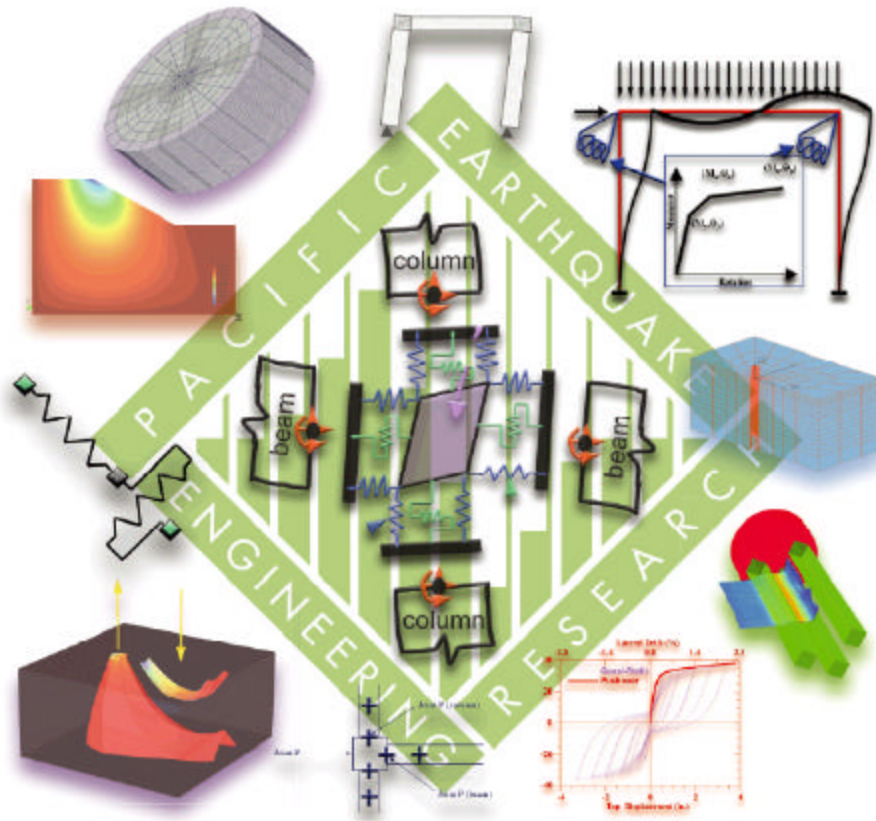


Open System for Earthquake Engineering Simulation (OpenSees)

Command Language Manual



Silvia Mazzoni, Frank McKenna, Michael H. Scott,

Gregory L. Fenves, Boris Jeremic

Contents

Introduction	9
Notation	9
Copyright	11
Tcl Basics	11
Tcl Commands	12
OpenSees Interpreter	15
Open System for Earthquake Engineering Simulation	16
ModelBuilder Object	17
Domain Object	18
Analysis Object	19
Recorder Object	20
Model Building Commands	21
Model Command	21
Basic Model Builder	21
build Command	22
Node Command	22
Mass Command	23
Constraints	24
Single-Point Constraints	24
Fix Command	24
fixX Command	25
fixY Command	25
fixZ Command	26
Multi-Point Constraints	27
equalDOF Command	27
rigidDiaphragm Command	27
rigidLink Command	28
uniaxialMaterial Command	29

Elastic Material	29
Elastic-Perfectly Plastic Material	30
Elastic-Perfectly Plastic Gap Material	31
Parallel Material	32
Series Material	33
Hardening Material	35
Steel01 Material	36
Concrete01 Material	37
Elastic-No Tension Material	38
Hysteretic Material	39
Viscous Material	40
Fedeas Materials	41
Concrete02 Material	41
Concrete03 Material	42
Steel02 Material	44
Bond01 Material	44
Bond02 Material	45

nDMaterial Command 46

Elastic Isotropic Material	46
J2 Plasticity Material	47
Plane Stress Material	47
Plate Fiber Material	48
Template Elasto-Plastic Material	48
Yield Surface	49
Potential Surface	49
Evolution Law	50
EPState	51

section Command 52

Elastic Section	53
Uniaxial Section	54
Fiber Section	55
Fiber Command	56
Quadrilateral Patch Command	57
Circular Patch Command	59
Straight Layer Command	60
Circular Layer Command	61
Section Aggregator	62
Elastic Membrane Plate Section	65
Plate Fiber Section	65
Bidirectional Section	66

element Command 67

Truss Element	67
Corotational Truss Element	68
Elastic Beam Column Element	69
NonLinear Beam-Column Elements	70
Nonlinear Beam Column Element	70
Beam With Hinges Element	71
Displacement-Based Beam-Column Element	72

Zero-Length Elements	73
Zero-Length Element	73
Zero-Length Section Element	74
Quadrilateral Elements	75
Quad Element	75
Shell Element	76
Bbar Plane Strain Quadrilateral Element	76
Enhanced Strain Quadrilateral Element	77
Brick Elements	77
Standard Brick Element	77
Bbar Brick Element	79
Eight Node Brick Element	80
Twenty Node Brick Element	81
u-p-U element	82
block Command	84
block2D Command	85
block3D Command	87
Geometric Transformation Commands	89
Linear Transformation	89
P-Delta Transformation	91
Corotational Transformation	92
recorder Command	94
Node Recorder	94
MaxNodeDisp Recorder	95
Drift Recorder	96
Element Recorder	96
Display Recorder	99
Plot Recorder	99
playback Command	100
Time Series	101
Constant Time Series	102
Linear Time Series	102
Rectangular Time Series	103
Sine Time Series	104
Path Time Series	105
pattern Command	106
plain Pattern	106
load Command	107
sp Command	108
eleLoad Command	108
UniformExcitation Pattern	108
MultipleSupport Pattern	109
groundMotion Command	109

imposedMotion Command	111
analysis Command	112
Static Analysis.....	112
Transient Analysis	113
VariableTransient Analysis.....	113
constraints Command	115
Plain Constraints	116
Penalty Method	116
Lagrange Multipliers	117
Transformation Method.....	117
integrator Command	119
Load Control	120
Displacement Control	121
Minimum Unbalanced Displacement Norm	122
Arc-Length Control	122
Newmark Method	123
Hilbert-Hughes-Taylor	124
algorithm Command	126
Linear Algorithm	126
Newton Algorithm	126
Newton with Line Search Algorithm	127
Modified Newton Algorithm.....	127
Krylov-Newton Algorithm.....	128
BFGS Algorithm	128
Broyden Algorithm	128
test Command	129
Norm Unbalance Test.....	129
Norm Displacement Increment T	130
Energy Increment Test.....	131
numberer Command	132
Plain Numberer	132
RCM Numberer	133
system Command	134

BandGeneral SOE	135
BandSPD SOE	135
ProfileSPD SOE	135
SparseGeneral SOE	135
UmfPack SOE	136
SparseSPD SOE	136

analyze Command	137
------------------------	------------

eigen Command	138
----------------------	------------

dataBase Commands	139
--------------------------	------------

FileDatastore Command	139
save Command	139
restore Command	140

Miscellaneous Commands	141
-------------------------------	------------

print Command	141
reset Command	142
wipe Command	142
wipeAnalysis Command	142
loadConst Command	143
getTime Command	143
nodeDisp Command	143
video Command	144
play Command	144

How To....	145
-------------------	------------

...Run OpenSees	146
...Define Units & Constants	148
...Generate Matlab Commands	149
...Define Tcl Procedure	149
...Read External files	151
Building The Model	152
...Define Variables and Parameters	152
...Build Model and Define Nodes	154
...Build Model and Define Nodes using Variables	155
...Define Materials	156
...Define Elements	157
Defining Output	158
...Define Analysis-Output Generation	158
...Define Data-Plot During Analysis	159
Gravity Loads	159
...Define Gravity Loads	159
...Run Gravity Analysis	160
Static Analysis	160
...Define Static Pushover Analysis	160
...Run Static Pushover Analysis	161

Dynamic Analysis	162
...Define Dynamic Ground-Motion Analysis	162
...Run Dynamic Ground-Motion Analysis.....	163
...Combine Input-File Components.....	163
...Run Parameter Study	164
...Run Moment-Curvature Analysis on Section.....	165
...Determine Natural Period & Frequency	167

Questions/Things missing	169
---------------------------------	------------

Index	171
--------------	------------

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.....	9
Copyright	11
Tcl Basics.....	11
OpenSees Interpreter.....	15

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 22) 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 21)): 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 21)).

