are passed to the setResponse() element method

The setResponse() element method is dependent on the element type, and is described with the *element Command* (page 146).

- **Beam-Column Elements *(page 150, page 151, page 148, page 149) :***

Common to all beam-column elements:

globalForce – element resisting force in global coordinates (does not include inertial forces)

example:

recorder EnvelopeElement -file ele1global.out -time -ele 1 globalForce

localForce – element resisting force in local coordinates (does not include inertial forces)

example:

recorder EnvelopeElement -file ele1local.out -time -ele 1 localForce

- **Sections:** ***(page 129)***

section $secNum – request response quantities from a specific section along the element length,

$secNum refers to the integration point whose data is to be output

force – section forces

example: recorder EnvelopeElement -file ele1sec1Force.out –time -ele 1 section 1 force

deformation – section deformations

example: recorder EnvelopeElement -file ele1sec1Force.out –time -ele 1 section 1 deformation

stiffness – section stiffness

example: recorder EnvelopeElement -file ele1sec1Force.out –time -ele 1 section 1 stiffness

stressStrain – record stress-strain response.

example: recorder EnvelopeElement -file ele1sec1Force.out –time -ele 1 section 1 fiber $y $z stressStrain

$y	local y coordinate of fiber to be monitored*
$z	local z coordinate of fiber to be monitored*

***NOTE:** The recorder object will search for the fiber closest to the location ($y,$z) on the section and record its stress-strain response

Display Recorder

This recorder opens a graphical window for displaying of graphical information.

recorder display $windowTitle $xLoc $yLoc $xPixels $yPixels <-file $fileName>

$windowTitle	title of graphical window
$xLoc $yLoc	horizontal and vertical location of graphical window (upper left-most corner)
$xPixels $yPixels	width and height of graphical window in pixels
$fileName	in addition to the window display, information is sent to a file to redisplay images at a later time. (optional)

A TclFeViewer object is constructed. This constructor adds a number of additional commands to OpenSees, similar to the construction of the *BasicBuilder* (page 26). These additional commands are used to define the viewing system for the image that is place on the screen. These commands are currently under review and will be presented in a future version of this document.

Plot Recorder

This recorder type opens a graphical window for the plotting of the contents of the prescribed file

```
recorder plot $fileName $windowTitle $xLoc $yLoc $xPixels $yPixels -columns
        $xCol0 $yCol0 <-columns $xCol1 $yCol1><-columns $xCol2 $yCol2>
        ...
```

$fileName	source file of plotted data
$windowTitle	title of graphical window
$xLoc $yLoc	horizontal and vertical location of graphical window in pixels (upper left-most corner)
$xPixels $yPixels	width and height of graphical window in pixels
$xCol0 $yCol0	Column number to be plotted in X-axis and Y-axis, respectively. One set of columns must be defined.
$xCol1 $yCol1 **$xCol2 $yCol2**	Additional lines may be plotted on the same graph by repeating the -columns command. These data come from the same source file. (optional)

playback Command

This command is used to invoke playback on all Recorder objects constructed with the *recorder command* (page 221).

```
playback $commitTag
```

$commitTag	integer used to invoke the record() method (????)

CHAPTER 18

Analysis Objects

The Analysis objects are responsible for performing the analysis. The analysis moves the model along from state at time t to state at time t + dt. This may vary from a simple *static* (page 256) linear analysis to a *transient* (page 257, page 258) non-linear analysis. In OpenSees each Analysis object is composed of several component objects, which define the type of analysis how the analysis is performed.

In general terms, the analysis objects are used to solve the following time-dependent equilibrium-equation problem for a transient analysis:

$$M\ddot{U} + C\dot{U} + K(U) = P(t)$$ -- transient equilibrium

Where M is the acceleration-dependent inertial force vector, C is the velocity (damping) and displacement-dependent (stiffness) resisting-force vector. $P(t)$ is the external applied-force vector. The acceleration, velocity and displacement vectors are all time-dependent.

The component classes consist of the following:

- **ConstraintHandler (page 232) --** determines how the constraint equations are enforced in the analysis -- how it handles the boundary conditions/imposed displacements
- **DOF_Numberer (page 237) --** determines the mapping between equation numbers and degrees-of-freedom
- **AnalysisModel (page 256) --** defines what time of analysis is to be performed
- **Integrator --** determines the predictive step for time t+dt
- **SolutionAlgorithm (page 245) --** determines the sequence of steps taken to solve the non-linear equation at the current time step
- **SystemOfEqn/Solver --** within the solution algorithm, it specifies how to store and solve the system of equations in the analysis

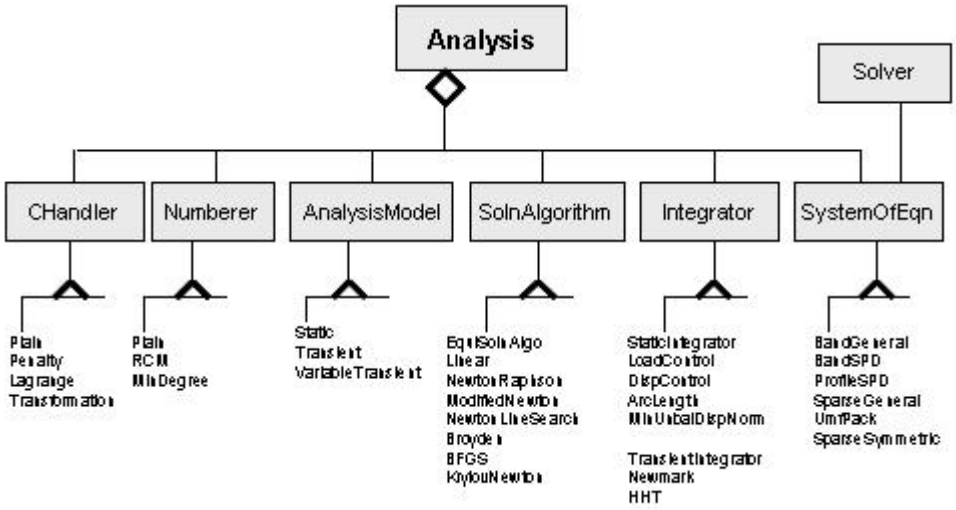

Analysis
Solver
CHandler
Numberer
AnalysisModel
SolnAlgorithm
Integrator
SystemOfEqn
Plain
Penalty
Lagrange
Transformation
Plain
RCM
MinDegree
Static
Transient
VariableTransient
EquiSolnAlgo
Linear
NewtonRaphson
ModifiedNewton
NewtonLineSearch
Broyden
BFGS
KrylovNewton
StaticIntegrator
LoadControl
DispControl
ArcLength
MinUnbalDispNorm
TransientIntegrator
Newmark
HHT
BandGeneral
BandSPD
ProfileSPD
SparseGeneral
UmfPack
SparseSymmetric

Figure 68: Analysis Object

In This Chapter

CHAPTER 19

constraints Command

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. The degrees of freedom can be broken down into U_R, the retained DOF's, and U_C, the condensed DOF's:

$$U = \begin{pmatrix} U_R \\ U_C \end{pmatrix}$$

- The **Plain Constraints** (page 234) command is used to enforce homogeneous single-point constraints, such as the case of homogeneous boundary conditions, where all boundary conditions are fixity, using *single-point constraints* (page 30, http://www.). For this case: $U_c=0.0$

The other constraints commands are used for ALL other cases, such as the case of non-homogeneous single-point constraints using the *sp command* (page 216), *multi-point constraints*, *imposed motions* (page 219) and *multi-support excitation* (page 217). For such cases, the relationship between DOF's can be written as:$U_C=C_{RC}U_R$. Where C_{RC} is a matrix of constants.

The following constraints handlers are currently availble:

- **Penalty Method** (page 234) -- consists of adding large numbers to the stiffness matrix and the restoring-force vectors to impose a prescribed zero or nonzero DOF. This method applies very stiff elements (numerically) at the boundary conditions. These additional stiffnesses affect the eigenvalues/eigenvectors in a transient analysis. This is the recommended method for a static analysis.
- **Lagrange Multipliers** (page 235) -- apply the method of lagrange multipliers to the system of equations, thus enlarging the size of the materices. Once Labrange Multipliers have been applied, the resulting stiffness matrix is no longer positive definite. Therefore, this method should be used only if there are condition-number problems with the penalty method.
- **Transformation Method** (page 236) -- transforms the stiffness matrix by condensing out the constrained DOF's. This method reduces the size of the system for multi-point constraints. This is the recommended method for a transient analysis. However, this method should not be used when nodes are constrained in series. For example, U_3 is constrained to U_2, which is then constrained to U_1.

Of the different methods, "the Lagrange multiplier method is more attractive than the transformation method if there are few constraint equations that couple many DOF. However, Lagrange multipliers are active at the structure level, but transformation equations can be applied at either the structure level or element by element. The latter has the appeal of disposing of constraints at an early stage, when the matrices are small and manageable". (Cook)

"In comparison with Lagrange multipliers, penalty functions have the advantage of introducing no new variables. However, the penalty matrix may significantly increase the bandwidth of the structural equations, depending on how DOF are numbered and what DOF are coupled by the constraint equations. Penalty functions have the disadvantage that penalty numbers must be chosen in an allowable range: large enough to be effective but not so large as to provoke numerical difficulties". (Cook)

More information and examples on these methods are discussed in detail in the Cook book.

In This Chapter

Plain Constraints

This command creates a PlainHandler which is only capable of enforcing homogeneous single-point constraints. If other types of constraints exist in the domain, a different constraint handler must be specified.

constraints Plain

Penalty Method

This command is used to construct a PenaltyConstraintHandler which will cause the constraints to be enforced using a penalty method. The penalty method consists of adding large numbers to the stiffness matrix and the restoring-force vectors to impose a prescribed zero or nonzero DOF.

constraints Penalty $alphaSP $alphaMP

$alphaSP	factor used when adding the single-point constraint into the system of equations
$alphaMP	factor used when adding the multi-point constraint into the system of equations

In this method the potential-energy equation which makes up the system of equations is augmented by a penalty function $\{\mathbf{t}\}^T[alpha]\{\mathbf{t}\}/2$

where [alpha] is a diagonal matrix of "penalty numbers". The resulting system of equations is of the form:

$[K + C^T alpha C]U = [R + C^T alpha Q]$

Where C^TalphaC can be called the penalty matrix. C and Q are matrices containing constants, K is the stiffness matrix, U represents the DOF and R the restoring forces. If alpha=0 the constraints are ignored. As alpha grows, U changes in such a way that the constraint equations are more nearly satisfied. In this case, however, the analysis becomes error prone, as the system represents a stiff region supported by a flexible region.

NOTE: The Penalty Method affects the maximum eigenvalues of the system and may cause problems in a Transient analysis.

The Penalty Method is discussed in detail in the Cook Book -- Concepts and Applications of Finite Element Analysis.

"Guideline for choice of alpha: If computer words carry approximately p decimal digits, experience has shown that alpha should not exceed $10^{p/2}$". (Cook)

Lagrange Multipliers

This command is used to construct a LagrangeConstraintHandler which will cause the constraints to be enforced using the method of Lagrange multipliers.

constraints Lagrange <$alphaSP> <$alphaMP>

$alphaSP	factor used when adding the single-point constraint into the system of equations (optional, default=1.0)
$alphaMP	factor used when adding the multi-point constraint into the system of equations (optional, default=1.0)

NOTE: Values for **$alphaSP** and **$alphaMP** other than 1.0 are permitted to offset numerical roundoff problems.

NOTE: The system of equations is not positive definite due to the introduction of zeroes on the diagonal by the constraint equations:

$$\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} U \\ \lambda \end{bmatrix} = \begin{bmatrix} R \\ Q \end{bmatrix}$$

From Cook: "Lagrange's method of undetermined multipliers is used to find the maximum or minimum of a function whose variables are not independent but have some prescribed relation. In structural mechanics the function is the potential energy and the variables are the DOF".

Transformation Method

This command is used to construct a TransofrmationConstraintHandler which will cause the constraints to be enforced using the transformation method.

constraints Transformation

NOTE: With the current implementation, a retained node in an MP_Constraint cannot also be specified as being a constrained node in another MP_Constraint.

The constraint equations takes the following form:

$$(T^T K T)\, U_r = T^T R$$

CHAPTER 20

numberer Command

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.

- **Plain (page 237) --** nodes are assigned degrees-of-freedom arbitrarily, based on the input file. This method is recommended for small problems or when sparse solvers are used, as they do their own internal DOF numbering.
- **RCM (page 238) --** nodes are assigned degrees-of-freedom using the Reverse Cuthill-McKee algorithm. This algorithm optimizes node numbering to reduce bandwidth using a numbering graph. This method will output a warning when the structure is disconnected.

As certain system of equation and solver objects do their own mapping, i.e. SuperLU, UmfPack, Kincho's, specifying a numberer other than plain may not be needed.

In This Chapter

Plain Numberer

This command is used to construct a PlainNumberer object.

numberer Plain

The Plain numberer assigns degrees-of-freedom to the nodes based on how the nodes are stored in the domain. Currently, the user has no control over how nodes are stored.

RCM Numberer

This command is used to construct a RCMNumberer object.

numberer RCM

The RCM numberer uses the reverse Cuthill McKee (REF?) algorithm to number the degrees of freedom.

CHAPTER 21

system Command

This command is used to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.

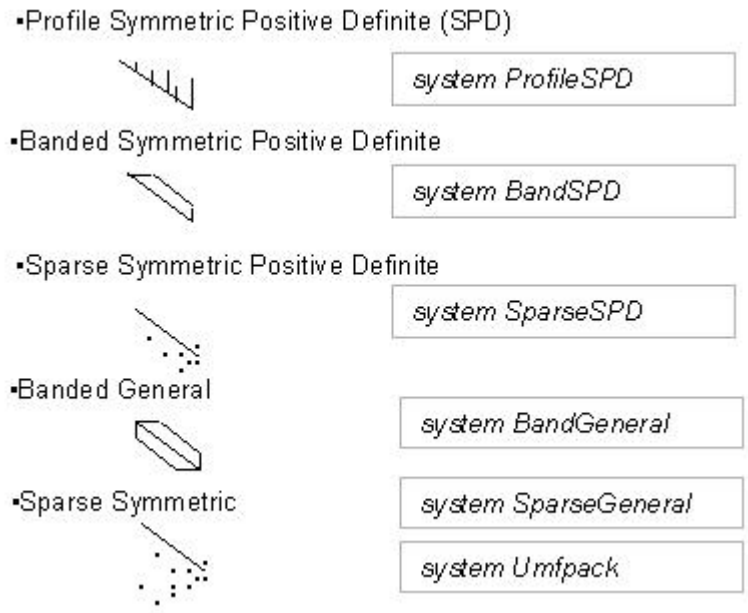


Figure 69: system command

In This Chapter

BandGeneral SOE

This command is used to construct an un-symmetric banded system of equations object which will be factored and solved during the analysis using the Lapack band general solver.

system BandGeneral

BandSPD SOE

This command is used to construct a symmetric positive definite banded system of equations object which will be factored and solved during the analysis using the Lapack band spd solver.

system BandSPD

ProfileSPD SOE

This command is used to construct symmetric positive definite profile system of equations object which will be factored and solved during the analysis using a profile solver.

system ProfileSPD

SparseGeneral SOE

This command is used to construct a general sparse system of equations object which will be factored and solved during the analysis using the SuperLU solver.

system SparseGeneral <-piv>

-piv	perform partial-pivoting (optional, Default: no partial pivoting is performed)

UmfPack SOE

This command is used to construct a general sparse system of equations object which will be factored and solved during the analysis using the UMFPACK solver. (REF?)

```
system UmfPack
```

SparseSPD SOE

This command is used to construct a sparse symmetric positive definite system of equations object which will be factored and solved during the analysis using a sparse solver developed at Stanford University by Kincho Law. (REF?)

```
system SparseSPD
```

CHAPTER 22

test Command

This command is used to construct a ConvergenceTest object. Certain *SolutionAlgorithm* **(page 245)** objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step. The convergence test is applied to the following equation:

$$K\Delta U = R$$

The test perform the following checks:

- **Norm Unbalance** $\sqrt{R \cdot R} < tol$
- **Norm Displacement Increment** $\sqrt{\Delta U^T \Delta U} < tol$
- **Energy Increment** $\frac{1}{2}(\Delta U^T R) < tol$

In This Chapter

Norm Unbalance Test

This command is used to construct a CTestNormUnbalance object which tests positive force convergence if the 2-norm of the b vector (the unbalance) in the *LinearSOE* (page 239) object is less than the specified tolerance.

test NormUnbalance $tol $maxNumIter <$printFlag>

$tol	convergence tolerance
$maxNumIter	maximum number of iterations that will be performed before "failure to converge" is returned
$printFlag	flag used to print information on convergence (optional)

- **0** no print output (default)
- **1** print information on each step
- **2** print information when convergence has been achieved
- **4** print norm, dU and dR vectors
- **5** at convergence failure, carry on, print error message, but do not stop analysis

The test performs the following check:

$\sqrt{R^R} < tol$

Norm Displacement Increment T

This command is used to construct a CTestNormDispIncr object which tests positive force convergence if the 2-norm of the x vector (the displacement increment) in the *LinearSOE* (page 239) object is less than the specified tolerance.

test NormDispIncr $tol $maxNumIter <$printFlag>

$tol convergence tolerance

$maxNumIter maximum number of iterations that will be performed before "failure to converge" is returned

$printFlag flag used to print information on convergence (optional)

- **0** no print output (default)
- **1** print information on each step
- **2** print information when convergence has been achieved
- **4** print norm, dU and dR vectors
- **5** at convergence failure, carry on, print error message, but do not stop analysis

The test performs the following check:

$\sqrt{\Delta U^T \Delta U} < tol$

Energy Increment Test

This command is used to construct a CTestEnergyIncr object which tests positive force convergence if one half of the inner-product of the x and b vectors (displacement increment and unbalance) in the *LinearSOE* (page 239) object is less than the specified tolerance.

test EnergyIncr $tol $maxNumIter <$printFlag>

$tol	convergence tolerance
$maxNumIter	maximum number of iterations that will be performed before "failure to converge" is returned
$printFlag	flag used to print information on convergence (optional) **0** no print output (default) **1** print information on each step **2** print information when convergence has been achieved **4** print norm, dU and dR vectors **5** at convergence failure, carry on, print error message, but do not stop analysis

The test performs the following check:

$\frac{1}{2}(\Delta U^T R) < tol$

CHAPTER 23

algorithm Command

This command is used to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.

In This Chapter

Linear Algorithm

This command is used to construct a Linear algorithm object which takes one iteration to solve the system of equations.

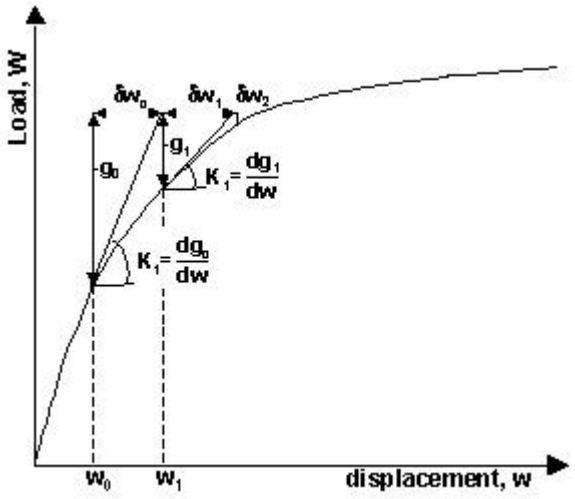
algorithm Linear

Newton Algorithm

This command is used to construct a NewtonRaphson algorithm object which uses the Newton-Raphson method to advance to the next time step.

algorithm Newton

NOTE: The tangent is updated at each iteration.



Newton with Line Search Algorithm

This command is used to construct a NewtonLineSearch algorithm object which uses the Newton-Raphson method with line search to advance to the next time step.

algorithm NewtonLineSearch $ratio

$ratio limiting ratio between the residuals before and after the incremental update (between 0.5 and 0.8)

If the ratio between the residuals before and after the incremental update is greater than the specified limiting ratio the line search algorithm developed by Crissfield (REF?) is employed to try to improve the convergence.

Modified Newton Algorithm

This command is used to construct a ModifiedNewton algorithm object which uses the Modified Newton-Raphson method to advance to the next time step. The difference between this method and the Newton-Raphson method is that the tangent stiffness is not updated at each step, thus avoiding expensive calculations needed in multi-DOF systems. However, more iterations may be needed to reach a prescribed accuracy.

algorithm ModifiedNewton

NOTE: The tangent at the first iteration of the current time step is used to iterate on the next time step.

Krylov-Newton Algorithm

This command is used to construct a KrylovNewton algorithm object which uses a modified Newton method with Krylov subspace acceleration to advance to the next time step.

algorithm KrylovNewton

The accelerator is described by Carlson and Miller in "Design and Application of a 1D GWMFE Code" from *SIAM Journal of Scientific Computing* (http://epubs.siam.org/sam-bin/dbq/toclist/SISC) (Vol. 19, No. 3, pp. 728-765, May 1998).

BFGS Algorithm

This command is used to construct a BFGS algorithm object for symmetric systems which performs successive rank-two updates of the tangent at the first iteration of the current time step.

algorithm BFGS <$count>

$count	number of iterations within a time step until a new tangent is formed

Broyden Algorithm

This command is used to construct a Broyden algorithm object for general unsymmetric systems which performs successive rank-one updates of the tangent at the first iteration of the current time step.

algorithm Broyden <$count>

$count	number of iterations within a time step until a new tangent is formed

CHAPTER 24

integrator Command

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

The system of nonlinear equations is of the form:

Static analysis: $R(U,l) = \lambda P^* - F_R(U)$

Transient analysis: $R(U,\dot{U},\ddot{U}) = P(t) - F_I(\ddot{U}) - F_R(U,\dot{U})$

The type of integrator used in the analysis is dependent on whether it is a **static analysis** (page 256) or **transient analysis** (page 257):

STATIC ANALYSIS*

- *LoadControl* (page 250) $\lambda_n = \lambda_{n-1} + d\lambda$
- *DisplacementControl* (page 251) $Uj_n = Uj_{n-1} + dUj$
- *MinUnbalDispNorm* (page 252) $d/d\lambda\,(dU_n{}^T dU_n)=0$
- *ArcLength* (page 252) $dU_n{}^T dU_n + \alpha^2\, d\lambda_n = ds^2$

TRANSIENT ANALYSIS

- *Newmark* (page 253)
- Hilbert-Hughes-Taylor Method (*HHT* (page 254))

***NOTE:** static integrators should only be used with a *Linear TimeSeries* (page 209) object with a factor of 1.0.

In This Chapter

Load Control

This command is used to construct a StaticIntegrator object of type LoadControl

integrator LoadControl $dLambda1 <$Jd $minLambda $maxLambda>

$dLambda1	first load increment (pseudo-time step) in the next invocation of *the analysis* (page 256) command.
$Jd	factor relating load increment at subsequent time steps. (optional, default: 1.0)
$minLambda $maxLambda	arguments used to bound the load increment (optional, default: $dLambda1 for both)

The load increment at iterations i, dLambda(i), is related to the load increment at (i-1), dLambda(i-1), and the number of iterations at (i-1), J(i-1), by the following:

dLambda(i) = dLambda(i-1)*Jd/J(i-1)

Displacement Control

This command is used to construct a StaticIntegrator object of the type DisplacementControl.

integrator DisplacementControl $nodeTag $dofTag $dU1 <$Jd $minDu $maxDu>

$nodeTag	node whose response controls the solution
$dofTag	degree-of-freedom whose response controls the solution. Valid range is from 1 through *ndf* (page 26), the number of nodal degrees-of-freedom.
$dU1	first displacement increment (pseudo-time step) in the next invocation of the analysis command
$Jd	factor relating displacement increment at subsequent time steps. (optional, default: 1.0)
$minDu $maxDu	arguments used to bound the displacement increment (optional, default: $dU1 for both)

The displacement increment at iterations i, dU(i), is related to the displacement increment at (i-1), dU(i-1), and the number of iterations at (i-1), J(i-1), by the following:

dU(i) = dU(i-1)*Jd/J(i-1)

$Uj_n = Uj_{n-1} + dUj$

Minimum Unbalanced Displacement Norm

This command is used to construct a StaticIntegrator object of type MinUnbalDispNorm.

integrator MinUnbalDispNorm $dlambda11 <$Jd $minLambda $maxLambda>

$dLambda11	first load increment (pseudo-time step) at the first iteration in the next invocation of the *analysis* (page 256) command.
$Jd	factor relating first load increment at subsequent time steps. (optional, default: 1.0)
$minLambda **$maxLambda**	arguments used to bound the load increment (optional, default: $dLambda11 for both)

The load increment at iterations i, dLambda1(i), is related to the load increment at (i-1), dLambda1(i-1), and the number of iterations at (i-1), J(i-1), by the following:

dLambda1(i) = dLambda1(i-1)*Jd/J(i-1)

Arc-Length Control

This command is used to construct a StaticIntegrator object of type ArcLength. Arc-length methods are used to enable the solution algorithm to pass limit points, such as maximum and minimum loads, and snap-through and snap-back responses. At these limit points, the stability of the numerical system is dependent on whether the analysis is performed under load or displacement control. In structural analysis, these limit points are characteristic of cracking of reinforced concrete and of buckling of shells.

integrator ArcLength $arclength $alpha

(??? is this the correct order, at the workshop Frank had it different)

$arclength	arclength value
$alpha	

SEE FMK

$$dU_n^T dU_n + \alpha^2 d\lambda_n = ds^2$$

The equilibrium equation can be written in the from:

$$g(p,\lambda) = q_i(p) - \lambda \cdot q_{ef} = 0$$

the arc-length method aims to find the intersection of the above equation with s=constant, where s is the arc-length, defined by:

$$s = \int 1 \, ds$$

and

$$ds = \sqrt{dp^T \cdot dp + d\lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef}}$$

the scaling parameter ψ is required because the load contribution depends on the adopted scaling between the load and displacement terms.

for the arc-length methods, one should replace the differential form of the equation for ds with an incremental form:

???

$$a = \left(\Delta p^T \cdot \Delta p + \Delta\lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef}\right) - \Delta l^2 = 0$$

$$a = \left(\Delta p^T \cdot \Delta p + \Delta\lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef}\right) - \Delta l^2 = 0$$

where Δl is the fixed 'radius of the desired intersection.

In the arc-length method the load parameter λ becomes a variable, adding one to the n displacement variables and equations.

Newmark Method

This command is used to construct a TransientIntegrator object of type Newmark.

integrator Newmark $gamma $beta <$alphaM $betaK $betaKinit $betaKcomm>

$gamma	Newmark parameter γ
$beta	Newmark parameter β
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero)

The damping matrix D is specified as a combination of stiffness and mass-proportional damping matrices:

D = $alphaM * M + $betaK * Kcurrent +$betaKinit * Kinit + $betaKcomm * KlastCommit

The mass and stiffness matrices are defined as:

M	mass matrix used to calculate Rayleigh Damping
Kcurrent	stiffness matrix at current state determination used to calculate Rayleigh Damping
Kinit	stiffness matrix at initial state determination used to calculate Rayleigh Damping
KlastCommit	stiffness matrix at last-committed state determination used to calculate Rayleigh Damping

Hilbert-Hughes-Taylor

This command is used to construct a TransientIntegrator object of type HHT or HHT1.

integrator HHT $gamma <$alphaM $betaK $betaKinit $betaKcomm>

$gamma	Newmark parameter γ
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero)

The damping matrix D is specified as a combination of stiffness and mass-proportional damping matrices:

D = $alphaM * M + $betaK * Kcurrent +$betaKinit * Kinit + $betaKcomm * KlastCommit

The mass and stiffness matrices are defined as:

M	mass matrix
Kcurrent	stiffness matrix at current state determination
Kinit	stiffness matrix at initial state determination
KlastCommit	stiffness matrix at last-committed state determination

CHAPTER 25

analysis Command

This command is used to construct the *Analysis object* (page 229), which defines what time of analysis is to be performed. The following analysis types are available:

- *Static Analysis* (page 256) -- solves the KU=R problem, without the mass or damping matrices.
- *Transient Analysis* (page 257) -- solves the time-dependent analysis. The time step in this type of analysis is constant. The time step in the output is also constant.
- *Variable Transient Analysis* (page 258) -- performs the same analysis type as the Transient Analysis object. The time step, however, is variable. This method is used when there are convergence problems with the Transient Analysis object at a peak or when the time step is too small. The time step in the output is also variable.

All currently-available analysis objects employ incremental solution strategies.

In This Chapter

Static Analysis

This command is used to construct a StaticAnalysis object.

analysis Static

This analysis object is constructed with the component objects previously created by the analyst. If none has been created, default objects are constructed and used:

Component Object	Default object

SolutionAlgorithm **(page 245)**, *StaticIntegrator* (page 251, page 249, page 250, page 252)	*NewtonRaphson* (page 245) EquiSolnAlgo with a *CTestNormUnbalance* (page 242) with a tolerance of 1e-6 and a maximum of 25 iterations
ConstraintHandler **(page 232)**	*PlainHandler* (page 234) ConstraintHandler
DOF_Numberer **(page 237)**	*RCM* **(page 238)** DOF_Numberer
LinearSOE (page 239), *LinearSolver* (page 239)	*profiled symmetric positive definite* (page 240) LinearSOE and Linear Solver
Integrator	*LoadControl StaticIntegrator* (page 250) with a constant load increment of 1.0

Transient Analysis

This command is used to construct a DirectIntegrationAnalysis object.

analysis Transient

This analysis object is constructed with the component objects previously created by the analyst. If none has been created, default objects are constructed and used:

Component Object	Default object
SolutionAlgorithm **(page 245)**, *TransientIntegrator* (page 252, page 254, page 249, page 253)	*NewtonRaphson* (page 245) EquiSolnAlgo with a *CTestNormUnbalance* (page 242) with a tolerance of 1e-6 and a maximum of 25 iterations
ConstraintHandler **(page 232)**	*PlainHandler* (page 234) ConstraintHandler
DOF_Numberer **(page 237)**	*RCM* **(page 238)** DOF_Numberer
LinearSOE (page 239), *LinearSolver* (page 239)	*profiled symmetric positive definite* (page 240) LinearSOE and Linear Solver

Integrator	*Newmark TransientIntegrator* (page 253) with γ=0.5 and β=0.25

VariableTransient Analysis

This command is used to construct a VariableTimeStepDirectIntegrationAnalysis object.

analysis VariableTransient

This analysis object is constructed with the component objects previously created by the analyst. If none has been created, default objects are constructed and used:

Component Object	Default object
SolutionAlgorithm **(page 245)**, *TransientIntegrator* (page 252, page 254, page 249, page 253)	*NewtonRaphson* (page 245) EquiSolnAlgo with a *CTestNormUnbalance* (page 242) with a tolerance of 1e-6 and a maximum of 25 iterations
ConstraintHandler **(page 232)**	*PlainHandler* (page 234) ConstraintHandler
DOF_Numberer **(page 237)**	*RCM* **(page 238)** DOF_Numberer
LinearSOE (page 239), *LinearSolver* (page 239)	*profiled symmetric positive definite* (page 240) LinearSOE and Linear Solver
Integrator	*Newmark TransientIntegrator* (page 253) with γ=0.5 and β=0.25

CHAPTER 26

rayleigh command

This command is used to assign damping to all previously-defined elements and nodes:

rayleigh $alphaM $betaK $betaKinit $betaKcomm

The damping matrix D is specified as a combination of stiffness and mass-proportional damping matrices:

D = $alphaM * M + $betaK * Kcurrent +$betaKinit * Kinit + $betaKcomm * KlastCommit

The mass and stiffness matrices are defined as:

M	mass matrix used to calculate Rayleigh Damping
Kcurrent	stiffness matrix at current state determination used to calculate Rayleigh Damping
Kinit	stiffness matrix at initial state determination used to calculate Rayleigh Damping
KlastCommit	stiffness matrix at last-committed state determination used to calculate Rayleigh Damping

CHAPTER 27

eigen Command

This command is used to perform a generalized eigenvalue problem to determine a specified number of eigenvalues and eigenvectors.

eigen <$type> $numEigenvalues

$type	eigen-value analysis type:	
	frequency	solve: K- λ M
	generalized	solve: K- λ M (default)
	standard	solve: K- λ I
$numEigenvalues	number of first eigenvalues (λ) to be determined	

The eigenvectors are stored at the nodes and can be printed out using *Node Recorder* (page 221) or the *Print command* (page 264).

CHAPTER 28

analyze Command

This command is invoked on the Analysis object constructed with the *analysis command* (page 256).

analyze $numIncr <$dt> <$dtMin $dtMax $Jd>

$numIncr		number of load steps
$dt		time-step increment. Required if a transient analysis (page 257) or variable time step transient analysis (page 258) was specified.
$dtMin	**$dtMax**	minimum and maximum time steps Required if a variable time step transient analysis (page 258) was specified.
$Jd		number of iterations performed at each step Required if a variable time step transient analysis (page 258) was specified.

This command RETURNS:

0	successful
<0	unsuccessful

CHAPTER 29

dataBase Commands

This command is used to construct a FE_Datastore object.

Currently there is only one type of Datastore object available.

In This Chapter

FileDatastore Command

This command is used to construct the FE_Datastore object.

database $type $dbName

$type	database type:	
	File	outputs database into a file
	MySQL	creates a SQL database
	BerkeleyDB	creates a BerkeleyDB database
$dbName	database name.	
	If the database type is **File**, the command will save the data into a number of files, e.g. $dbName.id11 for all ID objects of size 11 that sendSelf() is invoked upon.	

The invocation of this command will add the additional commands save and restore to the OpenSees interpreter to allow users to save and restore model states.

save Command

This command is used to save the state of the model in the database.

```
save $commitTag
```

$commitTag unique identifier that can be used to *restore* (page 263) the state at a later time

restore Command

This command is used to restore the state of the model from the information stored in the database.

```
restore $commitTag
```

$commitTag unique identifier used to restore the state at the model when the *save* (page 263) command was invoked

CHAPTER 30

Miscellaneous Commands

These are a few additional miscellaneous command used in OpenSees

In This Chapter

print Command

This command is used to print output.

To print all objects of the domain:

```
print <$fileName>
```

To print node information:

```
print <$fileName> -node <-flag $flag> <$node1 $node2 ...>
```

To print element information:

```
print <$fileName> -ele <-flag $flag> <$ele1 $ele2 ...>
```

$fileName fileName for printed output (optional, Default: stderr -- screen dump)

$flag integer flag to be sent to the print() method, depending on the node and element type (optional)

$node1 $node2 ...	node tag for selected-node output (optional)
	Default: all
$ele1 $ele2 ...	element tag for selected-element output (optional)
	Default: all

reset Command

This command is used to set the state of the domain to its original state.

```
reset
```

The command invokes revertToStart() on the *Domain* (page 23) object.

wipe Command

This command is used to destroy all constructed objects.

```
wipe
```

This command is used to start over without having to exist and restart the *interpreter* (page 17).

wipeAnalysis Command

This command is used to destroy all objects constructed for the analysis.

```
wipeAnalysis
```

This command is used to start a new type of analysis. This command does not *destroy* the elements (page 146), *nodes* (page 28), *materials* (page 108, page 35), etc. It does destroy the solution strategies: the *algorithm* (page 245), *analysis* (page 256), *equation solver* (page 239), *constraint handler* (page 232), etc.

loadConst Command

This command is used to invoke setLoadConst() on all *LoadPattern* (page 214) objects which have been created up to this point.

```
loadConst <-time $pseudoTime>
```

-time $pseudoTime set pseudo time in the domain to $pseudoTime (optional, default: zero)

getTime Command

This command returns the time in the domain.

```
getTime
```

nodeDisp Command

Returns the displacement or rotation at specified node.

```
nodeDisp $nodeTag $dof
```

$nodeTag node tag

$dof degree-of-freedom tag

Valid range is from 1 through *ndf* (page 26), the number of nodal degrees-of-freedom. (???)

video Command

This command is used to construct a TclVideoPlayer object for displaying the images in a file created by the recorder *display* (page 227) command.

```
video -file $fileName -window $windowName
```

$fileName	fileName of images file created by the recorder *display* (page 227) command
$windowName	name of window to be created

The images are displayed by invoking the *play* (page 267) command.

play Command

This command is used to play the TclVideoPlayer object created by the *video* (page 267) command.

```
play
```

CHAPTER 31

How To....

In this chapter, some examples on how to generate input for OpenSees for specific tasks will be presented.

In This Chapter

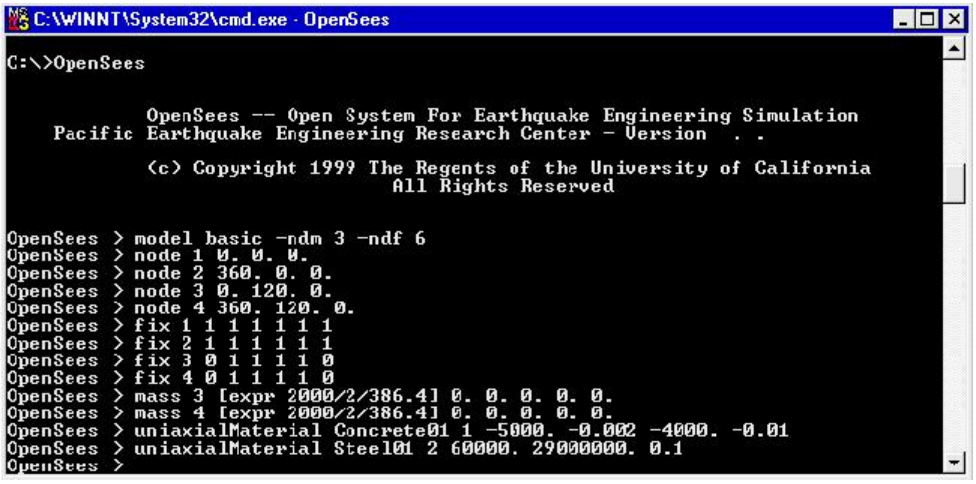
Run OpenSees

There are three ways that OpenSees/Tcl commands can be executed:

- ***Interactive***

Commands can be input directly at the propt, as shown in the figure (Win32 version):

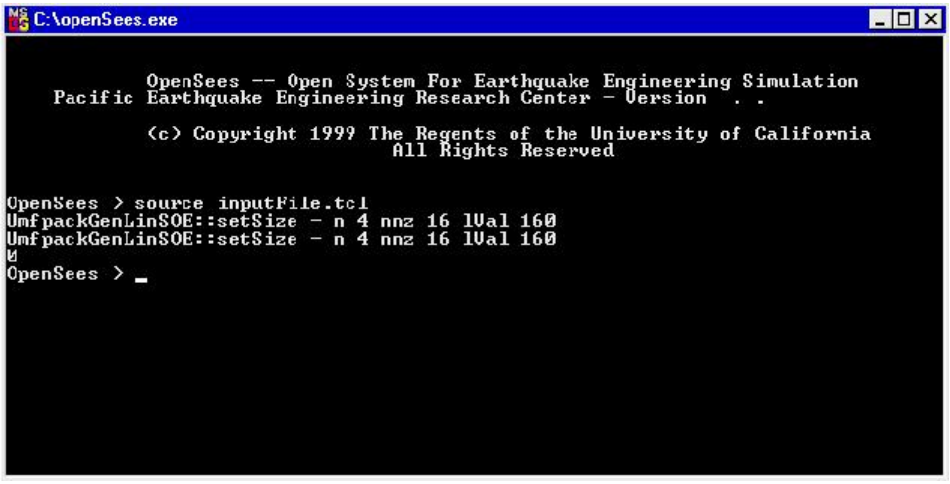
Figure 70: Run OpenSees -- Interactive



➢ *Execute Input File at OpenSees prompt*

This method is the most-commonly used one. An external file containing the input commands can be generated a-priori (inputFile.tcl) and be executed at the OpenSees prompt by using the source command. The generation of the input script files is presented in this chapter. The file execution is shown in the figure (Win32 version):

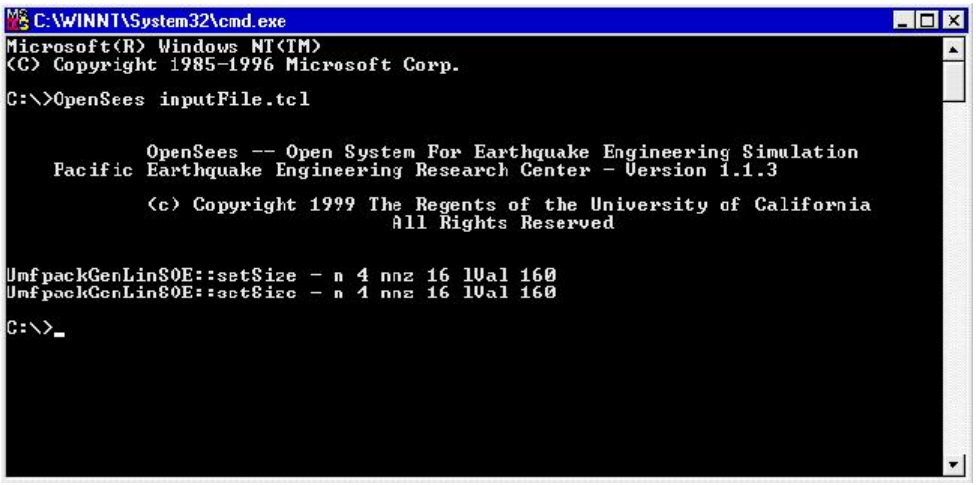
Figure 71: Run OpenSees -- Source File



➢ *Batch Mode*

The previously-created input file containing the Tcl script commands necessary to execute the analsis can also be executed at the MS-DOS/Unix prompt, as shown in the figure (Win32 version):

Figure 72: Run OpenSees -- Batch Mode



...Define Units & Constants

The OpenSees interpreter does not process units. Units, however, can be used when entering values if these units are defined previously. The unit definition consists of two parts: the basic units are defined first, all other units are subsequently defined. The basic units are assigned a value of one and all OpenSees output is in these units. It is very important that all basic units are independent of each other. The unit-definition file can contain both metric and Imperial units, as can the basic units. Hence, the input files may contain mixed units.

Constants, such as π and g can also be defined apriori.

An example of unit and constant definition is given in the following:

- ***Procedure to define units and constants***

```
# ----Units&Constants.tcl------------------------------------------------------
set in 1.;                          # define basic units
set sec 1.;
set kip 1.;
set ksi [expr $kip/pow($in,2)];             # define dependent units
set psi [expr $ksi/1000.];
set ft [expr 12.*$in];
set lb [expr $kip/1000];
set pcf  [expr $lb/pow($ft,3)];
set ksi  [expr $kip/pow($in,2)];
set psi  [expr $ksi/1000.];
set cm  [expr $in/2.54];            # define metric units
set meter [expr 100.*$cm];
set MPa [expr 145*$psi];
set PI [expr 2*asin(1.0)];          # define constants
set g [expr 32.2*$ft/pow($sec,2)];
set U     1.e10;            # a really large number
set u     [expr 1/$U];      # a really small number
```

CHAPTER

...Generate Matlab Commands

Matlab is a common tool for post-processing. Matlab command files can be generated using the Tcl scripting language. Using this technique ensures that the same analysis parameters are used.

Here is an example.

- ***# script to generate .m file to be read by matlab***

```
# -------------MatlabOutput.tcl-------------------------------------------------------------------
set Xframe 1;                # this parameter would be passed in
set fDir "Data/";
file mkdir $fDir;                    # create directory
set outFileID [open $fDir/DataFrame$Xframe.m w];  # Open output file for writing
puts $outFileID   "Xframe($Xframe) = $Xframe;";    # frame ID
puts $outFileID   "Hcol($Xframe) = $Hcol;";                # column diameter
puts $outFileID   "Lcol($Xframe) = $Lcol;";                # column length
puts $outFileID   "Lbeam($Xframe) = $Lbeam;";              # beam length
puts $outFileID    "Hbeam($Xframe) = $Hbeam;";   # beam depth
puts $outFileID    "Bbeam($Xframe) = $Bbeam;";   # beam width
puts $outFileID   "Weight($Xframe) = $Weight;"     ;          # superstructure weight
close $outFileID
```

...Define Tcl Procedure

The procedure is a useful tool available from the Tcl language. A procedure is a generalized funcion/subroutine using arguments. Whenever the Tcl procedure is invoked, the contents of body will be executed by the Tcl interpreter.

An example of a Tcl procedure is found in RCcircSec.tcl. It defines a procedure which generates a circular reinforced concrete section with one layer of steel evenly distributed around the perimeter and a confined core:

```
# ----RCcircSec.tcl-------------------------------------------------------------------
#                              by Michael H. Scott
# Define a procedure which generates a circular reinforced concrete section
```

```
# with one layer of steel evenly distributed around the perimeter and a confined core.
# Formal arguments
#    id - tag for the section that is generated by this procedure
#    ri - inner radius of the section
#    ro - overall (outer) radius of the section
#    cover - cover thickness
#    coreID - material tag for the core patch
#    coverID - material tag for the cover patches
#    steelID - material tag for the reinforcing steel
#    numBars - number of reinforcing bars around the section perimeter
#    barArea - cross-sectional area of each reinforcing bar
#    nfCoreR - number of radial divisions in the core (number of "rings")
#    nfCoreT - number of theta divisions in the core (number of "wedges")
#    nfCoverR - number of radial divisions in the cover
#    nfCoverT - number of theta divisions in the cover
# Notes
#    The center of the reinforcing bars are placed at the inner radius
#    The core concrete ends at the inner radius (same as reinforcing bars)
#    The reinforcing bars are all the same size
#    The center of the section is at (0,0) in the local axis system
#    Zero degrees is along section y-axis
proc RCcircSection {id ri ro cover coreID coverID steelID numBars barArea nfCoreR nfCoreT nfCoverR
nfCoverT} {
        section fiberSec $id {
                set rc [expr $ro-$cover];              # Core radius
                patch circ $coreID $nfCoreT $nfCoreR 0 0 $ri $rc 0 360;      # Core patch
                patch circ $coverID $nfCoverT $nfCoverR 0 0 $rc $ro 0 360;   # Cover patch
                if {$numBars <= 0} {
                        return
                }
                set theta [expr 360.0/$numBars];              # Angle between bars
                layer circ $steelID $numBars $barArea 0 0 $rc $theta 360;   # Reinforcing layer
        }
}
```

This procedure is invoked by the following commands, assuming that all arguments have been defined previously in the input generation:

```
source RCcircSection.tcl;

RCcircSection $IDcolFlex  $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol  $AbCol  $nfCoreR
$nfCoreT $nfCoverR $nfCoverT
```

NOTE: the file containing the definition of the procedure (RCcircSec.tcl) needs to be sourced before the procedure is invoked.

...Read External files

External files may either contain Tcl commands or data.

➢ *Common input file*

The external file may contain a series of commands that is common in most analyses. One set of Tcl commands that can be stored in a external file are ones which define units.

An example of an external file that may want to be read within the input commands is the unit-definition file presented earlier (*units&constants.tcl* (page 355)).

This file is invoked with the following command:

```
source units.tcl
```

➢ *Repeated Calculations*

An external file may contain a series of calculations that are repeated. An example of this is a parameter study:

```
set Hcolumn 66;
source analysis.tcl
set Hcolumn 78;
source analysis.tcl
```

The analysis.tcl file contains the commands that set up and execute the entire analysis.

➢ *External Data File*

The following commands open a data file (filename=inFilename), read the file row by row and assign the value of each row to the a single variable (Xvalues). If there are more than one value in the row, $Xvalues is a list array, and the individual components may be extracted using the **lindex** command. The user may change the commands to be exectued once the data-line has been read to whatever is needed in the analysis.

```
# ----ReadData.tcl-----------------------------------------------------------------
if [catch {open $inFilename r} inFileID] {;   # Open the input file and check for error
        puts stderr "Cannot open $inFilename for reading";          # output error statement
 } else {
        foreach line [split [read $inFileID] \n] { ;     # Look at each line in the file
                if {[llength $line] == 0} {;  # Blank line --> do nothing
                        continue;
                } else {
```

```
                    set Xvalues $line;          # execute operation on read data
                }
        }
        close $inFileID;  ;         # Close the input file
}
```

Building The Model

...Define Variables and Parameters

In the Tcl scripting language variables may be used to represent numbers. Once defined, these variables can be use instead of numbers in Tcl and OpenSees commands. When they are being recalled, the variables are precedented by the symbol $. If this symbol is not used, the variable name is interpreted as a string command and an error may result.