Copyright

Copyright © 1999,2000 The Regents of the University of California. All Rights Reserved.

Permission to use, copy, modify, and distribute this software and its documentation for educational, research and non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the above copyright notice, this paragraph and the following three paragraphs appear in all copies.

Permission to incorporate this software into commercial products may be obtained by contacting the University of California. [Bill Hoskins Office of Technology Licensing, 2150 Shattuck Avenue #150 Berkeley, CA 94720-1620, (510) 643-7201]

This software program and documentation are copyrighted by The Regents of the University of California. The software program and documentation are supplied "as is", without any accompanying services from The Regents. The Regents does not warrant that the operation of the program will be uninterrupted or error-free. The end-user understands that the program was developed for research purposes and is advised not to rely exclusively on the program for any reason.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Tcl Basics

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

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

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

Help

Tcl/Tk Primer (see <http://dev.scriptics.com/scripting/primer.html>)
(<http://dev.scriptics.com/scripting/primer.html>)

Practical Programming in Tcl and Tk, Brent B. Welch, Prentice Hall.

Tcl Commands

Tcl scripts are made up of commands separated by new lines or ;

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
```

bla bla bla, this needs to be more clear.....

Take material from the scriptics page

Include the following commands:

proc

expr

file-management commands

for loops commands

if commands




Example Tcl Commands

arithmetic	procedure	for & foreach functions
<pre>>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 ></pre>	<pre>>proc sum {a b} { return [expr \$a + \$b] } >sum 2 3 5 >set c [sum 2 3] 5 ></pre>	<pre>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 ></pre>
file manipulation	procedure & if statement	
<pre>>set fileId [open tmp w] anumber >puts \$fileId "hello" >close \$fileId >type tmp hello > >source Example1.tcl</pre>	<pre>>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 ></pre>	

OpenSees Interpreter

The main abstractions of OpenSees will be explained using the OpenSees interpreter. The interpreter is an extension of the *Tcl* (page 12) 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

<http://opensees.berkeley.edu/OpenSees/OpenSeesExamples.pdf> (see <http://opensees.berkeley.edu/OpenSees/OpenSeesExamples.pdf>)

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

CHAPTER 2

Open System for Earthquake Engineering Simulation

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.

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 in the figure:

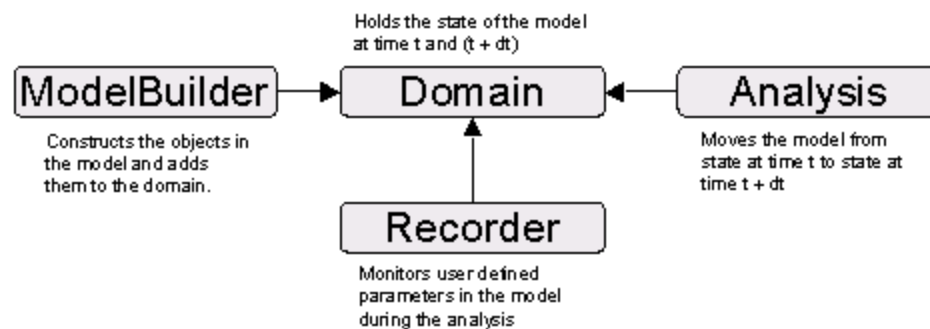


Figure 1: Principal OpenSees Objects











In This Chapter

ModelBuilder Object.....	17
Domain Object.....	18
Analysis Object.....	19
Recorder Object.....	20

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.

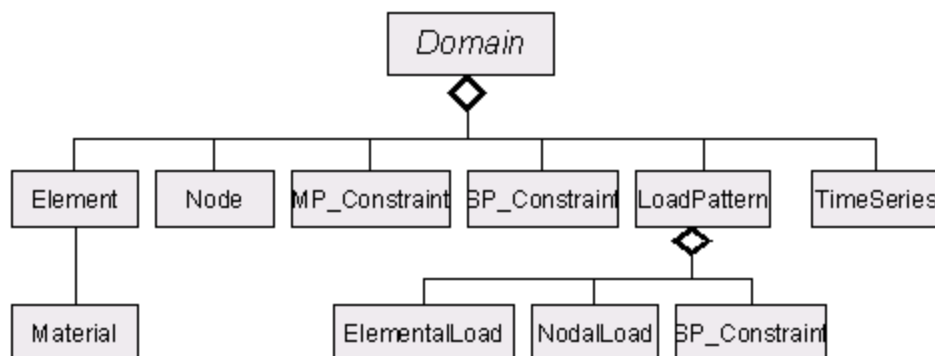
The ModelBuilder is the object in the program responsible for building the following objects in the model and adding them to the domain:

-  **Node** (page 22)
-  **Mass** (page 23)
-  **Material** (page 46, page 29)
-  **Section** (page 52)
-  **Element** (page 67)
-  **LoadPattern** (page 106)
-  **TimeSeries** (page 101)
-  **Transformation** (page 89)
-  **Block** (page 84)
-  **Constraint** (page 115)

Domain Object

The Domain object is responsible for storing the objects created by the *ModelBuilder* (page 17) object and for providing the *Analysis* (page 19) and *Recorder* (page 20) objects access to these objects.

Figure 2: Domain Object



Analysis Object

The Analysis object is responsible for performing the analysis. This may vary from a simple *static* (page 112) linear analysis to a *transient* (page 113, page 113) non-linear analysis. In OpenSees each Analysis object is composed of several component objects, which define how the analysis is performed. The component classes consist of the following:








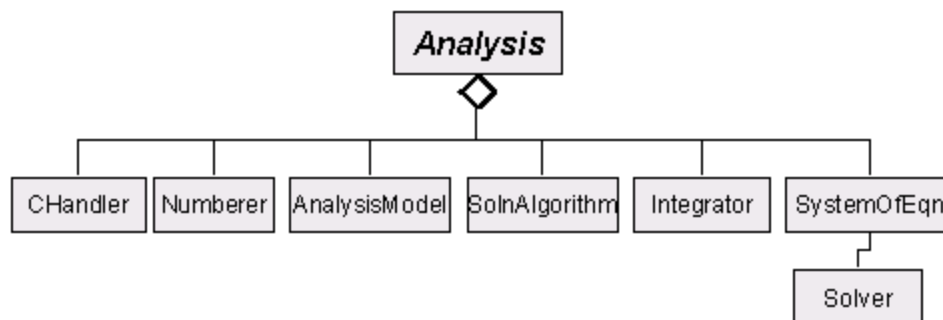
-  **ConstraintHandler** (page 115)
-  **DOF_Numberer** (page 132)
-  **AnalysisModel** (page 112)
-  **SolutionAlgorithm** (page 126)
-  **Integrator** (page 119)
-  **SystemOfEqn** (page 134)
-  **Solver** (page 134)




Figure 3: Analysis 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 94) 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.

CHAPTER 3

Model Building Commands

These commands are used to set up the physical model.

For an example of these commands, refer to the *Model Building Example* (page 154)

In This Chapter

Model Command	21
Node Command	22
Mass Command	23

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 154)

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 22), *Masses* (page 23), *Materials* (page 46, page 29), *Sections* (page 52), *Elements* (page 67), *LoadPatterns* (page 106), *TimeSeries* (page 101), *Transformations* (page 89), *Blocks* (page 84) and *Constraints* (page 115). 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 154)

build Command

This command is used to invoke build() (????) on the *ModelBuilder* (page 17) object.

build

This command has no effect a *BasicBuilder* (page 21) object, but will on other types of *ModelBuilder* (page 17) objects.

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 (<i>ndm</i> (page 21) arguments)
\$MassValues	nodal mass corresponding to each DOF (<i>ndf</i> (page 21) 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 154)

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 (<i>ndf</i> (page 21) 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 154)

CHAPTER 4

Constraints

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.....	24
Multi-Point Constraints	27

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). <i>ndf</i> (page 17) values are specified, corresponding to the <i>ndf</i> 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 154)

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). <i>ndf</i> (page 17) values are specified, corresponding to the <i>ndf</i> degrees-of-freedom. The two constraint types are: <div style="margin-left: 20px;"> 0 unconstrained 1 constrained </div>
\$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). <i>ndf</i> values are specified, corresponding to the <i>ndf</i> (page 21) 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). <i>ndf</i> values are specified, corresponding to the <i>ndf</i> (page 21) 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 <i>ndf</i> (page 21), the number of nodal degrees-of-freedom.

EXAMPLE:

```
equalDOF 2 3 1 3 5; # impose the traslational displacements in x and y 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 nodes that are to be constrained to the master node
\$slaveNodeTag2 ...

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 21) = 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 5

unialMaterial Command

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

The valid queries to any uniaxial material when creating an *ElementRecorder* (page 96) are 'strain,' 'stress,' and 'tangent.'

In This Chapter

Elastic Material	29
Elastic-Perfectly Plastic Material	30
Elastic-Perfectly Plastic Gap Material	31
Parallel Material	32
Series Material	33
Hardening Material	35
Steel01 Material	36
Concrete01 Material	37
Elastic-No Tension Material	38
Hysteretic Material	39
Viscous Material	40
Fedeas Materials	41

Elastic Material

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

```
unialMaterial Elastic $matTag $E <$eta>
```

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

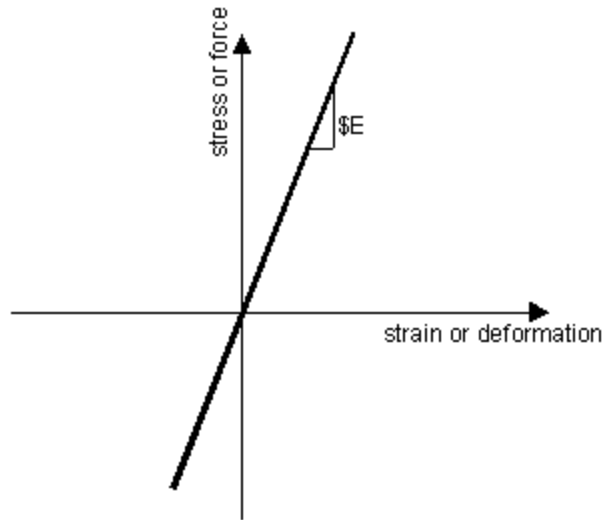


Figure 4: Elastic Material

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)

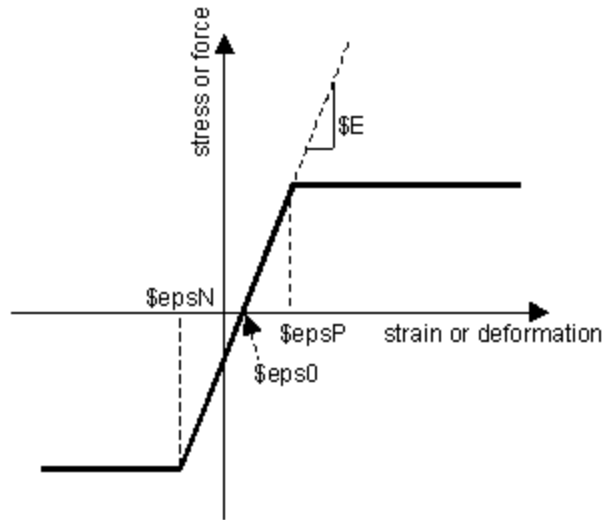


Figure 5: Elastic-Perfectly Plastic Material

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.

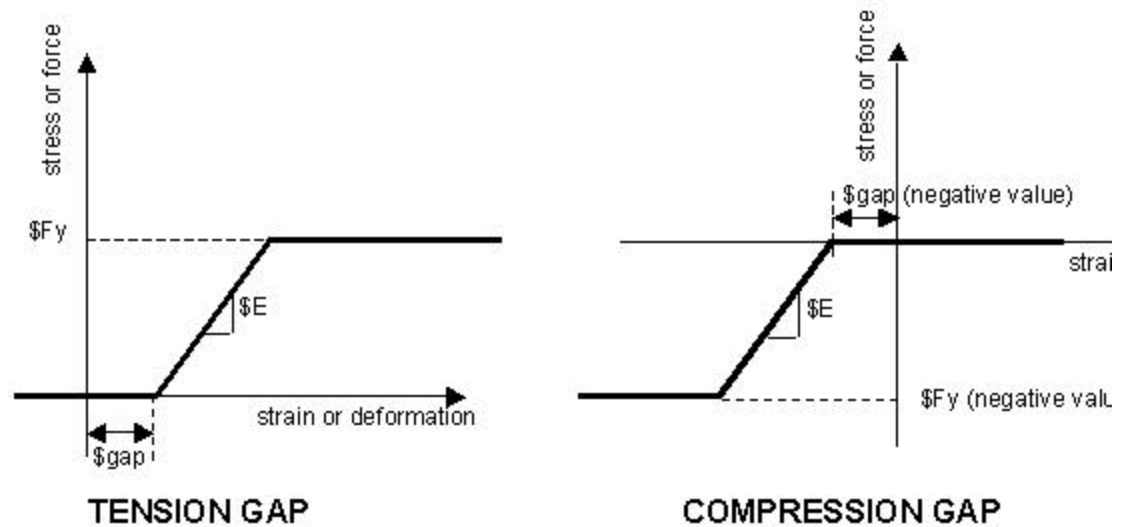


Figure 6: Elastic-
Perfectly Plastic Gap
Material

Parallel Material

This command is used to construct a parallel material object made up of an arbitrary number of previously-constructed *UniaxialMaterial* (page 29) 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:

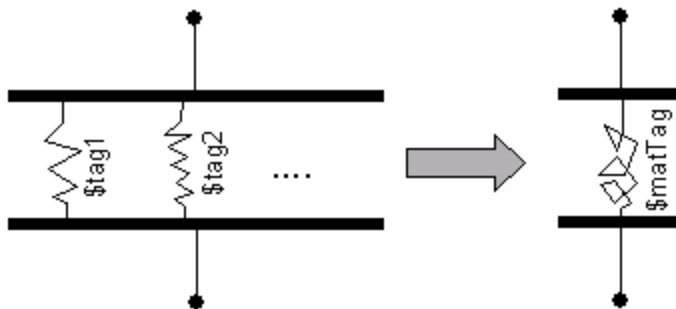
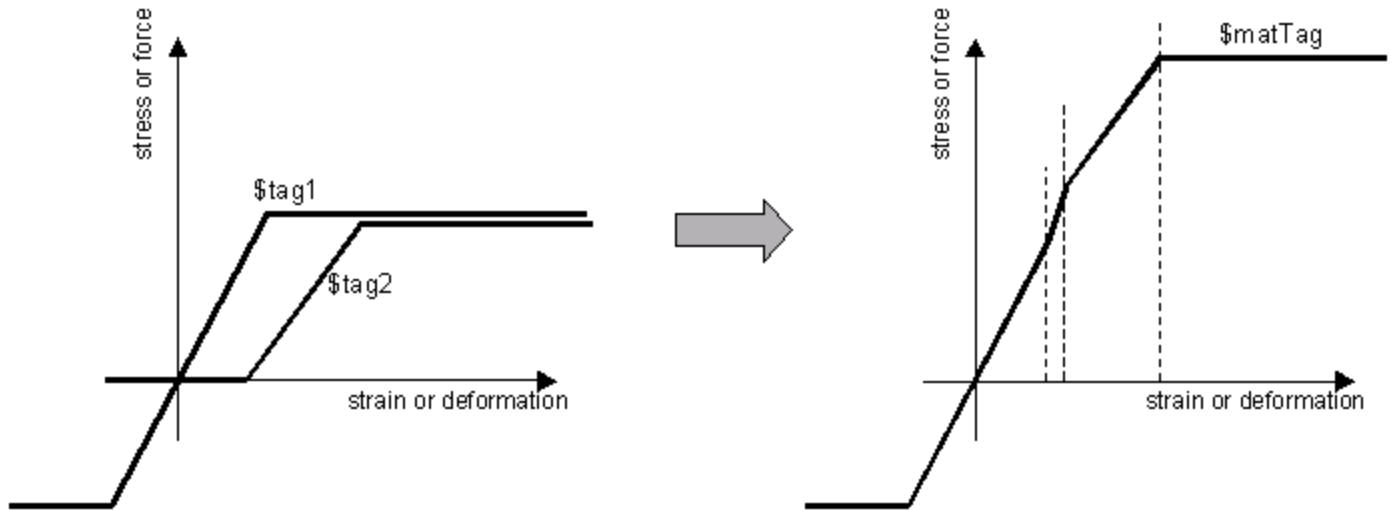