A few examples are given:

- ***MATERIAL PARAMETERS:***

```
# -----MaterialParameters.tcl------------------------------------------------------
set fc [expr -4.0*$ksi];            #  nominal compressive strength of concrete
set Ec [expr 57*$ksi*sqrt(-$fc/$psi)];      # Concrete Elastic Modulus
set fc1C [expr 1.26394*$fc];                # CONFINED concrete (mander model), max stress
set eps1C [expr 2.*$fc1C/$Ec];      # strain at maximum stress
set fc2C $fc;                       # ultimate stress
set eps2C [expr 2.*$fc2C/$Ec];      # strain at ultimate stress
set fc1U $fc;                       # UNCONFINED concrete (parabolic model), max stress
set eps1U -0.003;                   # strain at maximum stress
set fc2U [expr 0.1*$fc];            # ultimate stress
set eps2U -0.006;                   # strain at ultimate stress
set Fy [expr 70.*$ksi];             # STEEL yield stress
set Es [expr 29000.*$ksi];          # elastic modulus of steel
set epsY [expr $Fy/$Es];            # steel yield strain
set Fy1 [expr 95.*$ksi];            # steel stress post-yield
set epsY1 0.03;                     # steel strain post-yield
set Fu [expr 112.*$ksi];            # ultimate stress of steel
set epsU 0.08;              # ultimate strain of steel
set Bs [expr ($Fu-$Fy)/($epsU-$epsY)/$Es];   # post-yield stiffness ratio of steel
```

```
set pinchX      1.0;        # pinching parameter for hysteretic model
set pinchY      1.0;        # pinching parameter for hysteretic model
set damage1     0.0;        # damage parameter for hysteretic model
set damage2     0.0;        # damage parameter for hysteretic model
set betaMUsteel 0.0;        # degraded unloading stiffness for hysteretic material based on MU^(-beta)
set betaMUjoint 0.0;        # degraded unloading stiffness for hysteretic material based on MU^(-beta) -- timoshenko value of 0.5
set betaMUph    0.0;        # degraded unloading stiffness for hysteretic material based on MU^(-beta) -- timoshenko value of 0.5
set G    $U;            # Torsional stiffness Modulus
set J    1.;            # Torsional stiffness of section, place here just to keep with G
set GJ [expr $G*$J];    # Torsional stiffness
```

- ***ELEMENT PARAMETERS:***

```
# -----ElementParameters.tcl------------------------------------------------------
set Hcol        [expr 5.*$ft];      # column diameter
set Lcol        [expr 36*$ft];      # column length
set Hbeam       [expr 8.*$ft];      # beam depth
set Lbeam       [expr 36.*$ft];     # beam length
set GrhoCol     0.0125;  # column longitudinal-steel ratio
set Weight      [expr 2000.*$kip];          # superstructure weight
set Bbeam       $Hcol;   # beam width
set Rcol        [expr $Hcol/2];   # COLUMN radius
set Acol        [expr $Pl*pow($Rcol,2)]; # column cross-sectional area
set cover       [expr $Hcol/15];  # column cover width
set IgCol       [expr $Pl*pow($Rcol,4)/4];           # column gross moment of inertia, uncracked
set IyCol       $IgCol;  # elastic-column properties
set IzCol       $IgCol;  # elastic-column properties
set IzBeam      [expr $Bbeam*pow($Hbeam,3)/12]; # beam gross moment of inertia, about horizontal Z-axis
set IyBeam      [expr $Hbeam*pow($Bbeam,3)/12]; # beam gross moment of inertia, about the vertical Y-axis
set Abeam       [expr $Hbeam*$Bbeam*10000]; # beam cross-sectional area, make it very very stiff
# define COLUMN section parameters
set NbCol       20;         # number of column longitudinal-reinforcement bars
set AsCol       [expr $GrhoCol*$Acol];   # total steel area in column section
set AbCol       [expr $AsCol/$NbCol];    # bar area of column longitudinal reinforcement
set riCol       0.0;        # inner radius of column section
set roCol       $Rcol;   # outer radius of column section
set IDcore      1;          # ID tag for core concrete
set IDcover     2;          # ID tag for cover concrete
```

```
set IDsteel      3;      # ID tag for steel
set nfCoreR      8;      # number of radial fibers in core
set nfCoreT      16;     # number of tangential fibers in core
set nfCoverR     2;      # number of radial fibers in cover
set nfCoverT     16;     # number of tangential fibers in cover
set IDcolFlex    2;      # ID tag for column section in flexure, before aggragating torsion
set IDcolTors    10;     # ID tag for column section in torsion
set IDcolSec     1;      # ID tag for column section
set IDcolTrans   1;      # ID tag for column transformation, defining element normal
set IDbeamTrans  2;      # ID tag for beam transformation, defining element normal
set np   5;      # Number of integration points
```

- ***GRAVITY PARAMETERS:***

```
# -----GravityParameters.tcl----------------------------------------------------
# define GRAVITY paramters
set Pdl   [expr $Weight/2];                  # gravity axial load per column
set Wbeam      [expr $Weight/$Lbeam];        # gravity dead load distributed along beam length
set Mdl  [expr $Wbeam*pow($Lbeam,2)/12];     # nodal moment due to distributed dl
set Mass       [expr $Weight/$g];                  # mass of superstructure
set Mnode      [expr $Mass/2];               # nodal mass for each column joint
```

- ***ANALYSIS PARAMETERS:***

```
# -----AnalysisParameters.tcl---------------------------------------------------
set DxPush     [expr 0.1*$in];   # Displacement increment for pushover analysis
set DmaxPush   [expr 0.05*$Lcol];      # maximum displamcement for pushover analysis
set DtAnalysis   [expr 0.005*$sec];# time-step Dt for lateral analysis
set DtGround     [expr 0.02*$sec];# time-step Dt for input grond motion
set TmaxGround [expr 35 *$sec]; # maximum duration of ground-motion analysis
set gamma      0.5;              # gamma value for newmark integration
set beta       0.25              # beta value for newmark integration
```

...Build Model and Define Nodes

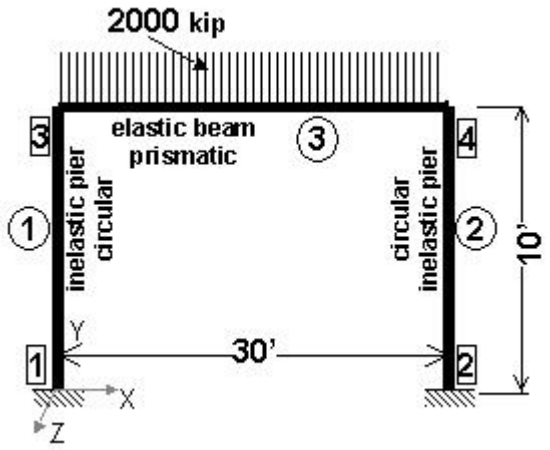
This example shows how to set up the geometry of the structure shown in the *Figure* (page 279).

These commands are typically placed at the beginning of the input file, after the header remarks.

```
# construct model builder using the model Command (page 26)
wipe;                    #  clear data from past analysis
model basic -ndm 3 -ndf 6;       # modelbuilder: basic (page 26), ndm= no. dimensions, ndf= no.
```

```
# Define nodes  ------ frame is in X-Y plane (X-horizontal, Y-vertical) using the node Command (page 28)
node  1    0.      0.     0.; # base of left column
node  2    360.    0.     0.;# base of right columnn
node  3    0.      120.   0.;# top of left column
node  4    360.    120.   0.;          # top of right columnn
# Define Boundary Conditions and nodal mass using the fix Command (page 30)  ! 1: restrained, 0: released
fix   1   1  1  1  1  1  1;
fix   2   1  1  1  1  1  1;
fix   3   0  1  1  1  1  0   -mass [expr 2000/2/386.4] 0.  0.  0.  0.  0.
fix   4   0  1  1  1  1  0
# define mass at node 4 using the mass Command (page 29):
mass 4  [expr 2000/2/32.2/12] 0.  0.  0.  0.  0.
```

NOTE: The second command assigning the mass of a specific node will override any previous mass assignments to that node.



...Build Model and Define Nodes using Variables

This example sets up the geometry of the cantilever column shown in the *Figure* (page 279) -- using variables.

```
# -------Build&Nodes.tcl------------------------------------------------------------
wipe;                          #  clear data from past analysis
model basic -ndm 3 -ndf 6;
# define units and constants
# source units.tcl;            # if contained in external file
# Define nodes  ------ frame is in X-Y plane (X-horizontal, Y-vertical) using the node Command (page 28)
node  1    0.            0.        0.;     # base of left column
```

```
node  2    $Lbeam    0.        0.;    # base of right columnn
node  3    0.            $Lcol    0.;    # top of left column
node  4    $Lbeam    $Lcol    0.;    # top of right columnn
 # Define Boundary conditions using the fix Command (page 30)  ! 1: restrained, 0: released
fix   1   1  1  1  1  1  1;       # fully-fixed support
fix   2   1  1  1  1  1  1;
fix   3   0  1  1  1  1  0    -mass $Mass 0.  0.  0.  0.  0.;
fix   4   0  1  1  1  1  0;
# define the mass at node 4 using the mass Command (page 29):
mass 4 $Mass 0.  0.  0.  0.  0.;
```

NOTE: The second command assigning the mass of a specific node will override any previous mass assignements to that node.

...Define Materials

The following is an example on how to define materials for reinforced-concrete structures. The examples assume that the variables have been defined apriori. If these commands are placed into an external file they can be used in a number of analyses without significant modifications using the source command.

```
# -------MaterialsRC.tcl-----------------------------------------------------------
set ConcreteMaterialType "inelastic";        # options: "elastic","inelastic"
set SteelMaterialType "hysteretic";                    # options: "elastic","bilinear","hysteretic"
# CONCRETE
if {$ConcreteMaterialType =="elastic"} {
        uniaxialMaterial Elastic $IDcore $Ec
        uniaxialMaterial Elastic $IDcover $Ec
}
if {$ConcreteMaterialType == "inelastic"} {
        uniaxialMaterial Concrete01 $IDcore $fc1C $eps1C $fc2C $eps2C;   # Core concrete
        uniaxialMaterial Concrete01 $IDcover $fc1U $eps1U $fc2U $eps2U; # Cover concrete
}
# STEEL
if {$SteelMaterialType == "elastic"} {
        uniaxialMaterial Elastic $IDsteel $Es
}
if {$SteelMaterialType == "bilinear"} {
        uniaxialMaterial Steel01    $IDsteel     $Fy     $Es     $Bs
}
if {$SteelMaterialType == "hysteretic"} {
```

```
        uniaxialMaterial Hysteretic $IDsteel $Fy $epsY $Fy1 $epsY1 $Fu $epsU -$Fy -$epsY -$Fy1 -$epsY1 -$Fu -$epsU $pinchX $pinchY $damage1 $damage2 $betaMUsteel
 }
```

...Define Elements

```
 # -----ELEMENTS.tcl------------------------------------------------------
# COLUMNS
set ColumnType "inelastic";        # options: "rigid" "elastic" "inelastic"
set np 5;                          # number of integration points
# source RCcircSection.tcl;        # proc to define circular fiber section for flexural characteristics
RCcircSection $IDcolFlex  $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol  $AbCol  $nfCoreR $nfCoreT $nfCoverR $nfCoverT
uniaxialMaterial Elastic $IDcolTors $GJ;              # Define torsional stiffness
section Aggregator $IDcolSec   $IDcolTors    T   -section   $IDcolFlex;     # attach torsion and flexure
geomTransf Linear     $IDcolTrans          0 0 1;              # Linear: no second-order effects
if {$ColumnType == "rigid"} {
        set $IyCol [expr $IyCol*$IyCol];
        set $IzCol [expr $IzCol*$IzCol];
        element elasticBeamColumn   1   1  3 $Acol   $Ec   $G   $J   $IyCol   $IzCol   $IDcolTrans
        element elasticBeamColumn   2   2  4 $Acol   $Ec   $G   $J   $IyCol   $IzCol   $IDcolTrans
}
if {$ColumnType == "elastic"} {
        element elasticBeamColumn   1   1  3 $Acol   $Ec   $G   $J   $IyCol   $IzCol   $IDcolTrans
        element elasticBeamColumn   2   2  4 $Acol   $Ec   $G   $J   $IyCol   $IzCol   $IDcolTrans
}
if {$ColumnType == "inelastic"} {
        element nonlinearBeamColumn  1   1  3   $np $IDcolSec  $IDcolTrans
        element nonlinearBeamColumn  2   2  4   $np $IDcolSec  $IDcolTrans
}
# BEAM
geomTransf Linear   $IDbeamTrans      0 0 1;       # define orientation of beam local axes
element elasticBeamColumn   3  3 4   $Abeam   $Ec   $G $J   $IyBeam   $IzBeam   $IDbeamTrans;
```

Defining Output

...Define Analysis-Output Generation

Different output may be generated depending on whether the analysis is static or dynamic. Here is an example:

```
# -------Output.tcl------------------------------------------------------------------------
set fDir "Data/";
file mkdir $fDir;                       # create directory
set ANALYSIS "Static";   # this variable would be passed in
set IDctrlNode 3;     # this variable would be passed in
if {$ANALYSIS == "Static"} {
        # Record nodal displacements -NODAL DISPLACEMENTS
        set fDstatFrame DStatFrame[expr $Xframe]
        set iNode "$IDctrlNode";     # can add node numbers to this list
        foreach xNode $iNode {
                set fNode Node$xNode
                set Filename $fDir$fDstatFrame$fNode
                recorder Node $Filename.out disp -time -node $xNode -dof 1 6;
        };   # end of xNode
        # Record element forces and deformations - COLUMNS
        set iEL "1 2"
        set fFstatFrame FStatFrame[expr $Xframe]
        set fDstatFrame DStatFrame[expr $Xframe]
        foreach xEL $iEL {
                set fEl El[expr $xEL]
                set iSEC "1 3 5"
                set Ffilename $fDir$fFstatFrame$fEl
                recorder Element $xEL -time -file $Ffilename.out localForce
                foreach xSEC $iSEC {
                        set fSec Sec[expr $xSEC]
                        set Dfilename $fDir$fDstatFrame$fEl$fSec
                        recorder Element $xEL -file $Dfilename.out -time section $xSEC deformations
                };      # end of xSEC
        };              # end of xEL
};              # end of static analysis
```

```
set ANALYSIS "Dynamic";   # this variable would be passed in
set GroundFile "ElCentro";     # this variable would be passed in
if {$ANALYSIS == "Dynamic"} {
        set fDDynaFrame DDynaFrame[expr $Xframe]
        set fGroundFile $GroundFile
        set Filename $fDir$fDDynaFrame$fGroundFile
        # Record nodal displacements
        recorder Node $Filename.out disp -time -node $lDctrlNode -dof 1
};     # end of dynamic analysis
```

...Define Data-Plot During Analysis

```
# ---------RecorderPlot.tcl-----------------------------------------------------------------------------
set pfile "Data/node.out";
recorder Node $pfile disp -time -node $lDctrlNode -dof 1
set title PushFrame$Xframe;
recorder plot $pfile $title  0 0 350 350 -columns 2 1

set pfile "Data/Elem1.out";
set title PushElem1;
recorder Element 1 -time -file $pfile globalForce
recorder plot $pfile $title 400 0 350 350 -columns 2 1
```

Gravity Loads

...Define Gravity Loads

```
# ---------DefineGravity.tcl----------------------------------------------------------------------------
set GravSteps 10
pattern Plain 1 Linear {
        load 3    0. -$Pdl   0.   0.   0.   -$Mdl ;  # Fx   Fy  Fz  Mx  My  Mz
        load 4    0. -$Pdl   0.   0.   0.   +$Mdl
 }
system UmfPack;          # solution procedure, how it solves system of equations
```

```
constraints Plain;          # how it handles boundary conditions, enforce constraints
test NormDispIncr 1.0e-5    10      0;
algorithm Newton;
numberer RCM;                       # renumber dof's to minimize band-width
integrator LoadControl [expr 1./$GravSteps] 1 [expr 1./$GravSteps] [expr 1./$GravSteps]
analysis Static
initialize; # this command will not be necessary in new versions of OpenSees
```

...Run Gravity Analysis

```
# ---------RunGravity.tcl-----------------------------------------------------------------------------
analyze $GravSteps                  # run gravity analysis
loadConst -time 0.0;                # keep gravity load and restart time -- lead to lateral-load analysis
```

Static Analysis

...Define Static Pushover Analysis

The following commands are executed once the gravity loads have been defined and applied

```
# ---------DefinePushover.tcl-------------------------------------------------------------------------------------
set analysis "STATIC";    # this variable would be passed in
# the following settings do not need to be here if they have been defined in the gravity analysis
system UmfPack;
constraints Plain;
test NormDispIncr 1.0e-5    10      0;
algorithm Newton;
numberer RCM;  analysis Static;
# ------------
set PUSHOVER "DispControl";    # run displacement-controlled static pushover analysis
        pattern Plain 2 Linear {
        load $IDctrlNode            100.0 0.0 0.0 0.0 0.0 0.0
}
if {$PUSHOVER == "LoadControl"} {
        integrator LoadControl  0.2 4  0.1 2.0
        set Nsteps 20
```

```
} elseif {$PUSHOVER == "DispControl"} {
        integrator DisplacementControl  $IDctrlNode   1   $DxPush 1 $DxPush $DxPush
        set Nsteps [expr int($DmaxPush/$DxPush)]
} else {
        puts stderr "Invalid PUSHOVER option"
}
```

...Run Static Pushover Analysis

While running a static pushover analysis may take a single command, convergence may not always be reached with a single analysis-parameter setting. A Tcl script which tries different solutions can be incorporated to improve the chances of convergence.

- ***No convergence issues***

The following command executes the static push-over analysis when convergence is not a problem.

```
# ---------RunPushover.tcl-----------------------------------------------------------------------------
analyze $Nsteps
```

- ***Convergence attemps***

The following Tcl script should be incorporated in the input file to run a number of attempts at convergence:

```
# ---------RunPushover2Converge.tcl-------------------------------------------------------------------
set ok [analyze $Nsteps]
# if analysis fails, try the following, performance is slowed inside this loop
if {$ok != 0} {
        set ok 0;
        set maxU $DmaxPush
        set controlDisp 0.0;
        test NormDispIncr   1.0e-8         20       0
        while {$controlDisp < $maxU && $ok == 0} {
                set ok [analyze 1]
                set controlDisp [nodeDisp $IDctrlNode 1]
                        if {$ok != 0} {
                        puts "Trying Newton with Initial Tangent .."
                        test NormDispIncr   1.0e-8         1000  1
                        algorithm Newton -initial
```

```
                        set ok [analyze 1]
                        test NormDispIncr   1.0e-8        20       0
                        algorithm Newton
                }
                if {$ok != 0} {
                        puts "Trying Broyden .."
                        algorithm Broyden 8
                        set ok [analyze 1]
                        algorithm Newton
                }
                if {$ok != 0} {
                        puts "Trying NewtonWithLineSearch .."
                        algorithm NewtonLineSearch .8
                        set ok [analyze 1]
                        algorithm Newton
                }
        };      # end while loop
};      # end original if $ok!=0 loop
if {$ok != 0} {
        puts "DispControl Analysis FAILED"
        puts "Do you wish to continue y/n ?";      # include if want to pause at analysis failure
        gets stdin ans;             # not recommended in parameter study
        if {$ans == "n"} done;              # as it interrupts batch file
} else {
        puts "DispControl Analysis SUCCESSFUL"
}
```

Dynamic Analysis

...Define Dynamic Ground-Motion Analysis

```
# ---------DefineDynamic.tcl-----------------------------------------------------------
wipeAnalysis
system UmfPack
```

```
constraints Plain
test NormDispIncr 1.0e-8 20 0;
algorithm Newton
numberer RCM
integrator Newmark $gamma $beta   $alphaM $betaK $betaKcomm $betaKinit;
analysis Transient
set Nsteps [expr int($TmaxGround/$DtAnalysis)];
# read a PEER strong motion database file, extracts dt from the header and converts the file
# to the format OpenSees expects for uniform ground motions
source ReadSMDFile.tcl;
set dir "GMfiles/"
set outFile $dir$GroundFile.g3;    # set variable holding new filename
set inFile $dir$GroundFile.th
ReadSMDFile $inFile $outFile dt;  # convert the ground-motion file
set GMfatt [expr $g*$GMfact];                # data in input file is factor of g
set Gaccel "Series -dt $dt -filePath $outFile -factor  $GMfatt";        # time series information
pattern UniformExcitation  2   1  -accel $Gaccel;                # create uniform excitation
```

...Run Dynamic Ground-Motion Analysis

```
# ---------RunDynamicGM.tcl----------------------------------------------------------------
analyze $Nsteps $DtAnalysis;
```

...Combine Input-File Components

A series of Tcl-script input-file components have been presented in this section. These components can be combined to perform a static lateral-load analysis of the portal frame under consideration using the source command:

```
# ----FullStaticAnalysis.tcl-------------------------------------------------------
wipe
model basic -ndm 3 -ndf 6
source Units&Constants.tcl
source MaterialParameters.tcl
source ElementParameters.tcl
```

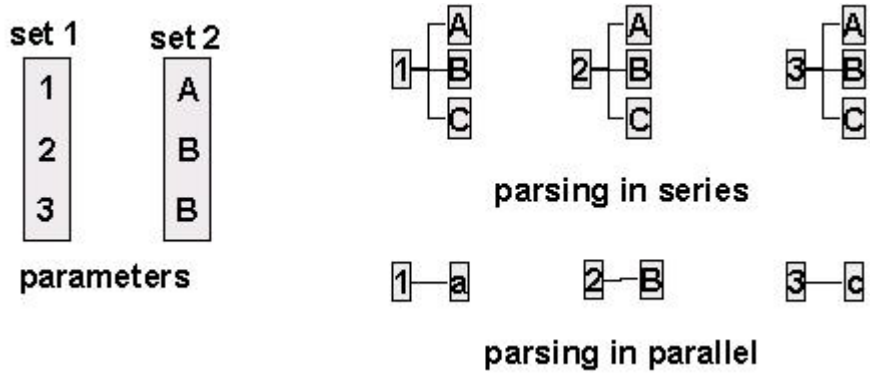
```
source GravityParameters.tcl
source AnalysisParameters.tcl
source MatlabOutput.tcl
source BuildModel&Nodesw/Variables--portal.tcl
source materialsRC.tcl
source RCcircSec.tcl
source Elements.tcl
source Output.tcl
source DefineGravity.tcl
source runGravity.tcl
source DefinePushover.tcl
source RunPushover2Converge.tcl
```

This method of breaking the input file into components is convenient when the size of the problem does not permit manageability of a single input file.

...Run Parameter Study

In a parameter study, the same analysis is performed on a number of models where only certain properties are varied, such as column height. There are two common types of parameter studies shown in this section: series and parallel parsing.

The following diagram illustrates the difference between series and parallel parsing for two parameter lists [1 2 3] and [A B C]:



- ***Parsing in series***

In this type of study, one parameter is held constant, while the others are parsed in sequence:

```
# -----ParameterStudySeries.tcl-------------------------------------------------
source units.tcl
set iHcol "[expr 5.*$ft] [expr 6.5*$ft]";        # column diameter
```

```
set iLcol "[expr 36*$ft] [expr 42*$ft]";        # column length
set Xframe 0;                  # initialize Frame Counter, used in output
foreach Hcol $iHcol {
        foreach Lcol $iLcol {
                set Xframe [expr $Xframe+1];
                set ANALYSIS "Static";
                source Analysis.tcl*
        };  # close iLcol loop
};  # close iHcol loop
```

***NOTE:** The file Analysis.tcl contains all the model and analysis commands.

- ***Parsing in parallel***

In this study, the ith elements of each parameter list are considered together, resulting in fewer study models.

```
# -----ParameterStudyParallel.tcl-----------------------------------------------------
source units.tcl
set iHcol "[expr 5.*$ft] [expr 6.5*$ft]";       # column diameter
set iLcol "[expr 36*$ft] [expr 42*$ft]";        # column length
set Xframe 0;                  # initialize Frame Counter, used in output
foreach Hcol $iHcol Lcol $iLcol{
        set Xframe [expr $Xframe+1];
        set ANALYSIS "Static";
        source Analysis.tcl*
 };  # close iHcol & iLcol loop
 *NOTE: The file Analysis.tcl contains all the model and analysis commands.
```

...Run Moment-Curvature Analysis on Section

A procedure for performing section analysis (only does moment-curvature, but can be easily modified to do any mode of section reponse):

```
# ---------MPhiProc.tcl-------------------------------------------------------------------------
```

```
# Sets up a recorder which writes moment-curvature results to file
#   make sure the node and element numbers are not used elsewhere in the model
#   this procedure is set up for a 3-D problem: 3 dimensions/node, 6 dof/node
# Arguments
#          secTag -- tag identifying section to be analyzed
#          axialLoad -- axial load applied to section (negative is compression)
#          maxK -- maximum curvature reached during analysis
#          numIncr -- number of increments used to reach maxK (default 100)
proc MomentCurvature {secTag axialLoad maxK {numIncr 100} } {
        node 1001 0.0 0.0 0.0;   # Define two nodes at (0,0)
        node 1002 0.0 0.0 0.0
        fix 1001 1 1 1 1 1 1; # Fix all degrees of freedom except axial and bending
        fix 1002 0 1 1 1 1 0
        element zeroLengthSection  2001   1001   1002  $secTag
        recorder Node Mphi.out disp -time -node 1002 -dof 6;# output moment & curvature

        integrator LoadControl 0 1 0 0;   # Define analysis parameters
        system SparseGeneral -piv;          # Overkill, but may need the pivoting!
        test NormUnbalance 1.0e-9 10
        numberer Plain;
        constraints Plain;
        algorithm Newton;
        analysis Static;

        pattern Plain 3001 "Constant" {
                load 1002 $axialLoad 0.0 0.0 0.0 0.0 0.0
        };   # Define constant axial load
        analyze 1;  # Do one analysis for constant axial load

        pattern Plain 3002 "Linear" {
                load 1002 0.0 0.0 0.0 0.0 0.0 1.0
        };   # Define reference moment
        set dK [expr $maxK/$numIncr]; # Compute curvature increment
        # Use displacement control at node 1002 for section analysis, dof 6
        integrator DisplacementControl 1002 6 $dK 1 $dK $dK
        analyze $numIncr;          # Do the section analysis
}
```

When including this procedure, ensure that the node and element numbers used by it are not used elsewhere in the OS model.