Figure 7: Parallel
Material

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 29) 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:

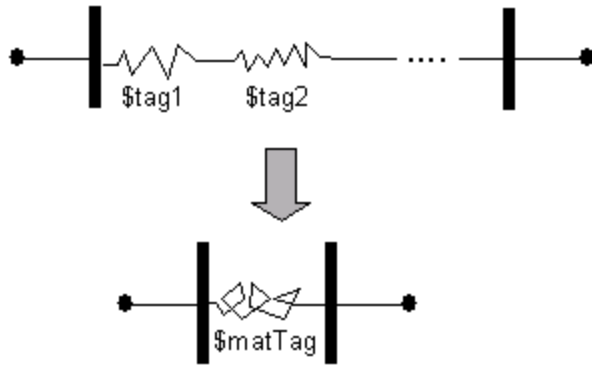


Figure 8: Series Material

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

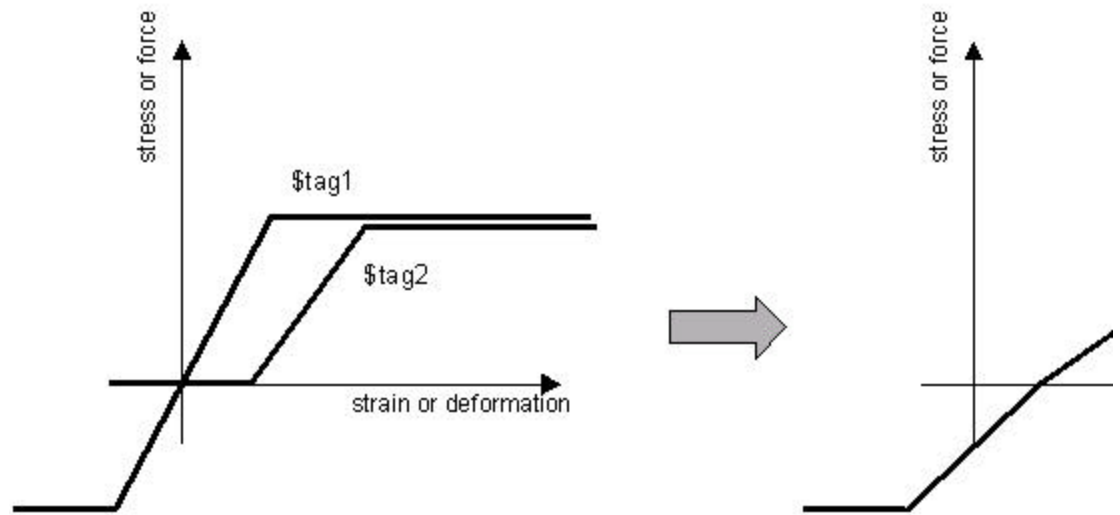


Figure 9: Series
Material Relationship

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)

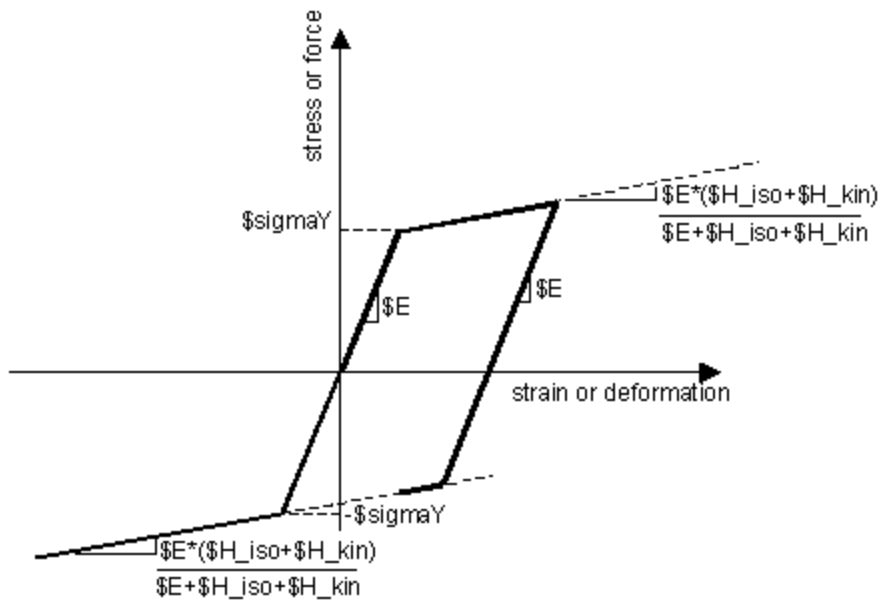


Figure 10: Hardening
Material

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: Fedeeas).

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

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

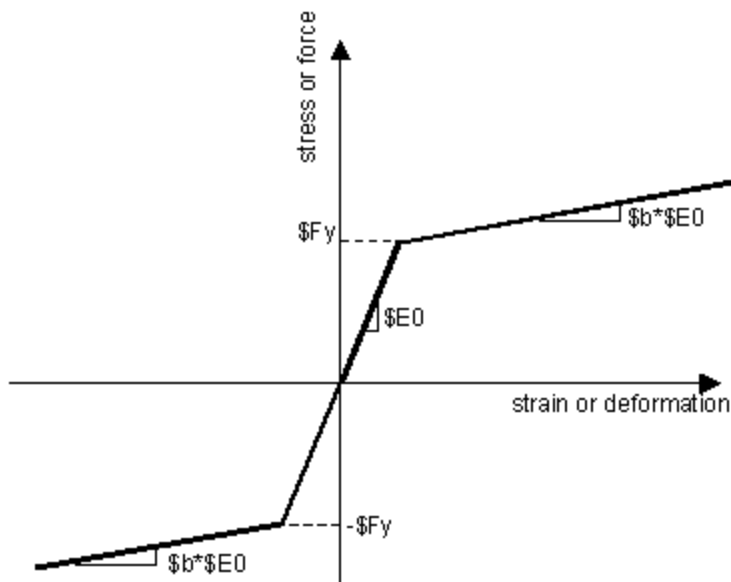


Figure 11: Steel01
Material

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: Fedeeas).

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

\$matTag	unique material object integer tag
\$fpc	compressive strength*
\$epsc0	strain at compressive strength*
\$fpcu	crushing strength*
\$epsU	strain at crushing strength*

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

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

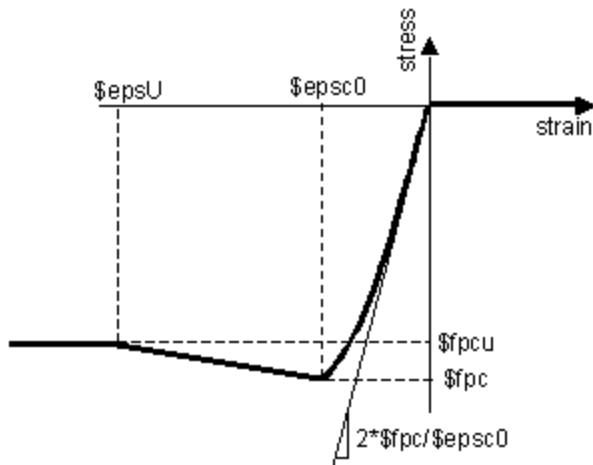


Figure 12: Concrete01
Material

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.

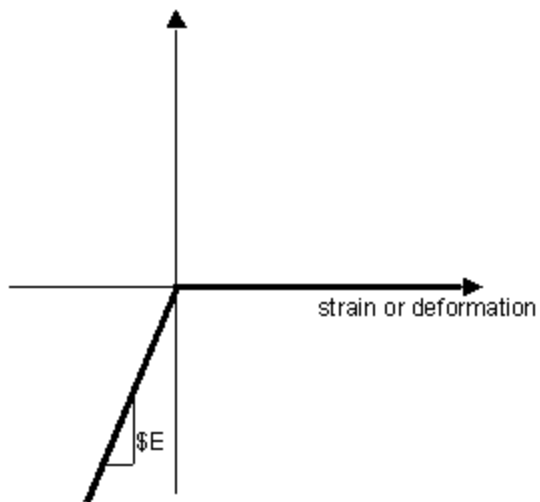


Figure 13: Elastic-No
Tension Material

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: $D_2(E_{ir}/E_{ult})$
\$beta	power used to determine the degraded unloading stiffness based on ductility, μ^{+beta} (optional, default=0.0)

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

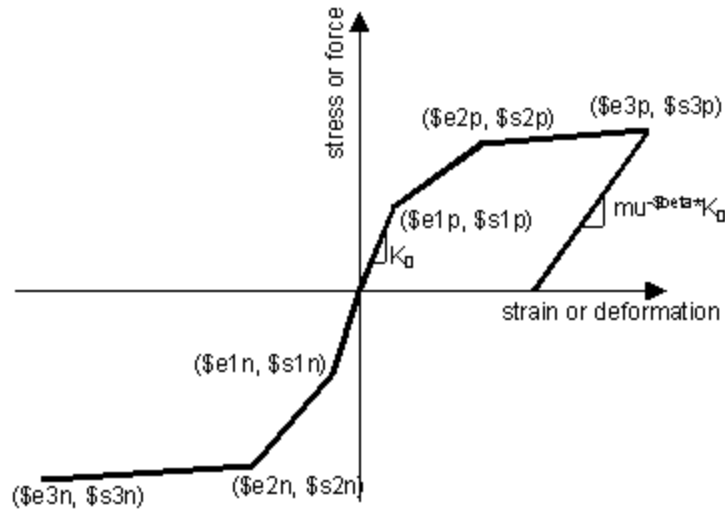


Figure 14: Hysteretic Material

Viscous Material

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

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

uniaxialMaterial Viscous \$matTag \$C \$alpha

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

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

Further information on the *Concrete01* (page 37) and *Steel01* (page 36) 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 \$ratio \$ft \$Ets

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

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

The initial slope for this model is $(2 * \text{\$fpc} / \text{\$epsc0})$

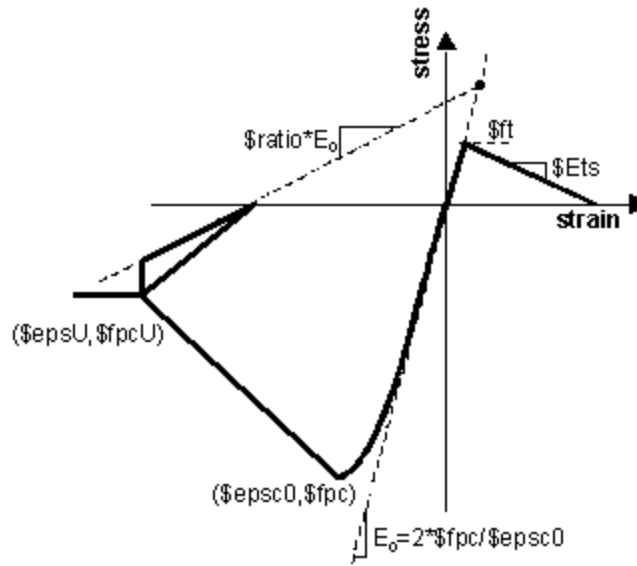


Figure 15: Concrete02
Material

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 \$ratio \$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*
\$ratio	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.

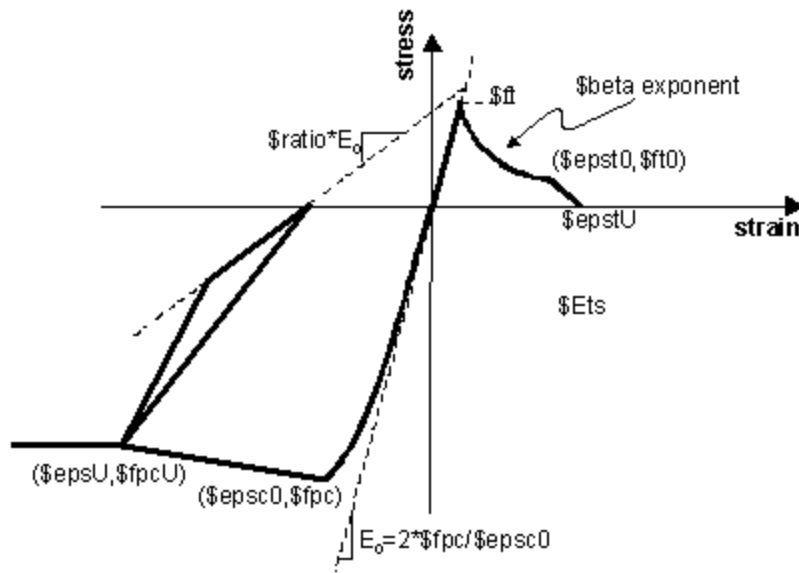


Figure 16: Concrete03
Material

Steel02 Material

This command is used to construct a uniaxial 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
\$R0 \$cR1 \$cR2	control the transition from elastic to plastic branches. Recommended values: \$R0=18.5, \$cR1=0.925, \$cR2=0.15
\$a1 \$a2 \$a3 \$a4	isotropic hardening parameters: no isotropic hardening: a1=a3=0; a2, a4=nonzero

Bond01 Material

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

```
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 Elgehausen 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.

CHAPTER 6

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 96) are 'strain,' 'stress,' and 'tangent.'

In This Chapter

Elastic Isotropic Material.....	46
J2 Plasticity Material	47
Plane Stress Material	47
Plate Fiber Material.....	48
Template Elasto-Plastic Material.....	48

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 79, page 76, page 77, page 75, page 76, page 77) 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 -YS $ys -PS $ps -EPS $eps -ELS1 $el <-  
ELT1 $et>
```

\$matTag	unique material object tag
\$ys	yield surface variable, previously defined in <i>Yield Surface</i> (page 49) object
\$ps	potential surface variable, previously defined in <i>Potential Surface</i> (page 49) object
\$eps	elasto-plastic state variable, previously defined in <i>EPState</i> (page 51) object
\$el	scalar evolution law variable, previously defined in <i>Evolution Law</i> (page 50) object
\$et	tensorial evolution law variable, previously defined in <i>Evolution Law</i> (page 50) object

Yield Surface

This command sets the yield surface variable **ys** to be the specified type. Currently these include: Drucker-Prager yield surface, von Mises yield surface and Cam-Clay 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 Cam-Clay yield surface

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

\$M

slope of the critical state line in p-q space

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, von Mises potential surface, Cam-Clay potential surface and Manzari Dafalias flow directions.

```
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 the Cam-Clay potential surface

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

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

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 evolution and tensorial 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, NLEij

 For linear scalar evolution law

set el "-Leq \$a"

 For nonlinear scalar evolution law

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

 For linear tensorial evolution law

set et "-LEij \$a1"

 For 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 include two states.

 To set the initial stress tensor to variable sts:

```
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 "$Eo $E $v $p -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)
\$Eo	Young's Modulus at atmospheric pressure
\$E	current Young's Modulus E
\$v	Poisson's ratio
\$p	mass density (mass/volume)
\$nt	number of tensorial internal variables
\$ns	number of scalar internal variables
\$scalar1 \$scalar2 ...	corresponding initial values of scalar internal variables
\$sts	initial stresses


CHAPTER 7

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 96) are 'force' and 'deformation.'