The above procedure may be incorporated into the static pushover analysis file:

```
# ----MomentCurvature.tcl------------------------------------------------------------
wipe
model basic -ndm 3 -ndf 6
source Units&Constants.tcl
source MaterialParameters.tcl
source ElementParameters.tcl
source GravityParameters.tcl
source materialsRC.tcl
source RCcircSec.tcl
RCcircSection $IDcolSec  $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol  $AbCol  $nfCoreR $nfCoreT $nfCoverR $nfCoverT
source MPhiProc.tcl
set phiYest [expr $epsY/(0.7*$Hcol)];        # estimate yield curvature
set axialLoad -$Pdl;                         # define axial load -- +tension in Mom-curv analysis
set maxK [expr 20*$phiYest];                 # maximum curvature reached during analysis
MomentCurvature $IDcolSec $axialLoad $maxK;
```

...Determine Natural Period & Frequency

The natural period and frequency of the structure can be determined at any point during the analysis using the *eigen* (page 260) command. In turn, these quantities can be stored as variables and used in defining analysis parameters, such as rayleigh-damping parameters:

```
# ------------PeriodFreq&Damping.tcl-----------------------------------------------------------
# determine Natural Period, Frequency & damping parameters for SDOF
set $xDamp 0.02;          # damping ratio (0.02-0.05-typical)
set lambda [eigen 1]
set omega [expr pow($lambda,0.5)]
set Tperiod      [expr 2*$PI/$omega];      # period (sec.)
puts $Tperiod
set alphaM       0;                # stiffness-prop. RAYLEIGH damping parameter; D = alphaM*M
set betaK                0;        # stiffness proportional damping;                +beatK*KCurrent
set betaKcomm            [expr 2*$xDamp/$omega];        # mass-prop. RAYLEIGH damping parameter;
```

```
        +betaKcomm*KlastCommitt
set betaKinit    0;             # initial-stiffness proportional damping          +beatKinit*Kini
```

CHAPTER 32

Getting Started with OpenSees

Under the NEESgrid support, a *Getting Started with OpenSees* (http://peer.berkeley.edu/~silvia/OpenSees/gettingstarted/) document has been produced.

In This Chapter

Introduction

Modern earthquake engineering utilizes modeling and simulation to understand the behavior and performance of systems during earthquakes. With the support of the National Science Foundation, the Pacific Earthquake Engineering Research Center (PEER) has developed the Open System for Earthquake Engineering Simulation, OpenSees for short, as a software platform for research and application of simulation for structural and geotechnical systems. The OpenSees software framework uses object-oriented methodologies to maximize modularity and extensibility for implementing models for behavior, solution methods, and data processing and communication procedures. The framework is a set of inter-related classes, such as domains (data structures), models, elements (which are hierarchical), solution algorithms, integrators, equation solvers, and databases. The classes are as independent as possible, which allows great flexibility in combining modules to solve simulation problems for buildings and bridges, including soil and soil-structure-foundation interaction, and most recently including reliability computational modules. The open source software is managed and made available to users and developers through the OpenSees website at *http://opensees.berkeley.edu* (http://opensees.berkeley.edu).

The software architecture and open-source approach for OpenSees provide many benefits to users interested in advanced simulation of structural and geotechnical systems with realistic models of nonlinear behavior. First, the modeling approach is very flexible in that allows selection and various combinations of a number of different element formulations and material formulations, along with different approximations of kinematics to account for large-displacements and P- effects. As an open-source project, developers and researchers are using the extensible features of the software architecture to add additional capability. A second advantage is that there is a wide range of solution procedures and algorithms that the user can adapt to solve difficult nonlinear problems for static and dynamic loads. Another feature is that OpenSees has a fully programmable scripting language for defining models, solution procedures, and post-processing that can provide simple problem solving capability, as illustrated in this manual, or very sophisticated modeling and parameters studies of large, complex systems. Finally, OpenSees provides a flexible interface to computer resources, storage and databases, and network communication to take advantage of high-end computing systems. Structural and geotechnical models can be analyzed from desktop PC's to parallel computers within OpenSees.

As an advanced platform for computational simulation, OpenSees provides an important resource for the National Science Foundation-sponsored George E. Brown, Jr. Network for Earthquake Engineering Simulation (*NEES* (http://www.nees.org)), and it has been adopted by *NEESgrid* (http://www.neesgrid.org) System Integration project as the NEES simulation component. The NEESgrid decision to utilize OpenSees and adapt it to interface with other NEESgrid resources provides an important capability for NEES researchers and users. The modular design of OpenSees means that it can be customized for the integrating physical and computation simulation through data repositories, visualization, and hybrid control for advanced experimental methods, all of which meet important NEES objectives.

Open source software, such as OpenSees, requires community support and participation. The objective of this “Getting Started” manual is to provide an introduction so that users of OpenSees can obtain, install, and begin using the software to solve problems.

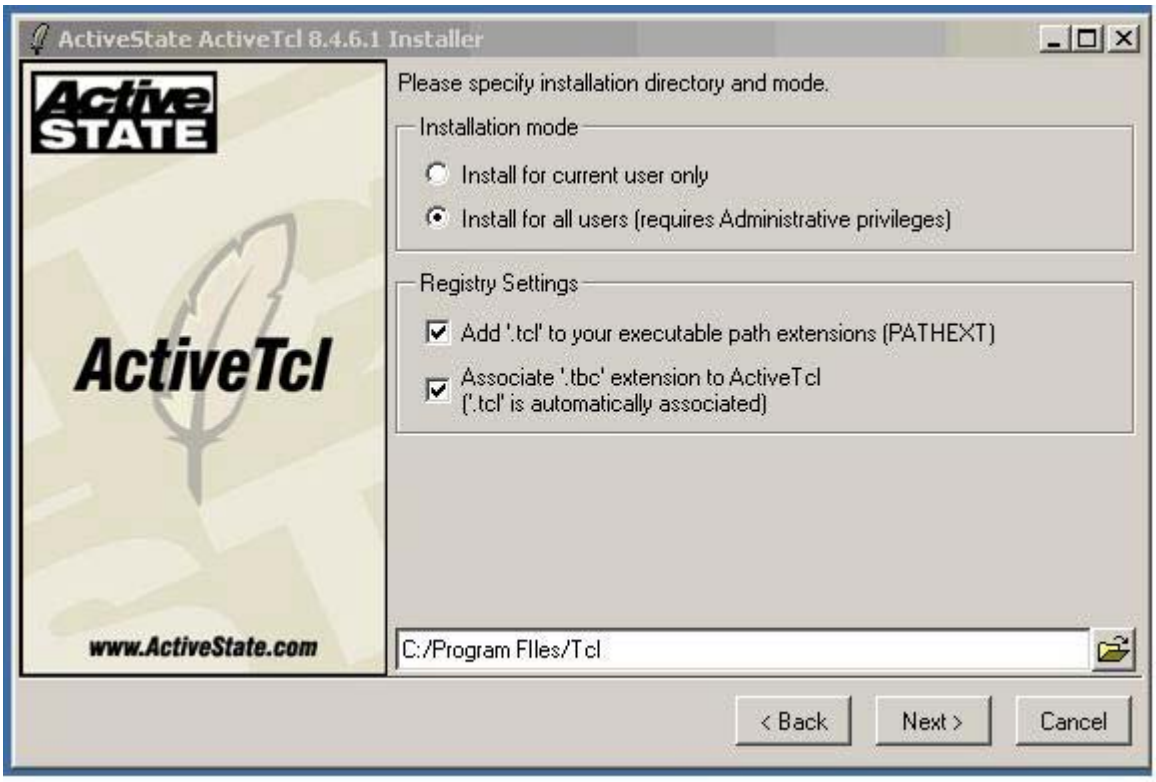
Download OpenSees

To download and install OpenSees the user is required to download both the OpenSees and Tcl/Tk packages. The OpenSees and Tcl/Tk packages can both be downloaded from the OpenSees binaries webpage http://opensees.berkeley.edu/binaries.html. This page can be found using the quick links pull down menu from any of the OpenSees web pages. The binaries download page will be similar to that shown below.

At this page the user is required to download two files OpenSees.X.X.X.exe and tcl/tk Y.Y.Y. This can be done by selecting (clicking) on the links located in the box labeled DOWNLOAD Windows Binaries.

The file downloaded by clicking on the OpenSees.X.X.X.exe link is a zip file, from which the OpenSees.X.X.X.exe file can be extracted using your favorite extractor. The user can place this executable anywhere they wish. It is recommended to place it in a directory near where all your scripts will be stored.

The file downloaded by clicking the tcl/tk Y.Y.Y link is an installer for Tcl/Tk. By clicking on the file the user is brought through a series of screens. The first screen shows the package information for Y.Y.Y, the second screen has the license agreement, you must accept the license to proceed with the installation. The third screen specifies the installation mode and location. Here the user must change the default installation location from C:/Tcl to C:/Program Files/Tcl (there is a space between Program and Files), as shown in the image below.



This is the only change the user must make. User now just keeps selecting Next until done.

Run OpenSees

There are three ways that OpenSees/Tcl commands can be executed:

- ***Interactive***

Commands can be input directly at the propt, as shown in the figure (Win32 version):

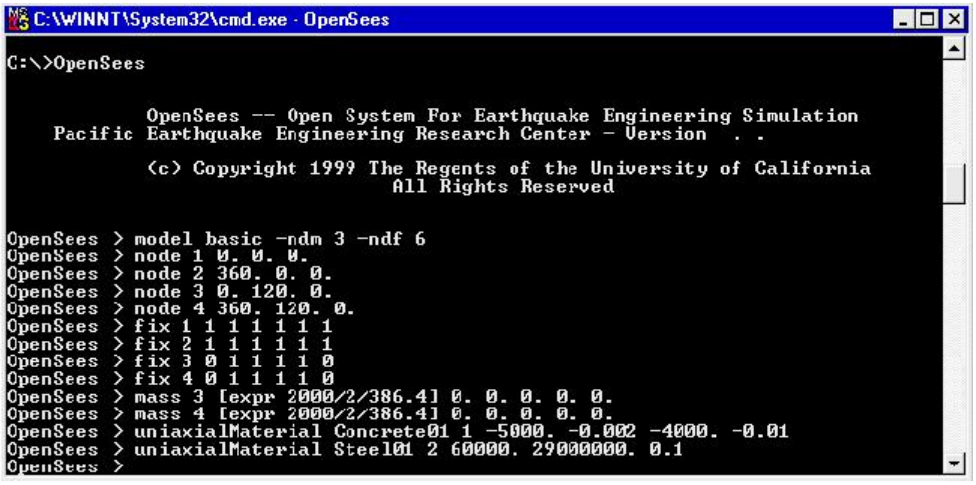
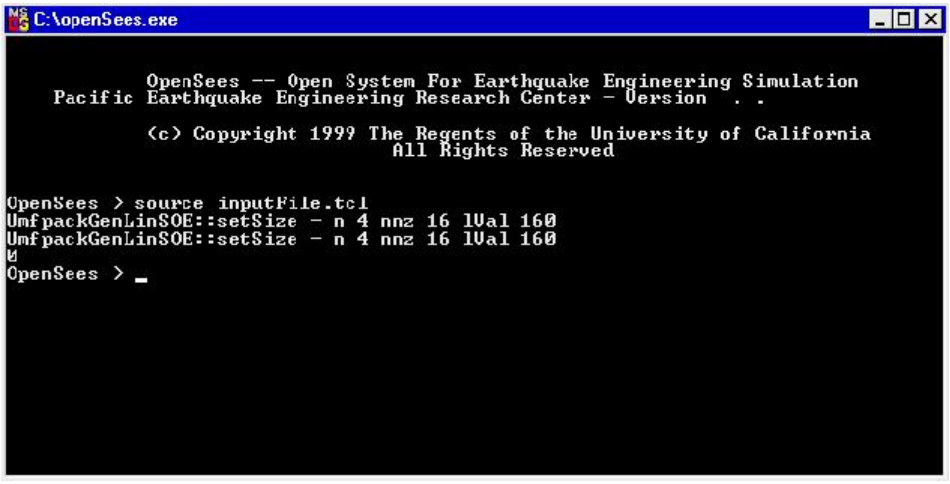


Figure 73: Run OpenSees -- Interactive

➢ *Execute Input File at OpenSees prompt*

This method is the most-commonly used one. An external file containing the input commands can be generated a-priori (inputFile.tcl) and be executed at the OpenSees prompt by using the source command. The generation of the input script files is presented in this chapter. The file execution is shown in the figure (Win32 version):

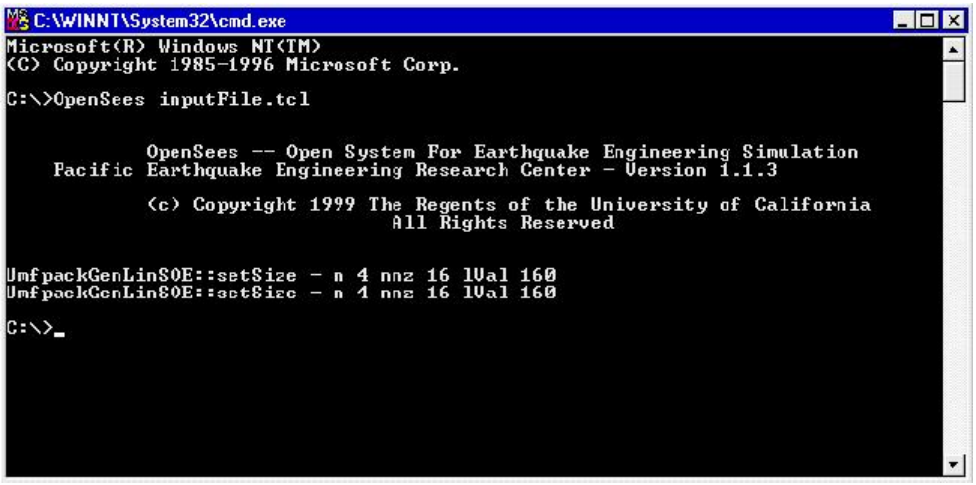
Figure 74: Run OpenSees -- Source File



➢ *Batch Mode*

The previously-created input file containing the Tcl script commands necessary to execute the analsis can also be executed at the MS-DOS/Unix prompt, as shown in the figure (Win32 version):

Figure 75: Run OpenSees -- Batch Mode

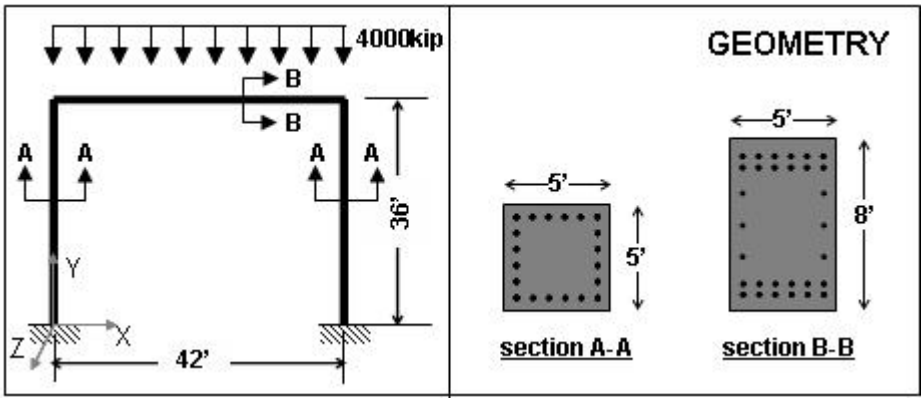


Problem Definition

A portal frame will be used to demonstrate the OpenSees commands. A structural model will be defined first. Subsequently, a number of static and dynamic analyses will be defined and implemented.

The structural model consists of the planar portal frame shown in the figure below:

Figure 76: Getting Started -- Problem Definition, Geometry



The columns and beam will be modeled as elastic elements. At a more advanced level, these elements can be replaced by more refined element models.

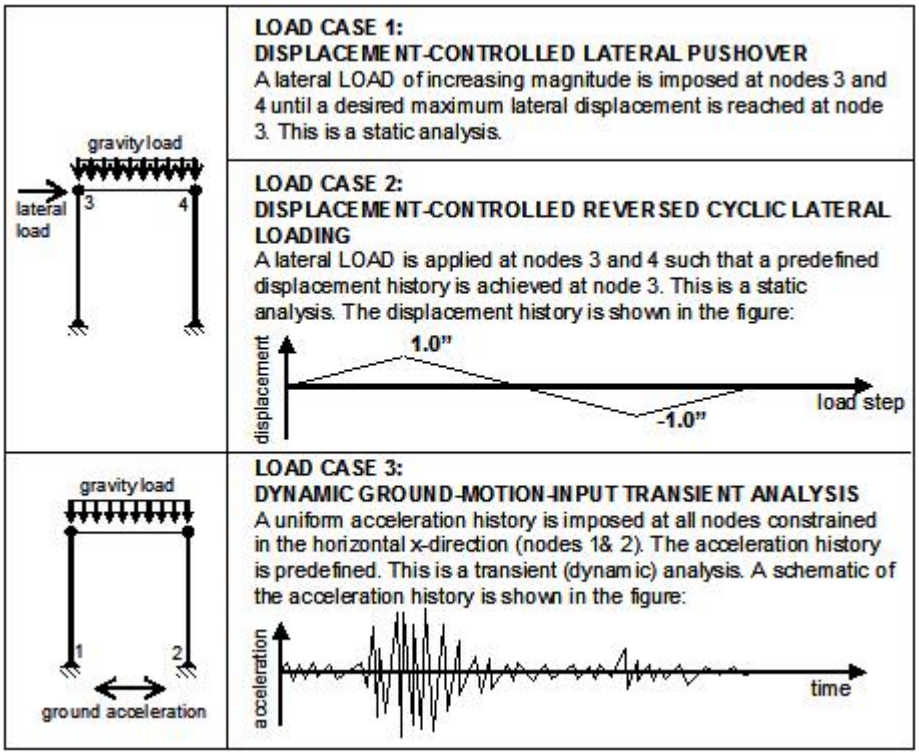
In the analysis phase, the frame will be subjected to three different load cases:

1 DISPLACEMENT-CONTROLLED LATERAL PUSHOVER;

2 DISPLACEMENT-CONTROLLED REVERSED CYCLIC LATERAL LOADING;

3 DYNAMIC GROUND-MOTION-INPUT TRANSIENT ANALYSIS.

In all cases, however, the frame will be subjected to constant static gravity loads:



Model Builder

Defining the model builder expands the Tcl command library to include OpenSees-specific commands, such as node and element definition, etc. Currently, there is only one model builder available, *basic model builder* (page 26), this is the model builder that includes all the commands presented in this library.

The model builder also defines the number of dimensions (ndm) and degrees of freedom per node (ndf):

model BasicBuilder -ndm $ndm <-ndf $ndf>

For a 2-D problem, you really only need three degrees of freedom at each node, the two translations in the plane and the rotation about the plane's normal:

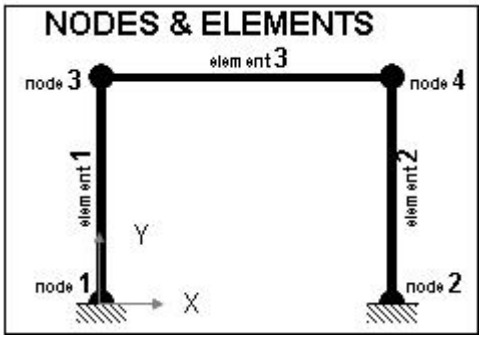
```
model basic -ndm 2 -ndf 3
```

Nodes

At this point the user needs to decide which units will be used. In this demonstration, inches and kips will be used for length and force. Seconds will be used for time.

The assignment of node and element numbers is defined in the figure below:

Figure 77: Getting Started -- Nodes & Elements



In a 2D problem only the x and y coordinates need to be defined, using the *node* (page 28) command:

node $nodeTag (ndm $coords) <-mass (ndf $MassValues)>

```
node 1   0     0
node 2   504 0
node 3   0     432
node 4   504 432
```

The boundary conditions are defined using the *fix* (page 30) command:

fix $nodeTag (ndf $ConstrValues)

with three degrees of freedom per node are constrained:

```
fix 1 1    1    1
fix 2 1    1    1
fix 3 0    0    0
fix 4 0    0    0
```

where a fixed constraint is defined with a 1, a free constraint is defined with a 0.

Nodal masses are typically defined at the same time as the nodal coordinates. The nodal mass is used to calculate the eigenvalues and to perform the dynamic analysis. Only the nodal mass in the horizontal direction will be defined in this demonstration. Nodal masses can either be defined within the *node* (page 28) command, or the can be "appended" using the *mass* (page 29) command:

mass $nodeTag (ndf $MassValues)

```
mass 3  5.180.                  0.
mass 4  5.180.                  0.
```

The mass value was calculated by dividing the nodal weight (1/2 of the total super-structure weight) by the gravitational constant g (32 ft/sec):

$$\text{mass} = \frac{\frac{4000 \cdot \text{kip}}{2}}{\left(32.2 \cdot \frac{\text{ft}}{\text{sec}}\right) \cdot \left(\frac{12 \cdot \text{inch}}{1 \cdot \text{ft}}\right)}$$

Elements

The elastic columns and beams are defined using the *elastic beam column element* (page 148). The characteristics of a 2-D elastic element depend on the material modulus and the section area and moment of inertia. Because the elements in this frame represent reinforced-concrete elements, the value of 4227 ksi for the elastic modulus of concrete will be used.

The following values represent the area and moment of inertia of the columns and beams:

	COLUMNS	BEAM
Area	$(5\cdot12)\cdot(5\cdot12) = 3600$	$(5\cdot12)\cdot(8\cdot12) = 5760$
Iz	$\frac{1}{12}\cdot(5\cdot12)\cdot(5\cdot12)^3 = 1080000$	$\frac{1}{12}\cdot(5\cdot12)\cdot(8\cdot12)^3 = 4423680$

The transformation command defines how the element coordinates correlate to the global model coordinates. In a 2D problem, element orientation does not need to be considered, and can be the same for all elements. The *linear transformation* (page 200) will be used in this demonstration:

geomTransf Linear $transfTag <-jntOffset $dXi $dYi $dXj $dYj>

```
geomTransf Linear 1
```

The following commands define the two columns (element 1 and 2) and the beam (element 3):

element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag

```
element elasticBeamColumn 1   1   3   3600   4227   1080000   1
element elasticBeamColumn 2   2   4   3600   4227   1080000   1
element elasticBeamColumn 3   3   4   5760   4227   4423680   1
```

Recorders

The recorder command is used to define the analysis output.