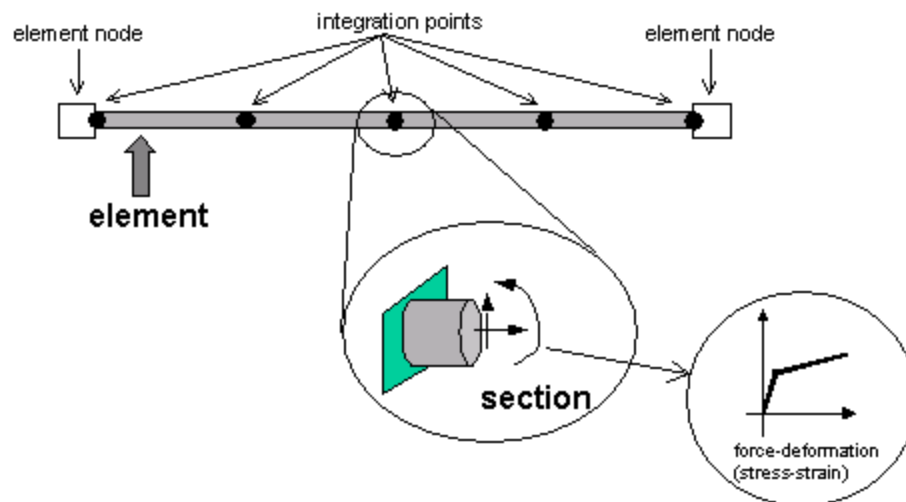


Figure 17: Section Representation

In This Chapter

Elastic Section	53
Uniaxial Section.....	54
Fiber Section	55
Section Aggregator.....	62
Elastic Membrane Plate Section.....	65
Plate Fiber Section.....	65
Bidirectional Section.....	66

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* . 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 29) 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 <i>UniaxialMaterial</i> (page 29) 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 57) and *layer* (page 61, page 60) are used to define the discretization of the section into fibers. Individual fibers, however, can also be defined using the *fiber* (page 56) command. During generation, the Fiber objects are associated with *uniaxialMaterial* (page 29) 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.

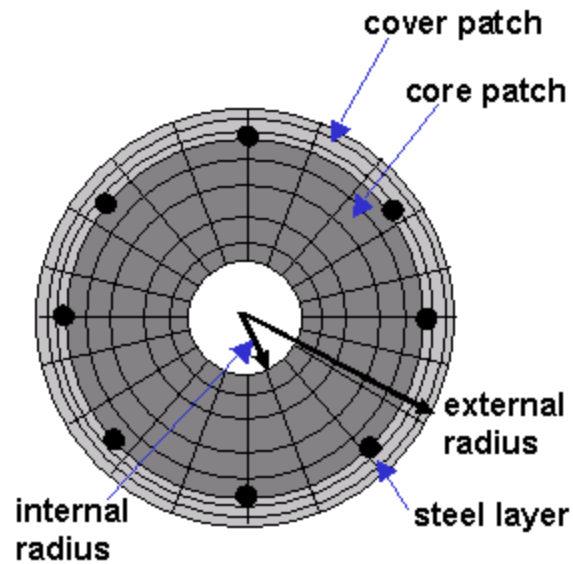


Figure 18: Fiber Section

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 <i>UniaxialMaterial</i> (page 29) object used to represent the stress-strain for the area of the fiber

NOTE: in 2D (page 21) 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
```

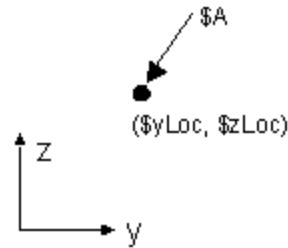



Figure 19: Fiber Command

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 <i>UniaxialMaterial</i> (page 29) 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 21) 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 rectangle of width 2b and depth 2h

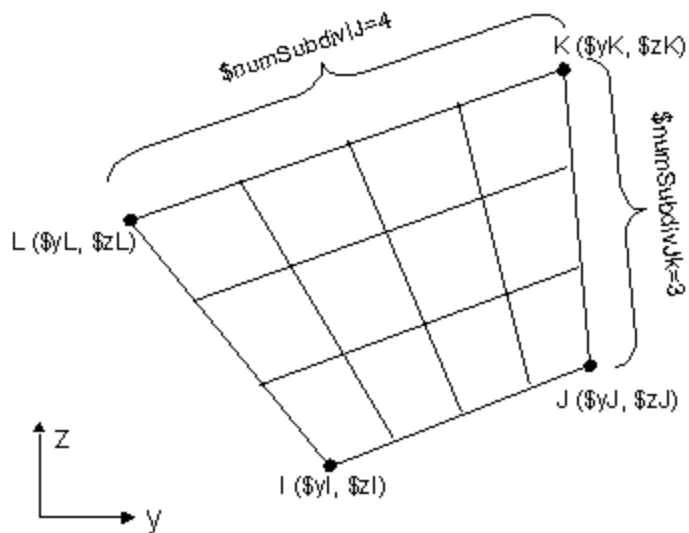


Figure 20:
Quadrilateral Patch

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 <i>UniaxialMaterial</i> (page 29) 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 21) 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
```

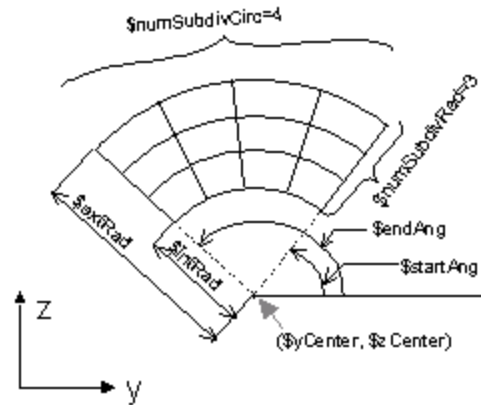


Figure 21: Circular Patch

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 <i>UniaxialMaterial</i> (page 29) 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 21) 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

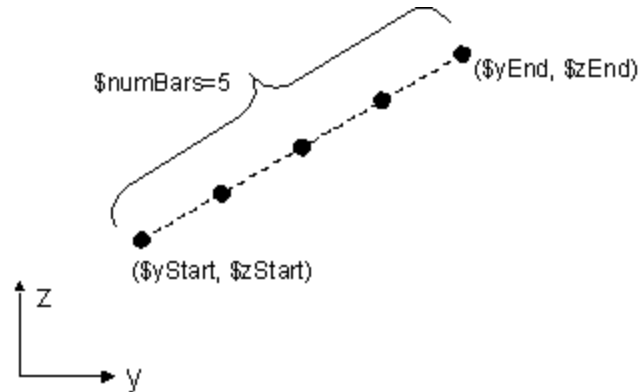


Figure 22: Straight Layer

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 <i>UniaxialMaterial</i> (page 29) 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 21) 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

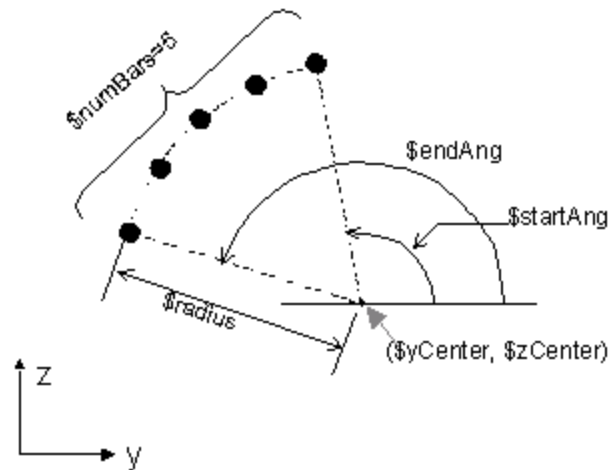


Figure 23: Circular Reinforcing Layer

Section Aggregator

This command is used to construct a `SectionAggregator` object which groups previously-defined *UniaxialMaterial* (page 29) objects into a single section force-deformation model.

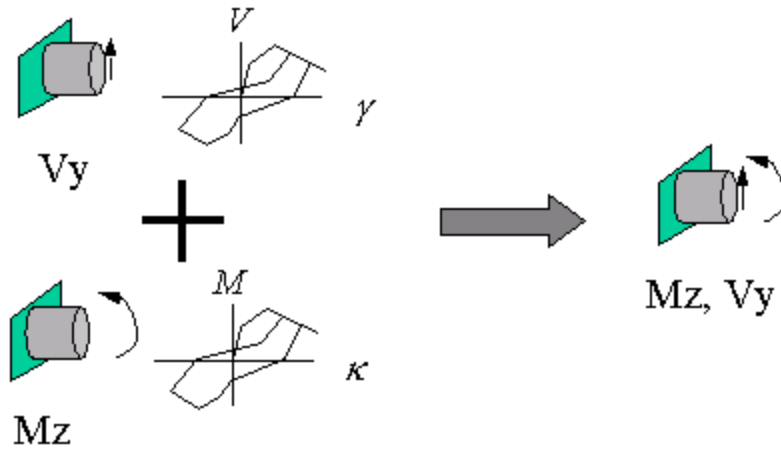
```
section Aggregator $secTag $matTag1 $string1 $matTag2 $string2 ..... <-  
section $sectionTag>
```

\$secTag	unique section object integer tag
\$matTag1, \$matTag2 ...	previously-defined <i>UniaxialMaterial</i> (page 29) 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 <i>Section</i> (page 52) object (identified by the argument <code>\$sectionTag</code>) to which these <i>UniaxialMaterial</i> (page 29) objects may be added to recursively define a new <i>Section</i> (page 52) object

NOTE: The *UniaxialMaterial* (page 29) objects aggregated in this *Section* (page 52) object are uncoupled from each other as well as from the *Section* (page 52) 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



EXAMPLE:

Figure 24: Section
Aggregator 1

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

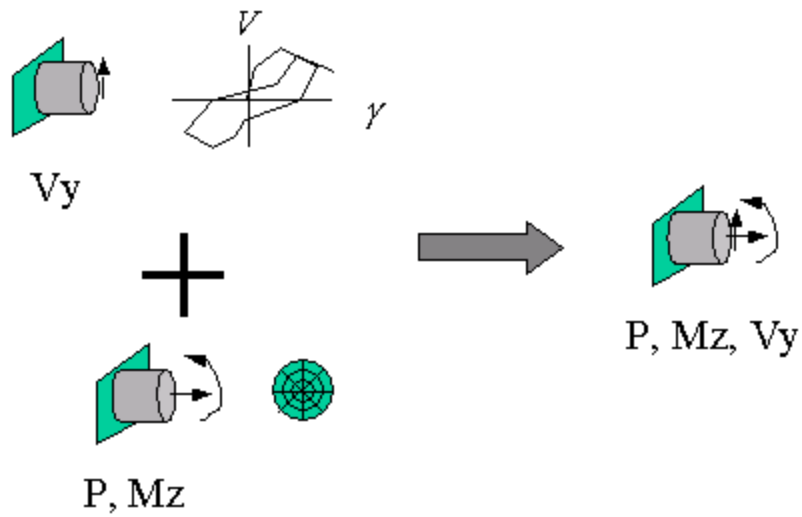


Figure 25: Section
Aggregator 2

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 between 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 48) 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 <i>plate fiber material</i> (page 48)
\$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 isotropic hardening parameters?????*****

CHAPTER 8

element Command

This command is used to construct an Element object.

In This Chapter

Truss Element.....	67
Corotational Truss Element	68
Elastic Beam Column Element.....	69
NonLinear Beam-Column Elements	70
Zero-Length Elements	73
Quadrilateral Elements.....	75
Brick Elements	77

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 29) identifier:

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

the other is to specify a *Section* (page 52) 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 <i>UniaxialMaterial</i> (page 29)
\$secTag	tag associated with previously-defined <i>Section</i> (page 52)

When constructed with a *UniaxialMaterial* (page 29) 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 96) 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 29) identifier:

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

the other is to specify a *Section* (page 52) 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 <i>UniaxialMaterial</i> (page 29) object
\$secTag	tag associated with previously-defined <i>Section</i> (page 52) object

NOTE: When constructed with a *UniaxialMaterial* (page 29) 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 96) 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 17):

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 <i>coordinate-transformation</i> (page 89) (<code>CrdTransf</code>) object

The valid queries to an elastic beam-column element when creating an *ElementRecorder* (page 96) 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 70))

 Concentrated plasticity with elastic interior (*beamWithHinges* (page 71))

Displacement based element

 Distributed plasticity with linear curvature distribution (*dispBeamColumn* (page 72))

*****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 <i>section</i> (page 52) object
\$transfTag	identifier for previously-defined <i>coordinate-transformation</i> (page 89) (<i>CrdTransf</i>) 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 52) 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 96) 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 17).

For a two-dimensional problem:

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

For a three-dimensional problem:

```
element beamWithHinges $eleTag $iNode $jNode $secTagI $ratioI $secTagJ
    $ratioJ $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 <i>section</i> (page 52) object corresponding to node I
\$ratioI	ratio of hinge length to total element length at node I
\$secTagJ	identifier for previously-defined <i>section</i> (page 52) object corresponding to node J
\$ratioJ	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 <i>coordinate-transformation</i> (page 89) (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 96) 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 <i>section</i> (page 52) object
\$transfTag	identifier for previously-defined <i>coordinate-transformation</i> (page 89) (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 96) 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 29) 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 29)

\$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 96) 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 52) 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 <i>Section</i> (page 52) 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 52) 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 96) 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 <i>NDMaterial</i> (page 46) object and its available material formulations. The type parameter can be either "PlaneStrain" or "PlaneStress."
\$matTag	tag associated with previously-defined <i>NDMaterial</i> (page 46) 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 zero-length element when creating an *ElementRecorder* (page 96) 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 <i>SectionForceDeformation</i> (page 52) object. Typically, corresponds to some <i>PlateFiberSection</i> (page 65), 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 zero-length element when creating an *ElementRecorder* (page 96) 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 <i>NDMaterial</i> (page 46) 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 <i>NDMaterial</i> (page 46) object and its available material formulations. The type parameter can be either "PlaneStrain" or "PlaneStress."
\$matTag	tag associated with previously-defined <i>NDMaterial</i> (page 46) 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 96) object are 'force,' 'stiffness,' and 'material matArg1 matArg2 ...'

Brick Elements

Standard Brick Element

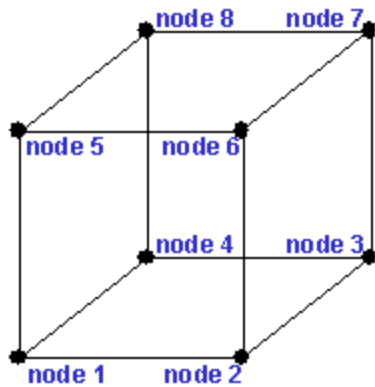
This element is used to construct an eight-node brick element object, which uses a trilinear isoparametric formulation.

```
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 <i>NDMaterial</i> (page 46) 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 26: Node Numbering for Eight-Node 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 <i>NDMaterial</i> (page 46) object

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

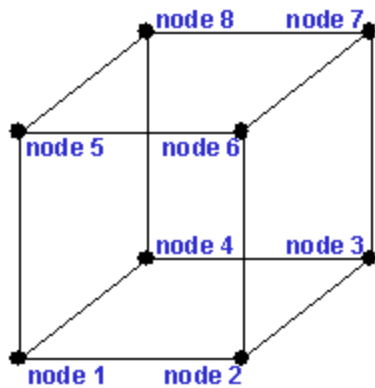


Figure 27: Node
Numbering for Eight-
Node Element

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 previously-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 96) 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

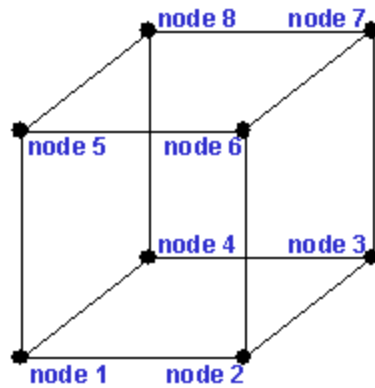


Figure 28: Node Numbering for Eight-Node Element

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 previously-defined <i>NDMaterial</i> (page 46) 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 96) 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

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 96) 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

CHAPTER 9

block Command

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

The *block2D* (page 85) 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 87) command generates three-dimensional meshes of eight-node brick solid element.