The node recorder will be used to output the horizontal and vertical displacements at node 3 into a file named Node3.out:

recorder Node <-file $fileName> <-time> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-region $RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType

```
recorder Node -file Node3.out -time -node 3 -dof 1 2 disp
```

The element recorder will be used to output the element forces. Element forces for element 1 will be output into file Element1.out:

recorder Element <-file $fileName> <-time> <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-region $regTag> <-ele all> ($arg1 $arg2 ...)

```
recorder Element -file Element1.out -time -ele 1 force
```

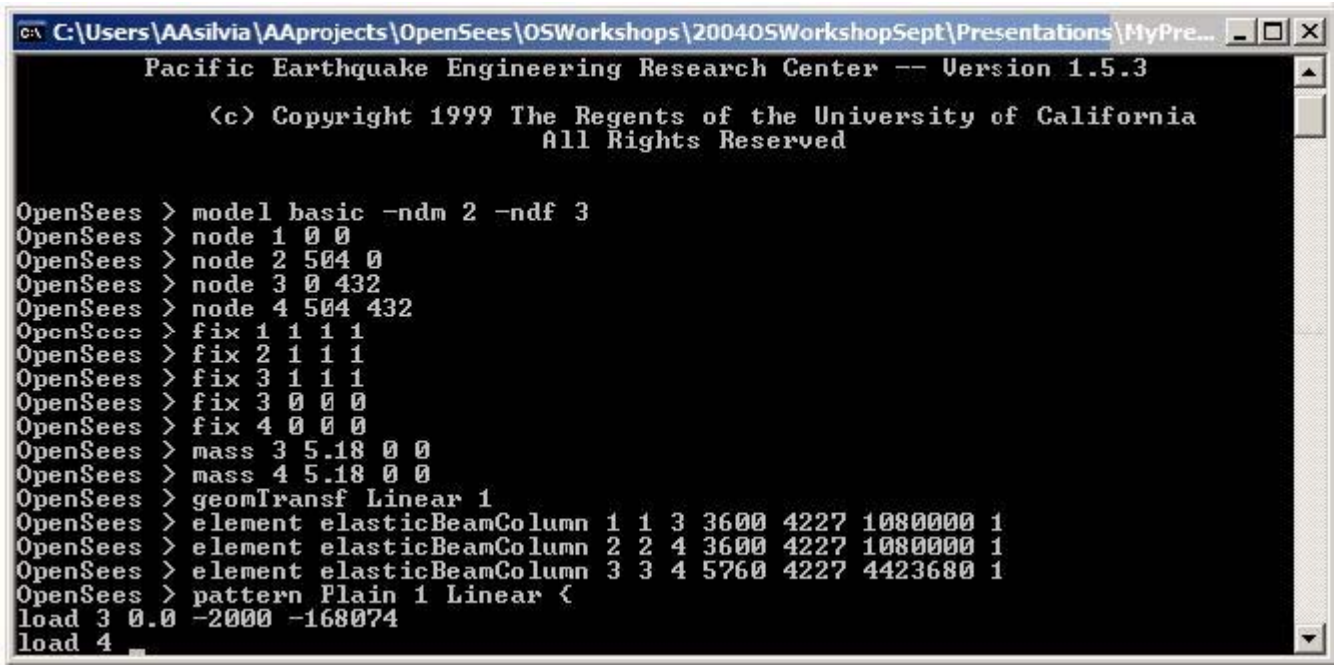
Summary of Model-Building Input File

The following is a compilation of all the commands necessary to build the model:

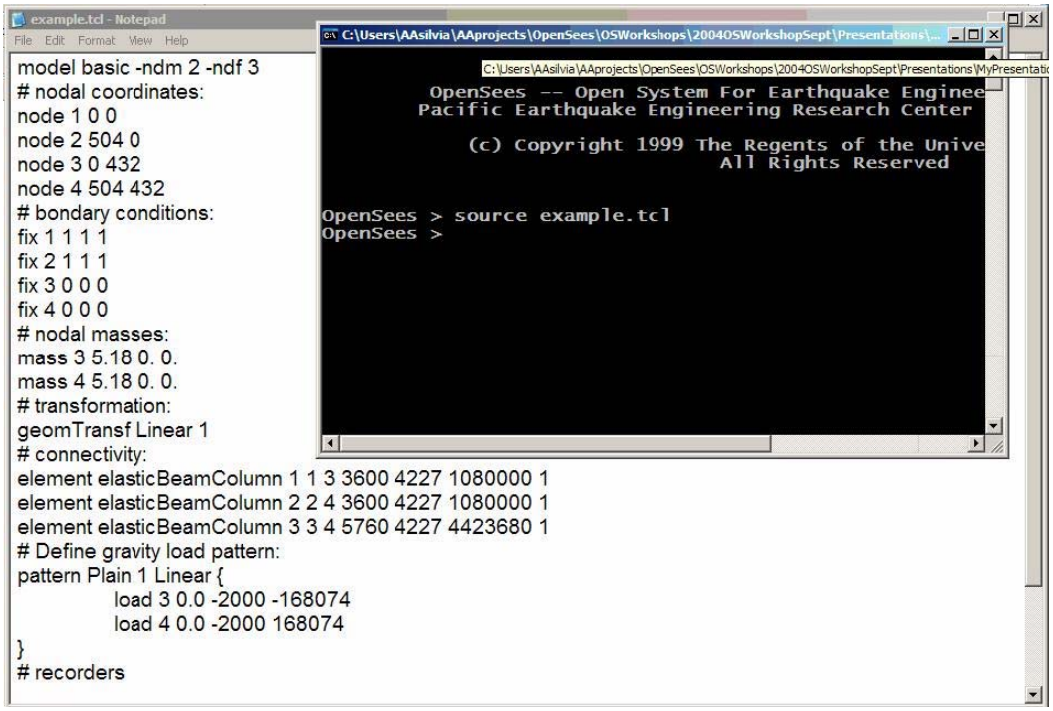
```
model basic -ndm 2 -ndf 3
# nodal coordinates:
node 1 0 0
node 2 504 0
node 3 0 432
node 4 504 432
# bondary conditions:
fix 1 1 1 1
fix 2 1 1 1
fix 3 0 0 0
fix 4 0 0 0
# nodal masses:
mass 3 5.18 0. 0.
mass 4 5.18 0. 0.
# transformation:
```

```
geomTransf Linear 1
# connectivity:
element elasticBeamColumn 1 1 3 3600 4227 1080000 1
element elasticBeamColumn 2 2 4 3600 4227 1080000 1
element elasticBeamColumn 3 3 4 5760 4227 4423680 1
# recorders
recorder Node -file Node3.out -time -node 3 -dof 1 2 disp
recorder Element -file Element1.out -time -ele 1 force
```

The above commands can be input line by line at the OpenSees command prompt:



Otherwise, they can be saved into an input file called **example.tcl**. This file can then be sourced in from the OpenSees command line:



Loads and Analysis

In OpenSees applying loads is a three-step process:

1. You must first define the loads in a load pattern

2. You must then define the analysis and its features

3. The loads are then applied when you execute the analysis

1. Load definition

Loads are defined using the pattern command. Three types of patterns are currently available:

a. plain Pattern -- this pattern is used to define the following:

i. nodal loads, such as gravity loads and lateral loads (or load-controlled nodal displacements)

ii. single-point constraints, such as displacement control at a node (typically used for a constant displacement at a node)

iii. element loads, such as distributed gravity loads along the element (this is a new option, which still needs documentation).

b. UniformExcitation Pattern -- this type of pattern imposes a user-defined acceleration record to all fixed nodes, in a specified direction.

c. MultipleSupport Pattern -- this type of pattern imposes a user-defined displacement record at specified nodes, in a specified direction, or a ground-motion record.

2. Analysis definition and features

The analysis-definition part of OpenSees allows the user to define the different linear and nonlinear analysis tools available. For each analysis, the following items need to be defined, preferably in this order:

constraints	The constraints command is used to construct the ConstraintHandler object. Constraints enforce a relationship between degrees-of-freedom. The ConstraintHandler object determines how the constraint equations are enforced in the analysis.
numberer	The numberer 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.
system	The system command is used to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
test	The test command is used to construct a ConvergenceTest object. Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
algorithm	The algorithm command is used to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
integrator	The *integrator* (page 249) command is used to construct the Integrator object. The Integrator object determines the meaning of the terms in the system of equation object. 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

analysis	The *analysis* (page 256) command is used to construct the Analysis object. This analysis object is constructed with the component objects previously created by the analyst. All currently-available analysis objects employ incremental solution strategies. There are three types of analysis currently available: *Static Analysis* (page 256) *Transient Analysis* (page 257) *VariableTransient Analysis* (page 258)

3. Analysis execution

The analysis is executed using the *analyze* (page 261) command. This command moves the analysis forward by the specified number of steps.

Gravity Loads

Gravity loads are independent of the type of lateral loading and here they are considered part of the structural model.

NODAL FORCES & MOMENTS

Because the beam is an elastic element, the vertical load distributed along the horizontal member can be represented by nodal forces and moments. The nodal forces are distributed equally to the two end nodes. The nodal bending moments are equal and opposite:

The nodal force is equal to one half of the superstructure weight:

$$\text{Force} = \frac{4000 \cdot \text{kip}}{2} = 2000 \cdot \text{kip}$$

the distributed load is calculated by dividing the total load by the beam length:

$$\text{DistributedLoad} = \frac{4000 \cdot \text{kip}}{(42 \cdot \text{ft}) \cdot \left(12 \cdot \frac{\text{inch}}{\text{ft}}\right)} = 7.94 \cdot \frac{\text{kip}}{\text{inch}}$$

The bending moment is then calculated from the distributed load:

$$\text{Moment} = \frac{\text{DistributedLoad} \cdot \text{BeamLength}^2}{12} = \frac{\left(7.94 \cdot \frac{\text{kip}}{\text{inch}}\right) \cdot \left(42 \cdot \text{ft} \cdot 12 \cdot \frac{\text{inch}}{\text{ft}}\right)^2}{12} = 168074 \cdot \text{kip} \cdot \text{in}$$

LOAD PATTERN DEFINITION

Like all loads in OpenSees, gravity loads require two steps. The first step defines the load into a load *pattern* (page 214), the second applies the load pattern and the associated gravity load. The *plain pattern* (page 214) command with a *linear time series* (page 209) is used in the load definition:

```
pattern Plain $patternTag (TimeSeriesType arguments) {


load $nodeTag (ndf $LoadValues)


}
```

```
pattern Plain 1 Linear {
    load 3 0.0 -2000 -168074
    load 4 0.0 -2000 168074
}
```

CREATE ANALYSIS

The *constraints* (page 232) command is used to construct the ConstraintHandler object. Constraints enforce a relationship between degrees-of-freedom. The ConstraintHandler object determines how the constraint equations are enforced in the analysis.

```
constraints Transformation
```

The *numberer* (page 237) 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. With the RCM numberer nodes are assigned degrees-of-freedom using the Reverse Cuthill-McKee algorithm

```
numberer RCM
```

The *system* (page 239) command is used to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. The BandGeneral command is used to construct an un-symmetric banded system of equations object which will be factored and solved during the analysis using the Lapack band general solver

```
system BandGeneral
```

The *test* (page 242) command is used to construct a ConvergenceTest object. Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step. The convergence test is applied to the following equation:

$$K\Delta U = R$$

The NormDispIncr test performs the following check:

Norm Displacement Increment $\sqrt{\Delta U^T \Delta U} < tol$

test NormDispIncr $tol $maxNumIter <$printFlag>

```
test NormDispIncr 1.0e-6 6
```

The *algorithm* (page 245) command is used to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.

```
algorithm Newton
```

The *integrator* (page 249) command is used to construct the Integrator object. The Integrator object determines the meaning of the terms in the system of equation object. 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

The system of nonlinear equations is of the form:

static analysis: $R(U,l) = \lambda P^* - F_R(U)$

The type of integrator used in the analysis is dependent on whether it is a static analysis or transient analysis:

static analysis

LoadControl (page 250) $\lambda_n = \lambda_{n-1} + d\lambda$

DisplacementControl (page 251) $Uj_n = Uj_{n-1} + dUj$

MinUnbalDispNorm (page 252) $d/d\lambda\,(dU_n{}^T dU_n)=0$

ArcLength (page 252) $dU_n{}^T dU_n + \alpha^2\, d\lambda_n = ds^2$

integrator LoadControl $dLambda1 <$Jd $minLambda $maxLambda>

```
integrator LoadControl 0.1
```

The *analysis* (page 256) command is used to construct the *Analysis object* (page 229). This analysis object is constructed with the component objects previously created by the analyst. All currently-available analysis objects employ incremental solution strategies.

```
analysis Static
```

The *analyze* (page 261) command is used to apply the gravity load (in 10 steps) and all loads that have been defined at this point, and the *loadConst* (page 266) command maintains the gravity load constant for the remainder of the the analyses and resets the current time to zero.

analyze $numIncr <$dt> <$dtMin $dtMax $Jd>

loadConst <-time $pseudoTime>

```
analyze 10
loadConst -time 0.0
```

Summary of Gravity Loads

The gravity loads can be placed into a file, called GravityLoads.tcl with the following commands:

```
pattern Plain 1 Linear {
    load 3 0.0 -2000 -168074
```

```
    load 4 0.0 -2000 168074
}
constraints Transformation
numberer RCM
system BandGeneral
test NormDispIncr 1.0e-6 6
algorithm Newton
integrator LoadControl 0.1
analysis Static
analyze 10
loadConst -time 0.0
```

This file can be sourced in after the example.tcl file:

```
source example.tcl
source GravityLoads.tcl
```

Lateral Loads -- Static Pushover

The following commands assume that the example.tcl and the gravityloads.tcl files have been run. The wipe command should be used at the beginning of the input to clear any previous OpenSees-objects definition:

```
wipe
source example.tcl
source gravityloads.tcl
```

In this analysis, a lateral displacement of increasing amplitude is imposed at the free nodes (3 and 4). The imposed displacements are applied using a displacement-control integrator, where the load factors are scaled to reach the desired displacement (compared to an imposed-displacement analysis). This method is the most efficient when you have a non-strength-degrading system.

The first step is to define the load pattern. To do so, we create a new load pattern (ID tag 2) for the lateral loads

```
pattern Plain 2 Linear {
    load 3 100.0 0.0 0.0
    load 4 100.0 0.0 0.0
}
```

Most of the analysis features that were defined in the gravity analysis are still valid since this type of analysis is also static. The loads, however, are applied differently. While gravity was applied as a load, using the LoadControl integrator, the DisplacementControl integrator is used in this pushover:

integrator DisplacementControl $nodeTag $dofTag $dU1 <$Jd $minDu $maxDu>

The load is applied to node 3, in the direction of DOF 1, with a displacement increment of 0.1

```
integrator DisplacementControl  3 1   0.1
```

A total of a 1-inch displacement needs to be applied, hence 10 steps are needed:

```
analyze 10
```

Lateral Loads -- Cyclic Lateral Load

The following commands assume that the example.tcl and the gravityloads.tcl files have been run. The wipe command should be used at the beginning of the input to clear any previous OpenSees-objects definition:

```
wipe
source example.tcl
source gravityloads.tcl
```

In this analysis, a lateral displacement cycle (positive and negative) of a prescribed amplitude is imposed at the free nodes (3 and 4). The imposed displacements are applied using a displacement-control integrator, where the load factors are scaled to reach the desired displacement (compared to an imposed-displacement analysis). This method is the most efficient when you have a non-strength-degrading system.

The first step is to define the load pattern. To do so, we create a new load pattern (ID tag 3) for the lateral loads

```
pattern Plain 3 Linear {
    load 3 100.0 0.0 0.0
    load 4 100.0 0.0 0.0
}
```

Most of the analysis features that were defined in the gravity analysis are still valid since this type of analysis is also static. The loads, however, are applied differently. While gravity was applied as a load, using the LoadControl integrator, the DisplacementControl integrator is used in this pushover. Similarly, while in the pushover analysis a single load increment was used, variable load increments are used to reverse the loading from positive to negative, and back to positive.

The load is applied to node 3, in the direction of DOF 1, with a displacement increment of 1 for the first rise to amplitude 1, -2 for the reversal to amplitude -1, and again positive 1 for the reversal back to amplitude zero:

```
integrator DisplacementControl  3   1   0.1
analyze 10
integrator DisplacementControl  3   1   -0.2
analyze 10
integrator DisplacementControl  3   1   0.1
analyze 10
```

this can be put into a foreach loop:

```
foreach Dincr {0.1 -0.2 0.1} {
    integrator DisplacementControl  3   1   $Dincr
    analyze 10
```

```
}
```

Lateral Loads -- Dynamic ground motion

The following commands assume that the example.tcl and the gravityloads.tcl files have been run. The wipe command should be used at the beginning of the input to clear any previous OpenSees-objects definition:

```
wipe
source example.tcl
source gravityloads.tcl
```

The dynamic ground-motion analysis is a transient, rather than static, type of analysis. Therefore, most of the analysis components should be redefined.

First of all, the load pattern needs to be defined. The load pattern here consists of defining an acceleration time-history applied at the support nodes. The time-history is defined in a file named BM68elc.th, taken from the PEER strong-motion database. The first lines of this file are:

```
PACIFIC ENGINEERING AND ANALYSIS STRONG-MOTION DATA
BORREGO MOUNTAIN 04/09/68 0230, EL CENTRO ARRAY #9, 270
ACCELERATION TIME HISTORY IN UNITS OF G
NPTS=  4000, DT= .01000 SEC
  -.1368849E-02  -.1659410E-02  -.1466880E-02  -.6865326E-03  -.6491235E-03
  -.6172128E-03  -.5942289E-03  -.5720329E-03  -.5517003E-03  -.5367939E-03
  -.5300330E-03  -.5315104E-03  -.5389920E-03  -.5492582E-03  -.5592027E-03
  -.5659958E-03  -.5672101E-03  -.5617805E-03  -.5502959E-03  -.5347288E-03
  -.5176619E-03  -.5013709E-03  -.4873454E-03  -.4763228E-03  -.4683559E-03
  -.4626830E-03  -.4579708E-03  -.4512405E-03  -.4376077E-03  -.4130071E-03
  -.3772566E-03  -.3363394E-03  -.3030926E-03  -.2926074E-03  -.3144186E-03
  -.3668375E-03  -.4373818E-03  -.5104884E-03  -.5745380E-03  -.6248976E-03
  -.6621411E-03  -.6878470E-03  -.7014600E-03  -.6985488E-03  -.6737667E-03
  -.6258232E-03  -.5616336E-03  -.4955459E-03  -.4432164E-03  -.4144737E-03
  -.4103149E-03  -.4251781E-03  -.4493034E-03  -.4742715E-03  -.4942019E-03
  -.5021476E-03  -.4907415E-03  -.4549906E-03  -.4008473E-03  -.3457893E-03
  -.3076823E-03  -.2975411E-03  -.3148410E-03  -.3469618E-03  -.3746677E-03
  -.3814194E-03  -.3598713E-03  -.3127679E-03  -.2503611E-03  -.1870004E-03
  -.1381307E-03  -.1166398E-03  -.1296596E-03  -.1730662E-03  -.2365307E-03
  -.3092055E-03  -.3795541E-03  -.4410602E-03  -.4886626E-03  -.5170590E-03
  -.5252823E-03  -.5139311E-03  -.4858098E-03  -.4459504E-03  -.3992677E-03
  -.3528575E-03  -.3119584E-03  -.2767747E-03  -.2493392E-03  -.2343354E-03
  -.2328132E-03  -.2443271E-03  -.2710242E-03  -.3142711E-03  -.3756243E-03
  -.4585760E-03  -.5641321E-03  -.6901264E-03  -.8277058E-03  -.9398496E-03
  -.9604577E-03  -.8289034E-03  -.5332316E-03  -.1380318E-03   .2342108E-03
   .4651973E-03   .5026886E-03   .3746881E-03   .1623773E-03  -.5159882E-04
  -.2224880E-03  -.3475301E-03  -.4529523E-03  -.5485990E-03  -.5963545E-03
  -.5423656E-03  -.3657646E-03  -.1048402E-03   .1563278E-03   .3291475E-03
   .3576328E-03   .2409380E-03   .2377646E-04  -.2316319E-03  -.4694656E-03
  -.6625028E-03  -.8119362E-03  -.9298628E-03  -.1028468E-02  -.1108383E-02
  -.1131224E-02  -.1020689E-02  -.7165112E-03  -.2327270E-03   .3181869E-03
   .7732246E-03   .1012043E-02   .1013671E-02   .8560355E-03   .6690806E-03
   .5646968E-03   .5919845E-03   .7377345E-03   .9490117E-03   .1157766E-02
   .1288701E-02   .1282487E-02   .1122759E-02   .8318915E-03   .4619941E-03
```

```
 .7578448E-04  -.2595554E-03  -.4939353E-03  -.6127490E-03  -.6275750E-03
-.5752344E-03  -.4995743E-03  -.4264229E-03  -.3601404E-03  -.3004083E-03
-.2406227E-03  -.1628483E-03  -.3998632E-04   .1649808E-03   .4539122E-03
 .7517190E-03   .9316287E-03   .8846478E-03   .5857181E-03   .1074495E-03
-.4397852E-03  -.9508948E-03  -.1343749E-02  -.1561289E-02  -.1537678E-02
-.1227062E-02  -.6359309E-03   .1519731E-03   .9751270E-03   .1659683E-02
```

A number of tcl scripts are available to the user at the openSees web site which have been written for specific tasks. The file ReadSMDFile.tcl is a script procedure which parses a ground motion record from the PEER strong motion database by finding dt in the record header, then echoing data values to the output file. This file should be saved in the same directory as the OpenSees executable:

```
# ------------------------------------------------------------------------------------------------------------------------------------------
# READSMDFILE.tcl
# Written: MHS
# Date: July 2000
# A procedure which parses a ground motion record from the PEER strong motion database by finding dt in the record header, then
# echoing data values to the output file. Formal arguments
#     inFilename -- file which contains PEER strong motion record
#     outFilename -- file to be written in format G3 can read
#     dt -- time step determined from file header
# Assumptions
#     The header in the PEER record is, e.g., formatted as follows:
#      PACIFIC ENGINEERING AND ANALYSIS STRONG-MOTION DATA
#       IMPERIAL VALLEY 10/15/79 2319, EL CENTRO ARRAY 6, 230
#       ACCELERATION TIME HISTORY IN UNITS OF G
#       NPTS=  3930, DT= .00500 SEC
proc ReadSMDFile {inFilename outFilename dt} {
      # Pass dt by reference
      upvar $dt DT
      # Open the input file and catch the error if it can't be read
      if [catch {open $inFilename r} inFileID] {
            puts stderr "Cannot open $inFilename for reading"
      } else {
            # Open output file for writing
            set outFileID [open $outFilename w]
            # Flag indicating dt is found and that ground motion
            # values should be read -- ASSUMES dt is on last line
            # of header!!!
            set flag 0
            # Look at each line in the file
            foreach line [split [read $inFileID] \n] {
                  if {[llength $line] == 0} {
                                    # Blank line --> do nothing
                                    continue
                  } elseif {$flag == 1} {
                                    # Echo ground motion values to output file
                                    puts $outFileID $line
                  } else {
                                    # Search header lines for dt
                                    foreach word [split $line] {
                                          # Read in the time step
                                          if {$flag == 1} {
                                                set DT $word
                                                break
                                          }
                                          # Find the desired token and set the flag
```

```
                                        if {[string match $word "DT="] == 1} {
                                            set flag 1
                                        }
                                    }
                                }
                            }
                            # Close the output file
                            close $outFileID
                            # Close the input file
                            close $inFileID
                        }
}
# ------------------------------------------------------------------------------------------------------------------------------------------------
```

Once this file has been created, it can be read-in by the input file and used in the analysis procedure, where an *acceleration time-series* (page 208) is defined and used in a *UniformExcitation* (page 216) load pattern:

Series -dt $dt -filePath $fileName <-factor $cFactor>

pattern UniformExcitation $patternTag $dir -accel (TimeSeriesType arguments) <-vel0 $ver0>

```
# create load pattern
source ReadSMDFile.tcl
ReadSMDFile BM68elc.th BM68elc.acc dt
set accelSeries "Series -dt $dt -filePath BM68elc.acc -factor  1";
pattern UniformExcitation 2 1 -accel $accelSeries
```

The stiffness and mass-proportional damping factors can then be defined using the *rayleigh* (page 259) command:

rayleigh $alphaM $betaK $betaKinit $betaKcomm

There are three different stiffnesses the user can use, the current value, the initial value, or the stiffness at the last committed state (which is what is used here):

```
# set damping factors
rayleigh 0. 0. 0. [expr 2*0.02/pow([eigen 1],0.5)]
```

The various analysis components are defined as follows:

```
# create the analysis
wipeAnalysis
constraints Plain
numberer RCM
system UmfPack
test NormDispIncr 1.0e-8 10
algorithm Newton
integrator Newmark 0.5 0.25
analysis Transient
```

In a transient analysis, the analyze command also requires a time step. This time step does not have to be the same as the input ground motion. The number of time steps is equal to the total duration of the analysis (10 seconds) divided by the time step (0.02):

```
analyze [expr 10/0.02] 0.02
```

CHAPTER 33

Getting Going with OpenSees (under development)

This document will walk you through the following:

- define gravity loads

1 Define the recorders for output

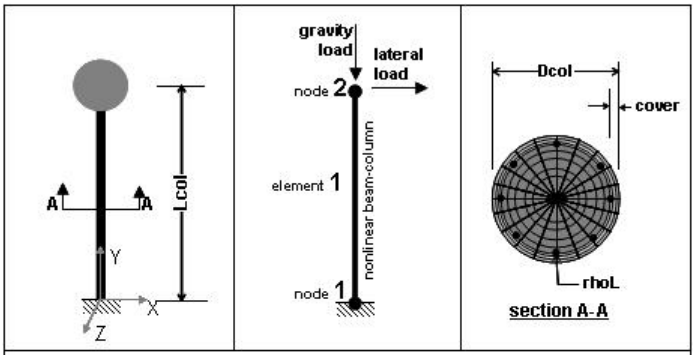
2 Define the analysis components. Three types of analyses will be set up:

- a static displacement-controlled pushover analysis
- a displacement-controlled reversed cyclic analysis
- a dynamic ground-motion-input transient analysis

In This Chapter

Problem Definition

The structural system that will be used in this document is the SDOF cantilever column shown in the figure below. The column cross section consist of a reinforced-concrete fiber section with different material properties for the confined core and the unconfined cover.



Column length, Lcol:	36 ft
Column diameter, Dcol:	6 ft
Cover thickness, cover:	6 inch
Longitudinal-steel ratio, rhoL:	1.5%
Superstructure weight, Weight:	3,000 lb
Nominal Concrete Compressive Strength, Fc:	5,500 psi
Nominal Steel Yield Strength: Fy:	60 ksi

Figure 78: Problem Definition -- Geometry

This structure will be subjected to gravity loads, as well as three types of lateral load, applied in three different analyses. These loads are shown in the figure:

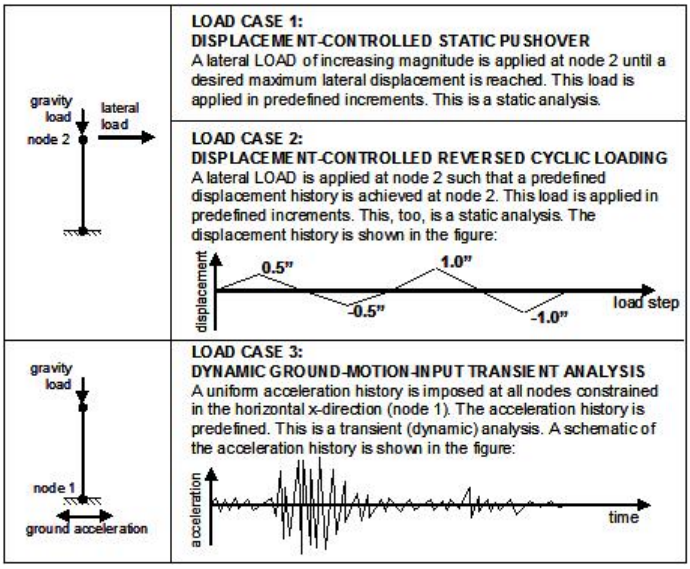


Figure 79: Problem Definition -- Loads

Model Building

This chapter describes the main steps necessary to define the physical model of a structure.

Variables and Units

The authors highly recommend that the user define variables and use these variables for the input commands. Variables can be defined using the Tcl "set" command:

```
set Radius 5;                                   # define radius of section
set Diameter [expr $Radius/2.];       # define section diameter
```

Therefore, it is recommended that the user define all material and geometric properties, as variables:

```
set fc -5000;
set fy 60000;
```

Because OpenSees does not use internal units, the user must keep track of the types of units being used. For example, the user must define all length units in either inches or meters, etc., consistently throughout.

The Tcl feature of being able to handle variables enables the user to define units as variables, and hence use them in building the model. Here is an example:

First of all, the basic units need to be defined. The OpenSees output will be in these units:

```
set in    1.;                    # define basic unit -- length
set sec   1.;                    # define basic unit -- time
set kip   1.;                    # define basic unit -- weight (or define force, but not both)
```

The basic units must be independent of each other. Once they have been defined, additional units that are made up of these basic units can be defined:

```
set ksi   [expr $kip/pow($in,2)];      # define engineering units
set psi   [expr $ksi/1000.];
set ft    [expr 12.*$in];
```

It is a good idea to define constants at the same time that units are defined:

```
set g     [expr 32.2*$ft/pow($sec,2)];      # gravitational acceleration
set PI    [expr 2*asin(1.0)];               # define constants
set U     1.e10;                                    # a really large number
set u     [expr 1/$U];                        # a really small number
```

Once the units have been defined, model variables can be defined in terms of these units and, hence, they don't all have to be defined in the basic units:

```
set fc    [expr -5500*$psi];                # CONCRETE Compressive Strength, ksi   (+Tension, -Compression)
set Ec    [expr 57000.*sqrt(-$fc/$psi)];# Concrete Elastic Modulus
set Fy    [expr 68.*$ksi];                  # STEEL yield stress
set Es    [expr 29000.*$ksi];               # modulus of steel
set epsY      [expr $Fy/$Es];               # steel yield strain
```

It is also a good idea to use variables for IDtags of materials, sections, elements, etc. This is done to ensure that the same ID tag is not used when defining the input. Also, it makes it easier in writing the input to use a variable name that makes sense. Here is an example:

```
# set up parameters for column section and element definition
set IDcore    1;                  # ID tag for core concrete
set IDcover   2;                  # ID tag for cover concrete
```

```
set IDsteel    3;                          # ID tag for steel
```

For the example structural model, the following variables need to be defined:

```
# define GEOMETRY variables
set Hcol        [expr 6.*$ft];          # column diameter
set Lcol        [expr 36*$ft];          # column length
set GrhoCol  0.015;                             # column longitudinal-steel ratio
set Weight    [expr 3000.*$kip];        # superstructure weight
set Rcol  [expr $Hcol/2];                       # COLUMN radius
set Acol  [expr $PI*pow($Rcol,2)];    # column cross-sectional area
set cover       [expr 6*$in];                   # column cover width
set G           $U;                             # Torsional stiffness Modulus
set J           1.;                             # Torsional stiffness of section
set GJ          [expr $G*$J];           # Torsional stiffness
# define COLUMN REINFORCEMENT variables
set NbCol                               20;                             # number of column longitudinal-reinforcement bars
set AsCol                               [expr $GrhoCol*$Acol];  # total steel area in column section
set AbCol                               [expr $AsCol/$NbCol];   # bar area of column longitudinal reinforcement
# define GRAVITY variables

set Mass                                [expr $Weight/$g];                              # mass of superstructure
set Mnode     [expr $Mass];                             # nodal mass for each column joint
# define DAMPING variables from $xDamp   --SDOF system, use stiffness proportional damping only
set xDamp                               0.02;           # modal damping ratio
# ------ set analysis variables
set DxPush    [expr 0.1*$in];           # Displacement increment for pushover analysis
set DmaxPush                            [expr 0.05*$Lcol]; # maximum displamcement for pushover analysis
set gamma     0.5;                      # gamma value for newmark integration
set beta        0.25;                   # beta value for newmark integration
set DtAnalysis                          [expr 0.005*$sec];              # time-step Dt for lateral analysis
set DtGround                            [expr 0.02*$sec];  # time-step Dt for input grond motion
set TmaxGround                          [expr 50.*$sec];    # maximum duration of ground-motion analysis
```

Model Builder

For a 2-D problem, you really only need three degrees of freedom at each node, the two translations in the plane and the rotation about the plane's normal:

```
model basic -ndm 2 -ndf 3
```

Nodal Coordinates & Masses, Boundary Conditions

NODAL COORDINATES AND NODAL MASSES

Once the dimensions of the problem is defined, it recommended that the user define the coordinates of the nodes, the mass associated with each node and DOF and the boundary conditions at the nodes.

The nodal coordinates are defined using the *node* (page 28) command. The number of parameters associated with this command are referenced to the model command. Nodal masses can be defined at the same time as the coordinates. In the two-dimensional problem considered here, only the x and y coordinates of each node need to be defined, and three mass parameters (two translation and one rotation of the plane) need to be defined:

```
node 1   0.   0.;                          # column base is located at the origin of the plane
node 2   0.   $Lcol                  -mass $Mnode 0. 0.;        # the column end has one translational mass in the x direction, only
```

the nodal *mass* (page 29) can also be defined using the mass command:

```
mass 2 $Mnode 0. 0.;                       # this command supersedes any previous mass definition at this node.
```

BOUNDARY CONDITIONS

The boundary conditions are defined using the *fix* (page 30) command. The tag 0 represents an unconstrained (free) degree of freedom, the tag 1 represents a constrained (fixed) DOF. For the structure under consideration, the column base is completely fixed (1-1-1) and the end is free (0-0-0). Three DOF's need to be defined here, the two translations and the rotation in the x-y plane:

```
fix 1 1 1 1;                       # fixed base
fix 2 0 0 0;                       # free end
```

Materials

Once the nodes have been defined, the next step towards defining elements is the *material* (page 108, page 35) definition. This step may not be necessary when using elastic *element* (page 146) or *sections* (page 129), as the materials are defined with the element or section.

There are two types of materials currently available in OpenSees, *uniaxial materials* (page 35) and *nDmaterials* (page 108). The different types of concrete and steel materials are among the uniaxial materials. There are three types of concrete available:

1 *Concrete01* (page 47): uniaxial Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength

2 *Concrete02* (page 73): uniaxial concrete material object with tensile strength and linear tension softening

3 *Concrete03* (page 79): uniaxial concrete material object with tensile strength and nonlinear tension softening.

Concrete02 will be used for the structure under consideration, as the tensile strength of the concrete is of interest in the elastic range, and modeling nonlinear tension softening is not considered necessary for the purpose of the example. The cover and core concrete will be modeled as different materials, using the same material type, but different stress and strain characteristics and different material tags. *Steel01* (page 43) will be used for the reinforcing steel.

Because some material characteristics are dependent on others, it is recommended that the user define the material properties using variables.

```
# Confined concrete:
set fc          [expr -5.5*$ksi];                   # CONCRETE Compressive Strength, ksi   (+Tension, -Compression)
set Ec          [expr 57*$ksi*sqrt(-$fc/$psi)];     # Concrete Elastic Modulus
set fc1C        [expr 1.26394*$fc];                 # CONFINED concrete (mander model), maximum stress
set eps1C                               [expr 2.*$fc1C/$Ec];                    # strain at maximum stress
set fc2C        $fc;                                                    # ultimate stress
set eps2C                               [expr 5*$eps1C];                # strain at ultimate stress
# Unconfined concrete:
set fc1U        $fc;                                                    # UNCONFINED concrete (todeschini parabolic model),
maximum stress
set eps1U                               -0.003;                         # strain at maximum stress
set fc2U        [expr 0.1*$fc];                     # ultimate stress
set eps2U                               -0.006;                         # strain at ultimate stress
# Concrete02 variables:
set lambda 0.1                          ;           # ratio between unloading slope at $epscu and initial slope
set ftC [expr -$fc1C/10.];              # tensile strength +tension
set ftU [expr -$fc1U/10.];              # tensile strength +tension
set Ets [expr $Ec/10.];                 # tension softening stiffness
# reinforcing steel
set Fy          [expr 68.*$ksi];        # STEEL yield stress
set Es          [expr 29000.*$ksi];     # modulus of steel
set epsY                                [expr $Fy/$Es];     # steel yield strain
```

```
set Fu          [expr 95.2*$ksi];       # ultimate stress of steel
set epsU                                0.1;                    # ultimate strain of steel
set E2 [expr ($Fu-$Fy)/($epsU-$epsY)];      # post-yield tangent stiffness
set Bs          [expr $E2/$Es];         # post-yield stiffness ratio of steel
```

To facilitate referencing the different material types, the user should set up material tags as variables:

```
# set up parameters for column section and element definition
set IDcore      1;                              # ID tag for core concrete
set IDcover     2;                              # ID tag for cover concrete
set IDsteel     3;                              # ID tag for steel
```

The materials are defined using the *uniaxialMaterial* (page 35) command:

```
uniaxialMaterial Concrete02 $IDcore             $fc1C   $eps1C  $fc2C   $eps2C $lambda $ftC $Ets;           # CORE
CONCRETE
uniaxialMaterial Concrete02 $IDcover            $fc1U   $eps1U  $fc2U   $eps2U $lambda $ftU $Ets;           #
COVER CONCRETE
uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs;          # REINFORCING STEEL
```

Element Cross Section

Some element types require that the element cross section be defined a-priori, this is done using the *section* (page 129) command. The section is used to represent force-deformation (or resultant stress-strain) relationships at beam-column and plate sample points.

While there are many types of sections available, the *fiber* (page 133) section will be used to define the cross section of the column in the structure under consideration. A fiber section has a general geometric configuration formed by subregions of simpler, regular shapes (e.g. quadrilateral, circular and triangular regions) called patches. In addition, individual or layers of reinforcement bars can be specified.The fiber section can be defined as a combination of the following:

fiber (page 133) -- a single fiber can be defined, such as a single reinforcing bar. The coordinates, associated area and material tag are prescribed with the fiber. (The coordinates are given with respect to the plane of the cross section, a coordinate transformation is later defined in the input using the transformation command)

patch (page 134) -- a patch defines an area that has a regular shape: *quadrilateral* (page 134) or *circular* (page 136). A different material can be associated with each patch.

layer (page 139, page 138) -- a layer defines a layer of reinforcement that has a regular shape: *straight* (page 138) or *circular.* (page 139) A different material can be associated with each layer.

The circular cross section of reinforced concrete will be defined using the patch and layer commands. First of all, it is important to define the variables:

```
# Notes
#   The center of the reinforcing bars are placed at the inner radius
#   The core concrete ends at the inner radius (same as reinforcing bars)
#   The reinforcing bars are all the same size
```

```
#    The center of the section is at (0,0) in the local axis system
#    Zero degrees is along section y-axis
set IDcolFlex                    2;                # ID tag for column section in flexure, before aggregating
torsion
set riCol 0.0;                   # inner radius of column section
set roCol      $Rcol;                    # outer radius of column section
set nfCoreR  8;                          # number of radial fibers in core (number of "rings")
set nfCoreT   16;                        # number of tangential fibers in core (number of "wedges")
set nfCoverR                     2;                # number of radial fibers in cover
set nfCoverT                     16;               # number of tangential fibers in cover
#    cover - cover thickness, has been defined with the geometry
#    IDcore - material tag for the core patch, has been defined with the materials
#    IDcover - material tag for the cover patches, has been defined with the materials
#    IDsteel - material tag for the reinforcing steel, has been defined with the materials
#     NbCol                          # number of column longitudinal-reinforcement bars, has been defined with the
geometry
#     AbCol                          # bar area of column longitudinal reinforcement, has been defined with the
geometry
```

The fiber cross section is defined as follows:

```
section fiberSec $IDcolFlex {
      set rc [expr $roCol-$cover];               # Core radius
      patch circ $IDcore $nfCoreT $nfCoreR 0 0 $riCol $rc 0 360;          # Define the core patch
      patch circ $IDcover $nfCoverT $nfCoverR 0 0 $rc $roCol 0 360;       # Define the cover patch
      set theta [expr 360.0/$NbCol];   # Determine angle increment between bars
      layer circ $IDsteel $NbCol $AbCol 0 0 $rc $theta 360;        # Define the reinforcing layer
 }
```

Elements and Element Connectivity

Once the element cross section has been defined, additional mechanical properties must be associated (aggregated) to it. Elastic torsion needs to be added to the column under consideration, using an *elastic* (page 35) uniaxial material:

```
set IDcolTors                    10;               # ID tag for column section in torsion
set IDcolSec  1;                         # ID tag for column section
uniaxialMaterial Elastic $IDcolTors $GJ;                 # Define torsional stiffness
section Aggregator $IDcolSec    $IDcolTors      T    -section    $IDcolFlex;       # attach torsion to flexure and create a
new section IDtag
```

The geometric *transformation* (page 200) is used to relate the local element, and section, coordinates to the global system coordinates (simple here for a 2-D problem):

```
set IDcolTrans                   1;                # ID tag for column transformation, defining element normal
geomTransf Linear $IDcolTrans;     # Linear: no second-order effects
```

The element, a *nonlinearBeamColulmn* (page 149) element, and its connectivity, are defined as follows:

```
set np          5;                               # Number of integration points
element nonlinearBeamColumn  1    1   2    $np          $IDcolSec        $IDcolTrans
```

Gravity and other Constant Loads

Gravity loads are independent of the type of lateral loading and are considered part of the structural model. These loads are first defined:

```
# apply constant gravity load (and other constant loads)
set Pdl         [expr $Weight];                 # gravity axial load per column
pattern Plain 1 Linear {
      load 2    0.0 -$Pdl 0.0
       }
```

The above defines the gravity load (ID=1) as a load in the negative y-direction at node 2 with a magnitude Pdl

And then applied:

```
# set up solution procedure
system UmfPack;                                 # solution procedure, Super-LU, how it solves system of equations
constraints Plain;                              # how it handles boundary conditions, enforce constraints through the
transformation
# set up convergence criteria
test NormDispIncr 1.0e-5    10      0;          # tolerance, max no. of iterations, and print code , 1: every iteration
algorithm Newton;                                       # use Newton's solution algorithm: updates tangent
stiffness at every iteration
numberer RCM;                                           # renumber dof's to minimize band-width
(optimization)
# set up load steppring
integrator LoadControl 0.1 1 0.1 0.1;                   # variable load-stepping: Do initial incr., desred no. of iterations
to converge, Dmax, Dmin
#  set up type of analysis, static for gravity
analysis Static
initialize
# RUN GRAVITY ANALYSIS
analyze 10
loadConst -time 0.0
```

Summary of Defining Structural Model

```
# DEFINE UNITS
set in      1.;                                 # define basic unit -- length
set sec     1.;                                 # define basic unit -- time
set kip     1.;                                 # define basic unit -- weight (or define force, but not both)
set ksi     [expr $kip/pow($in,2)];       # define engineering units
set psi     [expr $ksi/1000.];
set ft      [expr 12.*$in];
set g       [expr 32.2*$ft/pow($sec,2)];        # gravitational acceleration
set PI      [expr 2*asin(1.0)];                 # define constants
set U       1.e10;                                      # a really large number
set u       [expr 1/$U];                        # a really small number
# define GEOMETRY variables
```

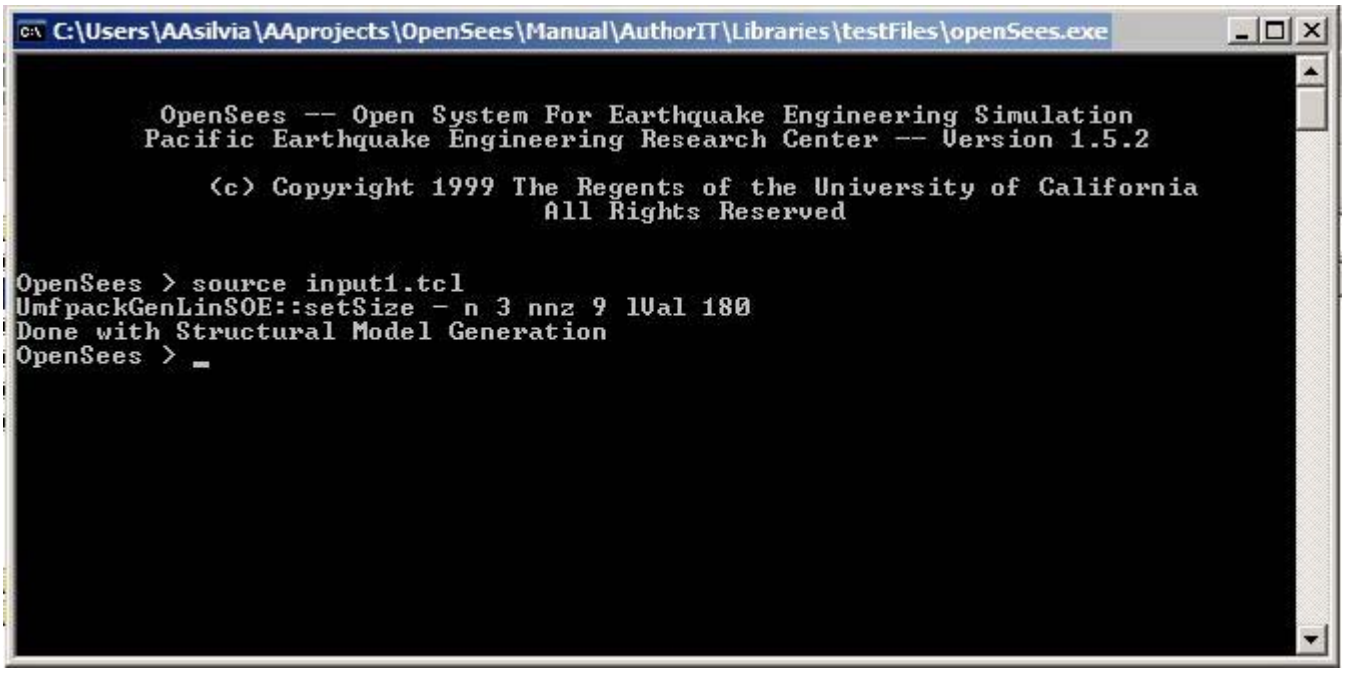
```
set Hcol      [expr 6.*$ft];          # column diameter
set Lcol      [expr 36*$ft];          # column length
set GrhoCol  0.015;                   # column longitudinal-steel ratio
set Weight    [expr 3000.*$kip];      # superstructure weight
set Rcol  [expr $Hcol/2];             # COLUMN radius
set Acol  [expr $PI*pow($Rcol,2)];    # column cross-sectional area
set cover     [expr $Hcol/15];        # column cover width
set G         $U;                     # Torsional stiffness Modulus
set J         1.;                     # Torsional stiffness of section
set GJ        [expr $G*$J];           # Torsional stiffness
# define COLUMN REINFORCEMENT variables
set NbCol                             20;                     # number of column longitudinal-reinforcement bars
set AsCol                             [expr $GrhoCol*$Acol];  # total steel area in column section
set AbCol                             [expr $AsCol/$NbCol];   # bar area of column longitudinal reinforcement
# define GRAVITY variables
set Mass                              [expr $Weight/$g];                    # mass of superstructure
set Mnode     [expr $Mass];                       # nodal mass for each column joint
# define DAMPING variables from $xDamp   --SDOF system, use stiffness proportional damping only
set xDamp                             0.02;           # modal damping ratio
# ------ set analysis variables
set DxPush    [expr 0.1*$in];         # Displacement increment for pushover analysis
set DmaxPush                          [expr 0.05*$Lcol]; # maximum displamcement for pushover analysis
set gamma     0.5;                    # gamma value for newmark integration
set beta      0.25;                   # beta value for newmark integration
set DtAnalysis                        [expr 0.005*$sec];      # time-step Dt for lateral analysis
set DtGround                          [expr 0.02*$sec];  # time-step Dt for input grond motion
set TmaxGround                        [expr 50.*$sec];   # maximum duration of ground-motion analysis
# define ModelBuilder
model basic -ndm 2 -ndf 3;            # basic: modelbuilder type, ndm= number of dimensions, ndf= #dof/node
# Nodal Coordinates and Nodal Masses
node 1   0.    0.;                    # column base is located at the origin of the plane
node 2   0.    $Lcol                  -mass $Mnode 0. 0.;     # the column end has one translational mass in the x
direction, only
# Boundary Conditions
fix 1 1 1 1;                          # fixed base
fix 2 0 0 0;                          # free end

# Confined concrete:
set fc [expr -5.5*$ksi]; # CONCRETE Compressive Strength, ksi (+Tension, -Compression)
set Ec [expr 57*$ksi*sqrt(-$fc/$psi)]; # Concrete Elastic Modulus
set fc1C [expr 1.26394*$fc]; # CONFINED concrete (mander model), maximum stress
set eps1C [expr 2.*$fc1C/$Ec]; # strain at maximum stress
set fc2C $fc; # ultimate stress
set eps2C [expr 5*$eps1C]; # strain at ultimate stress
# Unconfined concrete:
set fc1U $fc; # UNCONFINED concrete (todeschini parabolic model), maximum stress
set eps1U -0.003; # strain at maximum stress
set fc2U [expr 0.1*$fc]; # ultimate stress
set eps2U -0.006; # strain at ultimate stress
# Concrete02 variables:
set lambda 0.1                        ;       # ratio between unloading slope at $epscu and initial slope
set ftC [expr -$fc1C/10.];            # tensile strength +tension
set ftU [expr -$fc1U/10.];            # tensile strength +tension
set Ets [expr $Ec/10.];               # tension softening stiffness
# reinforcing steel
set Fy [expr 68.*$ksi]; # STEEL yield stress
set Es [expr 29000.*$ksi]; # modulus of steel
set epsY [expr $Fy/$Es]; # steel yield strain
set Fu [expr 95.2*$ksi]; # ultimate stress of steel
set epsU 0.1; # ultimate strain of steel
```

```
set E2 [expr ($Fu-$Fy)/($epsU-$epsY)]; # post-yield tangent stiffness
set Bs [expr $E2/$Es]; # post-yield stiffness ratio of steel
# set up parameters for column section and element definition
set IDcore 1; # ID tag for core concrete
set IDcover 2; # ID tag for cover concrete
set IDsteel 3; # ID tag for steel
uniaxialMaterial Concrete02 $IDcore $fc1C $eps1C $fc2C $eps2C $lambda $ftC $Ets; # CORE CONCRETE
uniaxialMaterial Concrete02 $IDcover $fc1U $eps1U $fc2U $eps2U $lambda $ftU $Ets; # COVER CONCRETE
uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs; # REINFORCING STEEL

# element cross-section
# Notes
# The center of the reinforcing bars are placed at the inner radius
# The core concrete ends at the inner radius (same as reinforcing bars)
# The reinforcing bars are all the same size
# The center of the section is at (0,0) in the local axis system
# Zero degrees is along section y-axis
set IDcolFlex 2; # ID tag for column section in flexure, before aggregating torsion
set riCol 0.0; # inner radius of column section
set roCol $Rcol; # outer radius of column section
set nfCoreR 8; # number of radial fibers in core (number of "rings")
set nfCoreT 16; # number of tangential fibers in core (number of "wedges")
set nfCoverR 2; # number of radial fibers in cover
set nfCoverT 16; # number of tangential fibers in cover
# cover - cover thickness, has been defined with the geometry
# IDcore - material tag for the core patch, has been defined with the materials
# IDcover - material tag for the cover patches, has been defined with the materials
# IDsteel - material tag for the reinforcing steel, has been defined with the materials
# NbCol # number of column longitudinal-reinforcement bars, has been defined with the geometry
# AbCol # bar area of column longitudinal reinforcement, has been defined with the geometry
section fiberSec $IDcolFlex {
set rc [expr $roCol-$cover]; # Core radius
patch circ $IDcore $nfCoreT $nfCoreR 0 0 $riCol $rc 0 360; # Define the core patch
patch circ $IDcover $nfCoverT $nfCoverR 0 0 $rc $roCol 0 360; # Define the cover patch
set theta [expr 360.0/$NbCol]; # Determine angle increment between bars
layer circ $IDsteel $NbCol $AbCol 0 0 $rc $theta 360; # Define the reinforcing layer
}

# element connectivity
set IDcolTors 10; # ID tag for column section in torsion
set IDcolSec 1; # ID tag for column section
uniaxialMaterial Elastic $IDcolTors $GJ; # Define torsional stiffness
section Aggregator $IDcolSec $IDcolTors T -section $IDcolFlex; # attach torsion to flexure and create a new section
IDtag
set IDcolTrans 1; # ID tag for column transformation, defining element normal
geomTransf Linear $IDcolTrans;      # Linear: no second-order effects
set np 5; # Number of integration points
element nonlinearBeamColumn 1 1 2 $np $IDcolSec $IDcolTrans

# apply constant gravity load (and other constant loads)
set Pdl         [expr $Weight];                 # gravity axial load per column
pattern Plain 1 Linear {
      load 2    0.0 -$Pdl 0.0
       }
# set up solution procedure
system UmfPack;                                 # solution procedure, Super-LU, how it solves system of equations
constraints Plain;                              # how it handles boundary conditions, enforce constraints through the
transformation
# set up convergence criteria
test NormDispIncr 1.0e-5    10      0;          # tolerance, max no. of iterations, and print code , 1: every iteration
```

```
algorithm Newton;                                   # use Newton's solution algorithm: updates tangent
stiffness at every iteration
numberer RCM;                                       # renumber dof's to minimize band-width
(optimization)
# set up load steppring
integrator LoadControl 0.1 1 0.1 0.1;               # variable load-stepping
#  set up type of analysis, static for gravity
analysis Static
initialize
# RUN GRAVITY ANALYSIS
analyze 10
loadConst -time 0.0

# print to screen that you are done with this step:
puts "Done with Structural Model Generation"
```

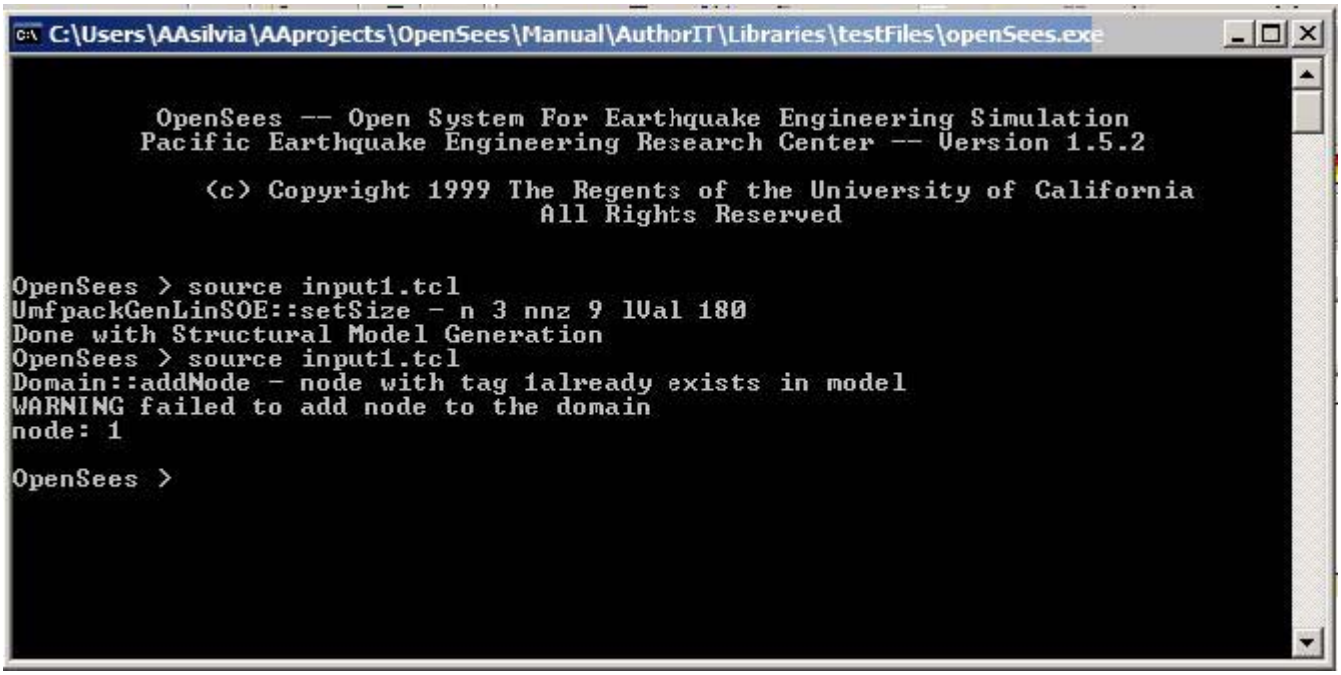
Please note the last command, it communicates to the user that all commands preceding it have been executed. The above commands can be submitted to OpenSees one-by-one, or they can be saved into a file, say input1.tcl.

Once the input file has been saved, it can be executed at the OpenSees command prompt:

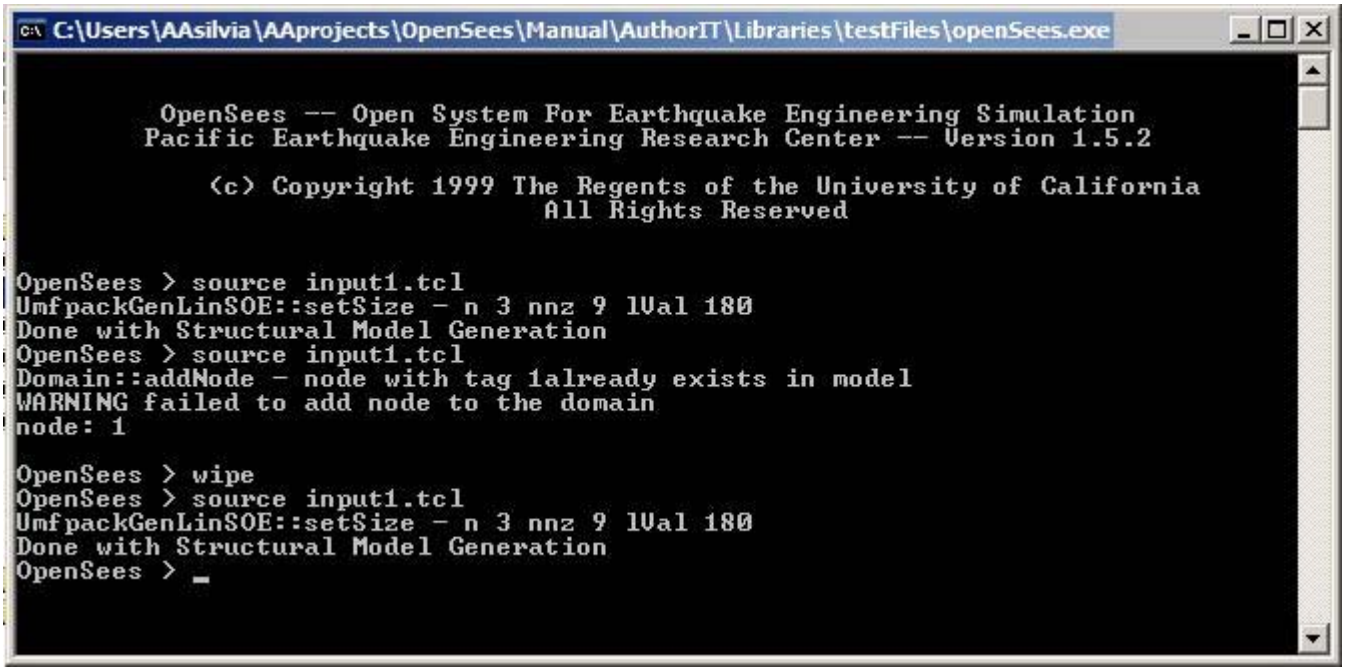


Of course, you will likely not get the clean response I got above.

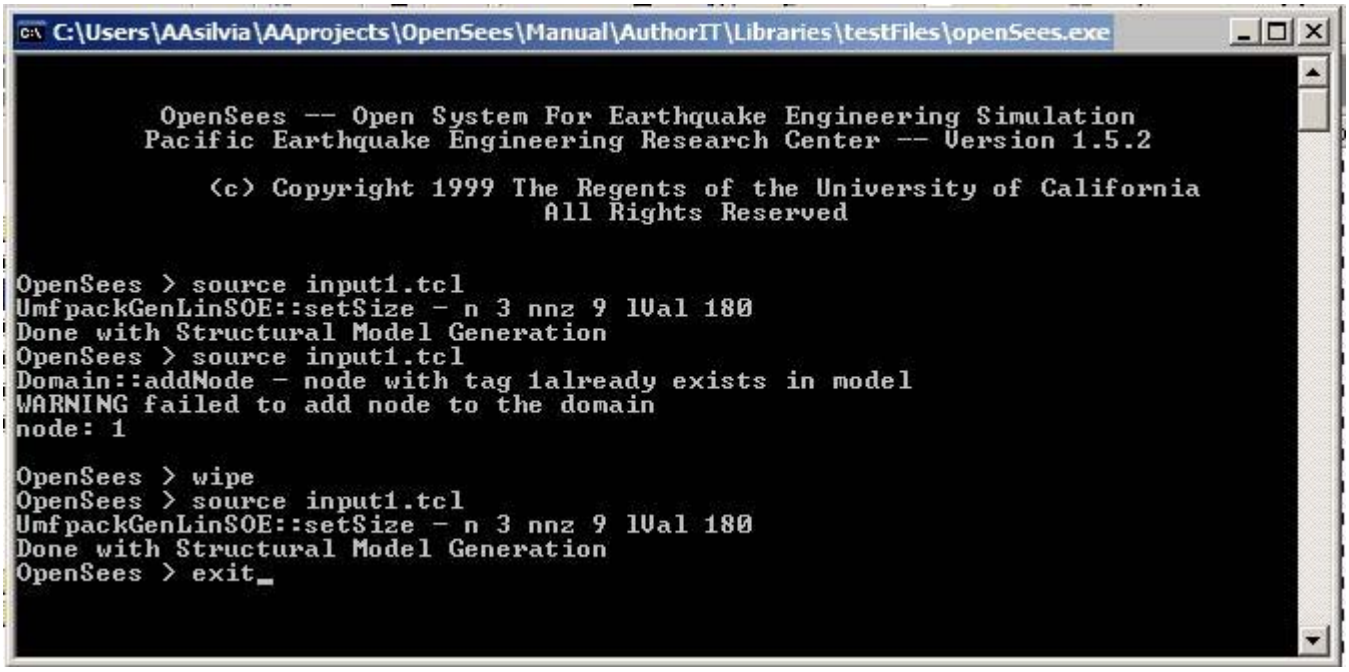
Say you make a mistake. Likely it is a simple mistake it and you go and fix it (most errors that we ALL commit are simple typing errors). To check it, you need to source the input file once more:



OpenSees does not allow you to define objects with the same IDtag more than once. To solve this problem, without exiting and re-entering OpenSees, you can use the *wipe* (page 265) command:



Once you get confirmation that all the input commands have been executed correctly, you are ready to move on to the next step. Type exit, press enter and go on.



Error-Checking Tip for Model Building

To enable error-checking, it is good practice to place markers within the input file. Markers are simple statements to be output to the screen using the *puts* (http://docsrv.sco.com:507/en/man/html.TCL/puts.TCL.html) tcl command. These markers tell the user what commands have been completed. This way, if an error occurs, the user is able to find the location of the error. Here is an example, where puts commands are placed here and there in the input1.tcl file:

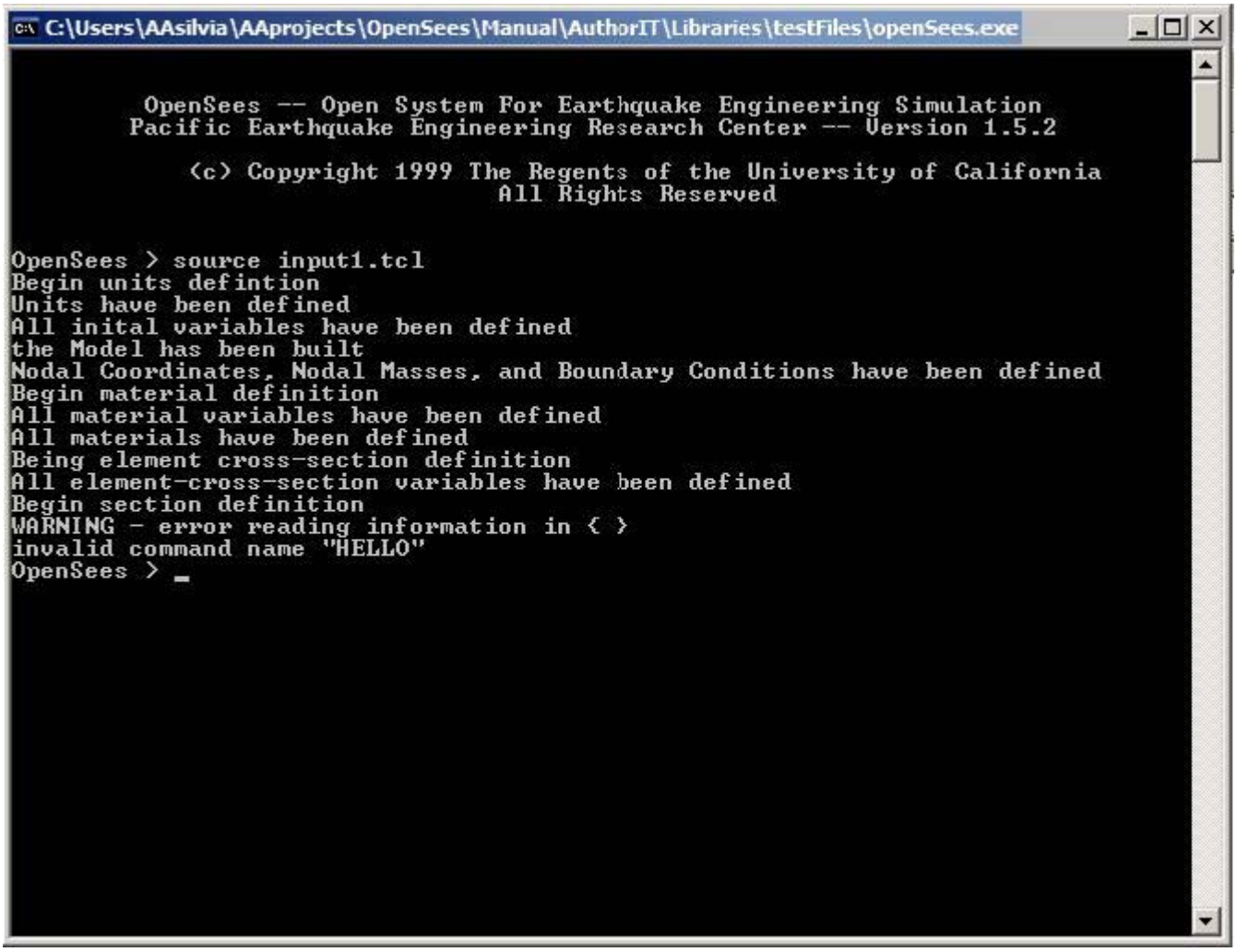
```
puts "Begin units defintion"
# DEFINE UNITS
set in     1.;                                  # define basic unit -- length
set sec    1.;                                  # define basic unit -- time
set kip    1.;                                  # define basic unit -- weight (or define force, but not both)
set ksi    [expr $kip/pow($in,2)];      # define engineering units
set psi    [expr $ksi/1000.];
set ft     [expr 12.*$in];
set g      [expr 32.2*$ft/pow($sec,2)];         # gravitational acceleration
set PI     [expr 2*asin(1.0)];                  # define constants
set U      1.e10;                                          # a really large number
set u      [expr 1/$U];                                 # a really small number
puts "Units have been defined"
# define GEOMETRY variables
set Hcol       [expr 6.*$ft];           # column diameter
set Lcol       [expr 36*$ft];           # column length
set GrhoCol  0.015;                     # column longitudinal-steel ratio
set Weight    [expr 3000.*$kip];       # superstructure weight
set Rcol  [expr $Hcol/2];                       # COLUMN radius
set Acol  [expr $PI*pow($Rcol,2)];    # column cross-sectional area
set cover      [expr $Hcol/15];                 # column cover width
set G          $U;                              # Torsional stiffness Modulus
set J          1.;                              # Torsional stiffness of section
set GJ         [expr $G*$J];            # Torsional stiffness
# define COLUMN REINFORCEMENT variables
set NbCol                               20;                             # number of column longitudinal-reinforcement bars
set AsCol                               [expr $GrhoCol*$Acol];     # total steel area in column section
set AbCol                               [expr $AsCol/$NbCol];      # bar area of column longitudinal reinforcement
# define GRAVITY variables
set Mass                                [expr $Weight/$g];                                      # mass of superstructure
set Mnode     [expr $Mass];                             # nodal mass for each column joint
# define DAMPING variables from $xDamp   --SDOF system, use stiffness proportional damping only
set xDamp                               0.02;                   # modal damping ratio
# ------ set analysis variables
set DxPush    [expr 0.1*$in];           # Displacement increment for pushover analysis
set DmaxPush                            [expr 0.05*$Lcol]; # maximum displamcement for pushover analysis
set gamma     0.5;                      # gamma value for newmark integration
set beta      0.25;                     # beta value for newmark integration
set DtAnalysis                          [expr 0.005*$sec];              # time-step Dt for lateral analysis
set DtGround                            [expr 0.02*$sec];  # time-step Dt for input grond motion
set TmaxGround                          [expr 50.*$sec];    # maximum duration of ground-motion analysis
puts "All inital variables have been defined"
# define ModelBuilder
model basic -ndm 2 -ndf 3;              # basic: modelbuilder type, ndm= number of dimensions, ndf= #dof/node
puts "the Model has been built"
# Nodal Coordinates and Nodal Masses
```

```
node 1   0.   0.;                              # column base is located at the origin of the plane
node 2   0.   $Lcol                  -mass $Mnode 0. 0.;        # the column end has one translational mass in the x
direction, only
# Boundary Conditions
fix 1 1 1 1;                         # fixed base
fix 2 0 0 0;                         # free end
puts "Nodal Coordinates, Nodal Masses, and Boundary Conditions have been defined"
puts "Begin material definition"
# Confined concrete:
set fc [expr -5.5*$ksi]; # CONCRETE Compressive Strength, ksi (+Tension, -Compression)
set Ec [expr 57*$ksi*sqrt(-$fc/$psi)]; # Concrete Elastic Modulus
set fc1C [expr 1.26394*$fc]; # CONFINED concrete (mander model), maximum stress
set eps1C [expr 2.*$fc1C/$Ec]; # strain at maximum stress
set fc2C $fc; # ultimate stress
set eps2C [expr 5*$eps1C]; # strain at ultimate stress
# Unconfined concrete:
set fc1U $fc; # UNCONFINED concrete (todeschini parabolic model), maximum stress
set eps1U -0.003; # strain at maximum stress
set fc2U [expr 0.1*$fc]; # ultimate stress
set eps2U -0.006; # strain at ultimate stress
# Concrete02 variables:
set lambda 0.1                       ;          # ratio between unloading slope at $epscu and initial slope
set ftC [expr -$fc1C/10.];           # tensile strength +tension
set ftU [expr -$fc1U/10.];           # tensile strength +tension
set Ets [expr $Ec/10.];              # tension softening stiffness
# reinforcing steel
set Fy [expr 68.*$ksi]; # STEEL yield stress
set Es [expr 29000.*$ksi]; # modulus of steel
set epsY [expr $Fy/$Es]; # steel yield strain
set Fu [expr 95.2*$ksi]; # ultimate stress of steel
set epsU 0.1; # ultimate strain of steel
set E2 [expr ($Fu-$Fy)/($epsU-$epsY)]; # post-yield tangent stiffness
set Bs [expr $E2/$Es]; # post-yield stiffness ratio of steel
# set up parameters for column section and element definition
set IDcore 1; # ID tag for core concrete
set IDcover 2; # ID tag for cover concrete
set IDsteel 3; # ID tag for steel
puts "All material variables have been defined"
uniaxialMaterial Concrete02 $IDcore $fc1C $eps1C $fc2C $eps2C $lambda $ftC $Ets; # CORE CONCRETE
uniaxialMaterial Concrete02 $IDcover $fc1U $eps1U $fc2U $eps2U $lambda $ftU $Ets; # COVER CONCRETE
uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs; # REINFORCING STEEL
puts "All materials have been defined"
puts "Being element cross-section definition"
# element cross-section
# Notes
# The center of the reinforcing bars are placed at the inner radius
# The core concrete ends at the inner radius (same as reinforcing bars)
# The reinforcing bars are all the same size
# The center of the section is at (0,0) in the local axis system
# Zero degrees is along section y-axis
set IDcolFlex 2; # ID tag for column section in flexure, before aggregating torsion
set riCol 0.0; # inner radius of column section
set roCol $Rcol; # outer radius of column section
set nfCoreR 8; # number of radial fibers in core (number of "rings")
set nfCoreT 16; # number of tangential fibers in core (number of "wedges")
set nfCoverR 2; # number of radial fibers in cover
set nfCoverT 16; # number of tangential fibers in cover
# cover - cover thickness, has been defined with the geometry
# IDcore - material tag for the core patch, has been defined with the materials
# IDcover - material tag for the cover patches, has been defined with the materials
```

```
# IDsteel - material tag for the reinforcing steel, has been defined with the materials
# NbCol # number of column longitudinal-reinforcement bars, has been defined with the geometry
# AbCol # bar area of column longitudinal reinforcement, has been defined with the geometry
puts "All element-cross-section variables have been defined"
puts "Begin section definition"
section fiberSec $IDcolFlex {
set rc [expr $roCol-$cover]; # Core radius
patch circ $IDcore $nfCoreT $nfCoreR 0 0 $riCol $rc 0 360; # Define the core patch
patch circ $IDcover $nfCoverT $nfCoverR 0 0 $rc $roCol 0 360; # Define the cover patch
puts "All patches have been defined"
set theta [expr 360.0/$NbCol]; # Determine angle increment between bars
layer circ $IDsteel $NbCol $AbCol 0 0 $rc $theta 360; # Define the reinforcing layer
puts "All layers have been defined"
}
puts "End of element-cross-section definition"
puts "Begin Element Connectivity"
# element connectivity
set IDcolTors 10; # ID tag for column section in torsion
set IDcolSec 1; # ID tag for column section
uniaxialMaterial Elastic $IDcolTors $GJ; # Define torsional stiffness
section Aggregator $IDcolSec $IDcolTors T -section $IDcolFlex; # attach torsion to flexure and create a new section
IDtag
puts "All has been aggregated"
set IDcolTrans 1; # ID tag for column transformation, defining element normal
geomTransf Linear $IDcolTrans;      # Linear: no second-order effects
set np 5; # Number of integration points
element nonlinearBeamColumn 1 1 2 $np $IDcolSec $IDcolTrans
puts "End Element defintion"
puts "Begin Gravity loads"
# apply constant gravity load (and other constant loads)
set Pdl         [expr $Weight];                 # gravity axial load per column
pattern Plain 1 Linear {
       load 2     0.0 -$Pdl 0.0
        }
puts "Gravity Load Pattern has been defined"
# set up solution procedure
system UmfPack;                                 # solution procedure, Super-LU, how it solves system of equations
constraints Plain;                              # how it handles boundary conditions, enforce constraints through the
transformation
# set up convergence criteria
test NormDispIncr 1.0e-5    10      0;          # tolerance, max no. of iterations, and print code , 1: every iteration
algorithm Newton;                                       # use Newton's solution algorithm: updates tangent
stiffness at every iteration
numberer RCM;                                           # renumber dof's to minimize band-width
(optimization)
# set up load steppring
integrator LoadControl 0.1 1 0.1 0.1;                   # variable load-stepping
#  set up type of analysis, static for gravity
analysis Static
puts "Static Analysis for Gravity and Constant loads has been defined"
initialize
# RUN GRAVITY ANALYSIS
analyze 10
loadConst -time 0.0
puts "End of Gravity Analysis"
# print to screen that you are done with this step:
puts "Done with Structural Model Generation"
```

The OpenSees control window will look like this:

When there is an error, it can be localized:



Based on the markers, the erroneous command is located within the section definition, between the two puts commands.

```
puts "Begin section definition"
section fiberSec $IDcolFlex {
set rc [expr $roCol-$cover]; # Core radius
patch circ $IDcore $nfCoreT $nfCoreR 0 0 $riCol $rc 0 360; # Define the core patch
HELLO
patch circ $IDcover $nfCoverT $nfCoverR 0 0 $rc $roCol 0 360; # Define the cover patch
puts "All patches have been defined"
set theta [expr 360.0/$NbCol]; # Determine angle increment between bars
layer circ $IDsteel $NbCol $AbCol 0 0 $rc $theta 360; # Define the reinforcing layer
puts "All layers have been defined"
}
```

Recorders for Output

The recorder commands are currently being modified and the documentation will be updated accordingly. So stay tuned..... The following, therefore, may not yet work well.....

Recorders are used to monitor the state of the model at each analysis step. They can be placed at nodes, elements, sections, fibers....

First of all, it is convenient to place all output data in a subdirectory:

```
file mkdir data
set FileName0 "data/"
```

The name $FileName0 will be appended to each filename so that the output be placed in the data subdirectory.

The first recorder of interest on the cantilever column is a recorder to measure the horizontal and vertical displacements of the column end, node 2, as well as the rotation. The *node* (page 221) recorder is used:

```
set FileName1 Node2DxDyQz;        # it is very good practice to give the output file a descriptive name
set $FileName $FileName0$FileName1
recorder Node $FileName.out disp -node 2 -dof 1 2 3;
```

The next items of interest are the element forces and deformations, using the *elemen* (page 224)t recorder. For the *nonlinearBeamColumn* (page 149) element, you really only need the forces at the element ends:

```
set FileName1 Fel1
set $FileName $FileName0$FileName1
recorder Element 1 -time -file $FileName.out localForce;          # output local element forces for element 1
```

The element deformations at the cross-section level can be recorded at any integration point. For this example, the deformation at the first integration point (the base of the column) will be recorded:

```
set FileName1 Del1sec1
set $FileName $FileName0$FileName1
recorder Element  -file  $FileName.out -ele 1 section 1 deformations
```

Recorders can also be placed anywhere on a fiber section to measure fiber stresses and strains. When more than one material may occupy the location specified (such as a steel bar at the edge of the confined-concrete core), a preferred material can be specified. The location of the recorder is specified using the local coordinate system. If no fiber is located at that coordinate, a blank file will be output (very common error).

```
set FileName1 SSel1sec1
set $FileName $FileName0$FileName1
recorder Element  -file  $FileName.out -ele 1 section 1fiber $Rcol 0. $IDcore        stressStrain;
```

CHAPTER 34

Analysis Components

CHAPTER 35

Script Utilities Library

A library of Tcl Procedures is included in this chapter. The user sould modify the procedures to meet the needs of the analysis.

In This Chapter

matTest.tcl

- ***A script for testing uniaxial materials with a single DOF truss.***

- This script requires an additional *file* (http://opensees.berkeley.edu/OpenSees/examples/pattern1.txt).

```
# matTest.tcl: SDOF truss to test uniaxial material models
# Units: kip, in
# MHS, Sept 1999
#   email: mhscott@ce.berkeley.edu
model BasicBuilder -ndm 1 -ndf 1
# Define nodes
node 1 0.0
node 2 1.0
# Fix node 1
fix 1  1
# Define uniaxialMaterial
```

```
#                       tag  f'c  epsc  f'cu  epscu
uniaxialMaterial Concrete01  1  -5.0 -0.002 -1.0 -0.004
# Define truss element with unit area
#            tag ndI ndJ  A  matTag
element truss  1   1   2  1.0    1
set dt 1.0                                                    ;# Increment between data points
set filename pattern1.txt     ;# Filename containing data points
set factor 0.006                                ;# Factor applied to data values
# Read displacement pattern from file
# Note, any pattern type can be used here: Linear, Path, Sine, etc.
pattern Plain 1 "Series -dt $dt -filePath $filename -factor $factor" {
            # Set reference displacement value
            #  node dof value
            sp   2   1   1.0
}
# Impose monotonic displacements
#pattern Plain 2 "Linear -factor $factor" {
#            sp 2 1 1.0
#}
# Record nodal displacements (same as strains since truss length is 1.0)
recorder Node truss.out disp -load -node 2 -dof 1
# Record truss force (same as stress since truss area is 1.0)
recorder Element 1 -time -file force.out force
system UmfPack
constraints Penalty 1.0e12 1.0e12
# Set increment in load factor used for integration
# Does not have to be the same as dt used to read in displacement pattern
set dl $dt
integrator LoadControl $dl 1 $dl $dl
test NormDispIncr 1.0e-6 10
algorithm Newton
numberer RCM
analysis Static
analyze 10000
```

RCcircSection.tcl

- ***# Define a procedure which generates a circular reinforced concrete section # with one layer of steel evenly distributed around the perimeter and a confined core.***

```
# Formal arguments
#    id - tag for the section that is generated by this procedure
#    ri - inner radius of the section
#    ro - overall (outer) radius of the section
#    cover - cover thickness
#    coreID - material tag for the core patch
#    coverID - material tag for the cover patches
#    steelID - material tag for the reinforcing steel
#    numBars - number of reinforcing bars around the section perimeter
#    barArea - cross-sectional area of each reinforcing bar
#    nfCoreR - number of radial divisions in the core (number of "rings")
#    nfCoreT - number of theta divisions in the core (number of "wedges")
#    nfCoverR - number of radial divisions in the cover
#    nfCoverT - number of theta divisions in the cover
#
# Notes
#    The center of the reinforcing bars are placed at the inner radius
#    The core concrete ends at the inner radius (same as reinforcing bars)
#    The reinforcing bars are all the same size
#    The center of the section is at (0,0) in the local axis system
#    Zero degrees is along section y-axis
#
proc RCcircSection {id ri ro cover coreID coverID steelID numBars barArea nfCoreR nfCoreT nfCoverR nfCoverT} {

  # Define the fiber section
  section fiberSec $id {
        # Core radius
        set rc [expr $ro-$cover]
        # Define the core patch
        patch circ $coreID $nfCoreT $nfCoreR 0 0 $ri $rc 0 360
        # Define the cover patch
```

```
        patch circ $coverID $nfCoverT $nfCoverR 0 0 $rc $ro 0 360
        if {$numBars <= 0} {
                return
        }
        # Determine angle increment between bars
        set theta [expr 360.0/$numBars]
        # Define the reinforcing layer
        layer circ $steelID $numBars $barArea 0 0 $rc $theta 360
 }
}
```

RCcircSectionFEDEAS.tcl

- ***# Define a procedure which generates a circular reinforced concrete section # with one layer of steel evenly distributed around the perimeter and a confined core.***

```
# Writes section information in FEDEAS format to the TCL file stream fedeas
#
# Formal arguments
#    id - tag for the section that is generated by this procedure
#    ri - inner radius of the section
#    ro - overall (outer) radius of the section
#    cover - cover thickness
#    coreID - material tag for the core patch
#    coverID - material tag for the cover patches
#    steelID - material tag for the reinforcing steel
#    numBars - number of reinforcing bars around the section perimeter
#    barArea - cross-sectional area of each reinforcing bar
#    nfCoreR - number of radial divisions in the core (number of "rings")
#    nfCoreT - number of theta divisions in the core (number of "wedges")
#    nfCoverR - number of radial divisions in the cover
#    nfCoverT - number of theta divisions in the cover
#    fedeas - file stream to which FEDEAS information is written
#                    Calling procedure should define a TCL file stream, e.g.
#                                set fedeas [open fedeas.out w]
 #
# Notes
```

```
#    The center of the reinforcing bars are placed at the inner radius
#    The core concrete ends at the inner radius (same as reinforcing bars)
#    The reinforcing bars are all the same size
#    The center of the section is at (0,0) in the local axis system
#    Zero degrees is along section y-axis
#    Assumes G3 material tags and FEDEAS material tags are consistent
#
proc RCcircSectionFEDEAS {id ri ro cover coreID coverID steelID numBars barArea nfCoreR nfCoreT nfCoverR
nfCoverT fedeas} {

 # Define the fiber section
 section fiberSec $id {

        puts $fedeas fsection;      # FEDEAS fsection command
        puts $fedeas $id;           # fsection id
        puts $fedeas 2,0;           # 2 patches, reference axis is geometric centroid
        # Core radius
        set rc [expr $ro-$cover]
        # Define the core patch
        patch circ $coreID $nfCoreT $nfCoreR 0 0 $ri $rc 0 360
        puts $fedeas $coreID,2,$nfCoreR,$nfCoreT,1,1;       # matID,circular,nfRad,nfAng,propIJ,propJK
        puts $fedeas 0,0;                                   # (y,z) center of patch
        puts $fedeas $ri,$rc;                               # R1,R2
        puts $fedeas 0,360;                                 # theta1,theta2
        puts $fedeas 0,0;                                   # NOT USED
        # Define the cover patch
        patch circ $coverID $nfCoverT $nfCoverR 0 0 $rc $ro 0 360
        puts $fedeas $coverID,2,$nfCoverR,$nfCoverT,1,1
        puts $fedeas 0,0
        puts $fedeas $rc,$ro
        puts $fedeas 0,360
        puts $fedeas 0,0
        if {$numBars <= 0} {
                puts $fedeas 0
                puts $fedeas ""
                return
        }
        # Determine angle increment between bars
```

```
        set theta [expr 360.0/$numBars]
        puts $fedeas 1;              # Number of layers
        # Define the reinforcing layer
        layer circ $steelID $numBars $barArea 0 0 $rc $theta 360
        puts $fedeas $steelID,2,$numBars,,$barArea;        # matID,circular,numBars,<barSize>,barArea
        puts $fedeas 0,0;                                  # (y,z) center of arc
        puts $fedeas $rc;                                  # RL
        puts $fedeas $theta,360;                           # theta1,theta2
        puts $fedeas ""
 }
}
```

RCFrameDisplay.tcl

- ***# a window showing the displaced shape***

```
        recorder display DispShape 10 10 300 300 -wipe

        # next three commmands define viewing system, all values in global coords
        vrp 288.0 150.0 0    # point on the view plane in global coord, center of local viewing system
        vup 0 1 0          # dirn defining up direction of view plane
        vpn 0 0 1          # direction of outward normal to view plane

        # next three commands define view, all values in local coord system
        prp 0 0 100              # eye location in local coord sys defined by viewing system
        viewWindow -400 400 -400 400  # view bounds uMin, uMax, vMin, vMax in local coords
        plane 0 150              # distance to front and back clipping planes from eye
        projection 0             # projection mode

        port -1 1 -1 1           # area of window that will be drawn into
        fill 1                 # fill mode
        display 1 0 10
```

MomentCurvature.tcl

- ***# A procedure for performing section analysis (only does moment-curvature, but can be easily modified to do any mode of section reponse.***

```
 # MHS
# October 2000
#
# Arguments
#       secTag -- tag identifying section to be analyzed
#       axialLoad -- axial load applied to section (negative is compression)
#       maxK -- maximum curvature reached during analysis
#       numIncr -- number of increments used to reach maxK (default 100)
#
# Sets up a recorder which writes moment-curvature results to file
# section$secTag.out ... the moment is in column 1, and curvature in column 2
proc 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

        # Create recorder
        recorder Node section$secTag.out disp -time -node 2 -dof 3

        # Define constant axial load
        pattern Plain 1 "Constant" {
                load 2 $axialLoad 0.0 0.0
        }
```

```
        # Define analysis parameters
        integrator LoadControl 0 1 0 0
        system SparseGeneral -piv;      # Overkill, but may need the pivoting!
        test NormUnbalance 1.0e-9 10
        numberer Plain
        constraints Plain
        algorithm Newton
        analysis Static

        # Do one analysis for constant axial load
        analyze 1

        # Define reference moment
        pattern Plain 2 "Linear" {
            load 2 0.0 0.0 1.0
        }

        # Compute curvature increment
        set dK [expr $maxK/$numIncr]

        # Use displacement control at node 2 for section analysis
        integrator DisplacementControl 2 3 $dK 1 $dK $dK

        # Do the section analysis
        analyze $numIncr
    }
```

ReadSMDFile.tcl

- ***A procedure which converts a PEER strong motion* database *(http://peer.berkeley.edu/smcat/) file to OpenSees format***

```
#
 # -------READSDMFILE.TCL-----------------------------------------------read gm input format
  #
  # Written: MHS
  # Date: July 2000
  #
```

```
# A procedure which parses a ground motion record from the PEER
# strong motion database by finding dt in the record header, then
# echoing data values to the output file.
#
# Formal arguments
#       inFilename -- file which contains PEER strong motion record
#       outFilename -- file to be written in format G3 can read
#       dt -- time step determined from file header
#
# Assumptions
#       The header in the PEER record is, e.g., formatted as follows:
#        PACIFIC ENGINEERING AND ANALYSIS STRONG-MOTION DATA
#         IMPERIAL VALLEY 10/15/79 2319, EL CENTRO ARRAY 6, 230
#         ACCELERATION TIME HISTORY IN UNITS OF G
#         NPTS=  3930, DT= .00500 SEC
proc ReadSMDFile {inFilename outFilename dt} {
        # Pass dt by reference
        upvar $dt DT

        # Open the input file and catch the error if it can't be read
        if [catch {open $inFilename r} inFileID] {
                puts stderr "Cannot open $inFilename for reading"
        } else {
                # Open output file for writing
                set outFileID [open $outFilename w]

                # Flag indicating dt is found and that ground motion
                # values should be read -- ASSUMES dt is on last line
                # of header!!!
                set flag 0

                # Look at each line in the file
                foreach line [split [read $inFileID] \n] {

                        if {[llength $line] == 0} {
                                # Blank line --> do nothing
                                continue
                        } elseif {$flag == 1} {
```

```
                    # Echo ground motion values to output file
                    puts $outFileID $line
                } else {
                    # Search header lines for dt
                    foreach word [split $line] {
                        # Read in the time step
                        if {$flag == 1} {
                            set DT $word
                            break
                        }
                        # Find the desired token and set the flag
                        if {[string match $word "DT="] == 1} {
                            set flag 1
                        }
                    }
                }
            }
            # Close the output file
            close $outFileID
            # Close the input file
            close $inFileID
        }
    }
```

RotSpring2D

➢ ***# Procedure which creates a rotational spring for a planar problem***

```
# rotSpring2D.tcl
# SETS A MULTIPOINT CONSTRAINT ON THE TRANSLATIONAL DEGREES OF FREEDOM,
# SO DO NOT USE THIS PROCEDURE IF THERE ARE TRANSLATIONAL ZEROLENGTH
# ELEMENTS ALSO BEING USED BETWEEN THESE TWO NODES
#
# Written: MHS
# Date: Jan 2000
#
# Formal arguments
#           eleID - unique element ID for this zero length rotational spring
```

```
#           nodeR - node ID which will be retained by the multi-point constraint
#           nodeC - node ID which will be constrained by the multi-point constraint
#           matID - material ID which represents the moment-rotation relationship
#                       for the spring
proc rotSpring2D {eleID nodeR nodeC matID} {
        # Create the zero length element
        element zeroLength $eleID $nodeR $nodeC -mat $matID -dir 6

        # Constrain the translational DOF with a multi-point constraint
        #       retained constrained DOF_1 DOF_2 ... DOF_n
        equalDOF    $nodeR    $nodeC    1    2
}
```

StFramePZLdisplay.tcl

- ***# a window to plot the nodal displacements versus load***

```
if {$displayMode == "displayON"} {

    recorder plot StFramePZL1.out Node4Xdisp 10 400 300 300 -columns 3 1

    # a window showing the displaced shape
    recorder display g3 10 10 300 300 -wipe
    prp 288.0 150.0 100.0
    vrp 288.0 150.0 0
    vup 0 1 0
    vpn 0 0 1
    viewWindow -400 400 -400 400
    plane 0 150
    port -1 1 -1 1
    projection 0
    fill 1
    display 2 0 10
}
```

Wsection.tcl

- ***Procedure for creating a wide flange steel fiber section***

```
# Wsection.tcl: tcl procedure for creating a wide flange steel fiber section
# written: Remo M. de Souza
# date: 06/99
# modified: 08/99  (according to the new general modelbuilder)
```

```
  # input parameters
  # secID - section ID number
  # matID - material ID number
  # d  = nominal depth
  # tw = web thickness
  # bf = flange width
  # tf = flange thickness
  # nfdw = number of fibers along web depth
  # nftw = number of fibers along web thickness
  # nfbf = number of fibers along flange width
  # nftf = number of fibers along flange thickness
proc Wsection { secID matID d tw bf tf nfdw nftw nfbf nftf} {
    set dw [expr $d - 2 * $tf]
  set y1 [expr -$d/2]
  set y2 [expr -$dw/2]
  set y3 [expr  $dw/2]
  set y4 [expr  $d/2]
    set z1 [expr -$bf/2]
  set z2 [expr -$tw/2]
  set z3 [expr  $tw/2]
  set z4 [expr  $bf/2]
    #
  section fiberSec  $secID  {
    #                 nfIJ  nfJK    yI  zI    yJ  zJ    yK  zK    yL  zL
    patch quadr  $matID  $nfbf $nftf   $y1 $z4   $y1 $z1   $y2 $z1   $y2 $z4
    patch quadr  $matID  $nftw $nfdw   $y2 $z3   $y2 $z2   $y3 $z2   $y3 $z3
    patch quadr  $matID  $nfbf $nftf   $y3 $z4   $y3 $z1   $y4 $z1   $y4 $z4
  }
}
```

RigidFrame3Ddisplay.tcl

- ***# a window showing the displaced shape***

```
            set displayType "PLAN"
            #set displayType "PERSPECTIVE"


            # a window showing the displaced shape
```

```
recorder display g3 10 10 300 300 -wipe

if {$displayType == "PERSPECTIVE"} {
        prp -7500 -5000 50000
        vrp 0 -500 250
        vup 0 0 1
        vpn 0 -1 0
        viewWindow -200 400 -300 300
}

if {$displayType == "PLAN"} {
        prp 0 0 1000
        vrp 0 0 0
        vup 0 -1 0
        vpn 0 0 -1
        viewWindow -200 200 -200 200
}

plane 0 1e12
port -1 1 -1 1
projection 1
fill 0
display 1 0 10
```

Units&Constants.tcl

- ***Procedure to define units and constants***

```
# ----Units&Constants.tcl------------------------------------------------------
set in 1.;                              # define basic units
set sec 1.;
set kip 1.;
set ksi [expr $kip/pow($in,2)];                 # define dependent units
set psi [expr $ksi/1000.];
set ft [expr 12.*$in];
set lb [expr $kip/1000];
set pcf  [expr $lb/pow($ft,3)];
set ksi  [expr $kip/pow($in,2)];
```

```
set psi  [expr $ksi/1000.];
set cm  [expr $in/2.54];              # define metric units
set meter [expr 100.*$cm];
set MPa [expr 145*$psi];
set PI [expr 2*asin(1.0)];            # define constants
set g [expr 32.2*$ft/pow($sec,2)];
set U     1.e10;              # a really large number
set u     [expr 1/$U];        # a really small number
```

MatlabOutput.tcl

- ***# script to generate .m file to be read by matlab***

```
# -------------MatlabOutput.tcl-------------------------------------------------------------------
set Xframe 1;                 # this parameter would be passed in
set fDir "Data/";
file mkdir $fDir;                     # create directory
set outFileID [open $fDir/DataFrame$Xframe.m w];  # Open output file for writing
puts $outFileID   "Xframe($Xframe) = $Xframe;";    # frame ID
puts $outFileID   "Hcol($Xframe) = $Hcol;";                  # column diameter
puts $outFileID   "Lcol($Xframe) = $Lcol;";                  # column length
puts $outFileID   "Lbeam($Xframe) = $Lbeam;";                # beam length
puts $outFileID    "Hbeam($Xframe) = $Hbeam;";   # beam depth
puts $outFileID    "Bbeam($Xframe) = $Bbeam;";   # beam width
puts $outFileID   "Weight($Xframe) = $Weight;"     ;          # superstructure weight
close $outFileID
```

genPlaneFrame.tcl

- ***# Define a procedure which will generate nodes and elements for a plane frame having absolute column line locations in the list***

```
        #'columnLine',  absolute girder line locations in the list 'girderLine',
        # section IDs for the columns and girders, 'columnID' and 'girderID', and
        # 'nIntPt' integration points along every member.
        #
        # Notes: automatically fixes the base nodes
        #       only geneartes nonlinearBeamColumn elements
```

```
#       allows columns and girders to be spaced arbitrarily
#       does not add nodal masses or loads, but can be extended to do so
#       starts node numbering at 1
#
# Formal arguments
#    columnLine - a list of column line locations
#    The actual argument would be defined as so,  set columns {0 120 240 360}
#    girderLine - a list of grider line locations
#       The actual argument would be defined as so,  set girders {180 300 420}
#    columnID - an integer representing the section ID (tag) for  the columns in the frame
#    girderID - an integer representing the section ID (tag) for the girders in the frame
#    nIntPt - an integer representing the number of integration points along each member in the frame
proc genPlaneFrame {columnLine girderLine columnID girderID nIntPt} {
    set n 1;   # Node number counter
     geomTransf Linear 1;   # Geometric transformation for all elements
    # For each column line
    foreach xLoc $columnLine {
 node $n $xLoc 0
# Fix the base node
 fix $n 1 1 1
 incr n 1
 # For each girder line
foreach yLoc $girderLine {
        node $n $xLoc $yLoc
        incr n 1
}
    }

    # Useful variables
    set numCol [llength $columnLine]
    set numGir [llength $girderLine]
    set e 1;   # Element number counter
    # For each column line
    for {set i 1} {$i <= $numCol} {incr i 1} {
```

```
            # Node number at the base of this column line
            set bottom [expr ($i-1)*$numGir + $i]
            # Node number at the top of this column line
            set top [expr $i*$numGir + $i]
            # Travel up this column line creating elements
            for {set j $bottom} {$j <= [expr $top-1]} {incr j 1} {
                element nonlinearBeamColumn $e $j [expr $j+1] $nIntPt $columnID 1
                incr e 1
            }
        }

        # Difference in node numbers I and J for any girder in the frame
        set delta [expr $numGir+1]

        # For each girder line
        for {set j 1} {$j <= $numGir} {incr j 1} {
            # Node number at the left end of this girder line
            set left [expr $j+1]
            # Node number at the right end of this girder line
            set right [expr ($numCol-1)*$numGir + $numCol + $j]
            # Travel across this girder line creating elements
            for {set k $left} {$k < $right} {incr k $delta} {
                element nonlinearBeamColumn $e $k [expr $k+$delta] $nIntPt $columnID 1
                incr e 1
            }
        }

    }; #end proc
```

CHAPTER 36

References

API(1993). Recommended Practice for Planning, Design, and Constructing Fixed Offshore Platforms. API RP 2A - WSD, 20th ed., American Petroleum Institute.

Boulanger, R. W. (2003). The PySimple1 Material. http://opensees.berkeley.edu.

Cook, R. D., D. S. Malkus, M. E. Plesha. "Concepts and Applications of Finite Element Analysis." John Wiley & Sons, 1989.

Crisfield, M. A. "Non-linear Finite Element Analysis of Solids and Structures." John Wiley & Sons, vol. 1, 1991

Elgamal, A., Lai, T., Yang, Z. and He, L. (2001). "Dynamic Soil Properties, Seismic Downhole Arrays and Applications in Practice," State-of-the-art paper, Proc., 4th Intl. Conf. on Recent Advances in Geote. E.Q. Engrg. Soil Dyn. March 26-31, San Diego, CA, S. Prakash (Ed.).

Elgamal, A., Yang, Z. and Parra, E. (2002). "Computational Modeling of Cyclic Mobility and Post-Liquefaction Site Response," Soil Dyn. Earthquake Engrg., 22(4), 259-271.

Elgamal, A., Yang, Z., Parra, E. and Ragheb, A. (2003). "Modeling of Cyclic Mobility in Saturated Cohesionless Soils," Int. J. Plasticity, 19(6), 883-905.

Georgiadis, M. (1983). “Development of p-y curves for layered soils.” Proc., Geotechnical Practice in Offshore Engineering, ASCE, pp. 536-545.

Matlock, H. (1970). ?Correlations of design of laterally loaded piles in soft clay.? Proc. Offshore Technology Conference, Houston, TX, Vol 1, No.1204, pp. 577-594.

McKenna, F. and Fenves, G. (2001). "The OpenSees Command Language Manual: version 1.2," Pacific Earthquake Engineering Center, Univ. of Calif., Berkeley. (http://opensees.berkeley.edu).

Mosher, R. L., (1984) ?Load Transfer Criteria for Numerical Analysis of Axially Loaded Piles in Sand,? US Army Engineering Waterways Experimental Station, Automatic Data Processing Center, Vicksburg, Mississippi, January.

Parra, E. (1996). "Numerical Modeling of Liquefaction and Lateral Ground Deformation Including Cyclic Mobility and Dilation Response in Soil Systems," Ph.D. Thesis, Dept. of Civil Engineering, Rensselaer Polytechnic Institute, Troy, NY.

Reese, L.C., and O?Neill, M. W., (1987) ?Drilled Shafts: Construction Procedures and Design Methods,? Report No. FHWA-HI-88-042, U.S. Department of Transportation, Federal Highway Administration, Office of Implementation, McLean, Virginia.

Vijayvergiya, V.N. (1977) ?Load-Movement Characteristics of Piles,? Proceedings, Ports 77 Conference, American Society of Civil Engineers, Long Beach, CA, March.

Yang, Z. (2000). "Numerical Modeling of Earthquake Site Response Including Dilation and Liquefaction," Ph.D. Thesis, Dept. of Civil Engineering and Engineering Mechanics, Columbia University, NY, New York.

Yang, Z. and Elgamal, A. (2002). "Influence of Permeability on Liquefaction-Induced Shear Deformation," J. Engrg. Mech., ASCE, 128(7), 720-729.

Yang, Z., Elgamal, A. and Parra, E. (2003). "A Computational Model for Liquefaction and Associated Shear Deformation," J. Geotechnical and Geoenvironmental Engineering, ASCE, December (in press).

Index

M

N

O

P

Q