In This Chapter

<code>block2D</code> Command	85
<code>block3D</code> Command	87

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 <i>quadrilateral element</i> (page 76, page 77, page 75, page 76) 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.

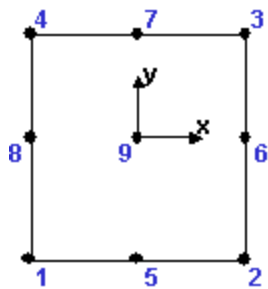
EXAMPLE:

```

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 29: 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 <i>brick element</i> (page 79, page 77) 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.

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 10

Geometric Transformation Commands

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	89
P-Delta Transformation.....	91
Corotational Transformation.....	92

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	<p>X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system.</p> <p>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.</p> <p>These items need to be specified for the three-dimensional problem.</p>
\$dXi \$dYi \$dZi	<p>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)</p>
\$dXj \$dYj \$dZj	<p>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)</p>

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 the local x-z plane -- vecxz . 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.

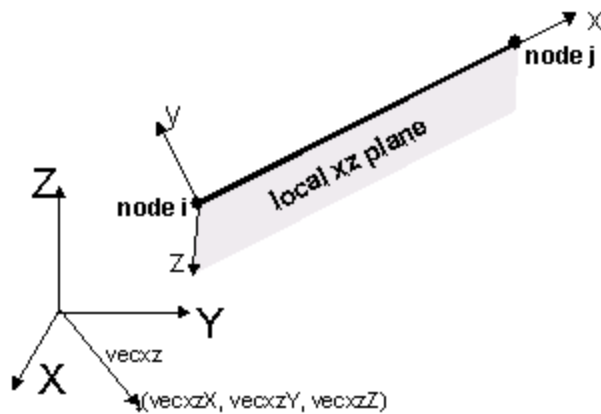


Figure 30: Definition of the Local Coordinate System

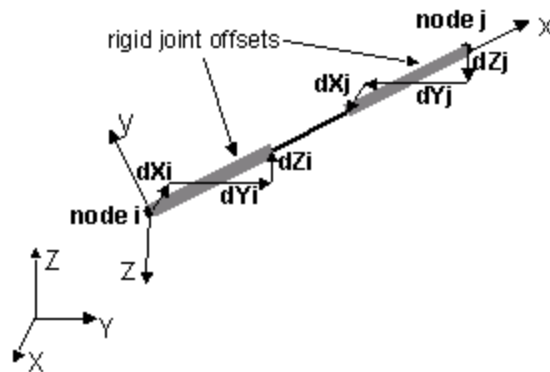


Figure 31: Definition of Rigid Joint Offset
(note: check sign of dXi , etc components)

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 89).

\$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 <i>model</i> (page 21)) (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 <i>model</i> (page 21)) (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* (see <http://opensees.berkeley.edu/OpenSees/binaries.html>).

The element coordinate system and joint offset values are specified as in the *Linear transformation* (page 89).

\$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 21)) (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 21)) (optional)

CHAPTER 11

recorder Command

The recorder command is used to construct a Recorder object, which is used to monitor items of interest to the analyst at each commit(). The format of the recorder command is:

```
recorder recorderType <arguments>
```

where the arguments correspond to the recorderType.

Valid strings for the recorderType are presented in this chapter, and given below.

In This Chapter

Node Recorder	94
MaxNodeDisp Recorder	95
Drift Recorder	96
Element Recorder	96
Display Recorder	99
Plot Recorder	99
playback Command	100

Node Recorder

The Node type records the displacement, velocity, acceleration and incremental displacement at the nodes (translational & rotational)

```
recorder Node $fileName $respType <-time> -node $node1 $node2 ... -dof  
$dof1 $dof2 ...
```

\$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain						
\$respType	defines response type to be recorded. The following response types are available: <table> <tr> <td>disp</td><td>displacement</td></tr> <tr> <td>vel</td><td>velocity</td></tr> <tr> <td>accel</td><td>acceleration</td></tr> </table>	disp	displacement	vel	velocity	accel	acceleration
disp	displacement						
vel	velocity						
accel	acceleration						

incrDisp	incremental displacement
<-time>	this argument will place the pseudo time of the <i>Domain</i> (page 18) as the first entry in the line. (optional)
\$node1 \$node2 ...	<i>nodes</i> (page 22) where response is being recorded
\$dof1 \$dof2 ...	degrees of freedom of response being recorded. Valid range is from 1 through <i>ndf</i> (page 21), the number of nodal degrees-of-freedom.

Example:

```
recorder Node node.out disp -time -node 1 5 -dof 2
```

MaxNodeDisp Recorder

The MaxNodeDisp type records the values of the maximum absolute values of the displacement in the prescribed direction of a prescribed set of nodes

```
recorder MaxNodeDisp $dof $node1 $node2 ...
```

\$dof	displacement degree-of-freedom direction. Valid range is from 1 through <i>ndf</i> (page 21), the number of nodal degrees-of-freedom.
\$node1 \$node2 ...	<i>nodes</i> (page 22) where maximum displacement is being recorded

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 <i>Domain</i> (page 18) as the first entry in the line. (optional)
\$node1 \$node2 ...	the two <i>nodes</i> (page 22) for which drift is recorded
\$dof	nodal degree of freedom to monitor for drift Valid range is from 1 through <i>ndf</i> (page 21), 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. (REF?)

```
recorder Element $eleID1 $eleID2 ... <-file $fileName> <-time> $arg1 $arg2 ...
```

\$eleID1 \$eleID2 ... Tags of elements whose response is being recorded.

\$fileName	file where results are stored. Each line of the file contains the result for a committed state of the domain. (optional; default: screen display).
<-time>	this argument will place the pseudo time of the <i>Domain</i> (page 18) as the first entry in the line. (optional)
\$arg1 \$arg2 ...	arguments which are passed to the <code>setResponse()</code> element method

The `setResponse()` element method is dependent on the element type, and is described with the *element Command* (page 67).

 **Beam-Column Elements** (page 71, page 72, page 69, page 70):

Common to all beam-column elements:

globalForce – element resisting force in global coordinates (does not include inertial forces)

example: recorder Element 1 ele1global.out –time globalForce

localForce – element resisting force in local coordinates (does not include inertial forces)

example: recorder Element 1 ele1local.out –time localForce

 **Sections** (page 52):

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 1 ele1sec1Force.out –time section 1 force

deformation – section deformations

example: recorder Element 1 ele1sec1Defo.out –time section 1 deformation

stiffness – section stiffness

example: recorder Element 1 ele1sec1Stiff.out –time section 1 deformation

stressStrain – record *fiber* (page 56) stress-strain response.

example: recorder Element 1 ele1sec1fiberYZ.out –time 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 21). 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 94).

playback \$commitTag

\$commitTag

integer used to invoke the record() method (????)

CHAPTER 12

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 106) and *groundMotion* (page 109) objects.

Time series act differently depending on what type of object they are applied to:

LoadPattern (page 106) **object:**

Load factors are applied to the loads and constraints

groundMotion (page 109) **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	102
Linear Time Series	102
Rectangular Time Series	103
Sine Time Series	104
Path Time Series	105

Constant Time Series

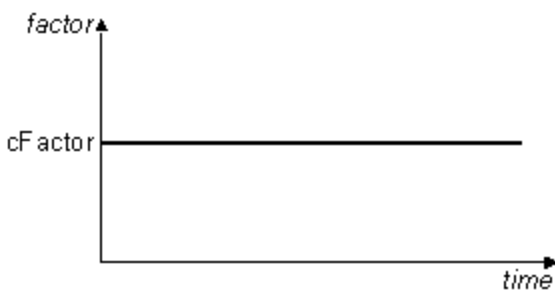
This command creates a `ConstantSeries` *TimeSeries* (page 101) object and associates it to the *LoadPattern* (page 106) object being constructed.

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 32: Constant Time Series



Linear Time Series

This command creates a `LinearSeries` *TimeSeries* (page 101) object and associates it to the *LoadPattern* (page 106) or *groundMotion* (page 109) 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**

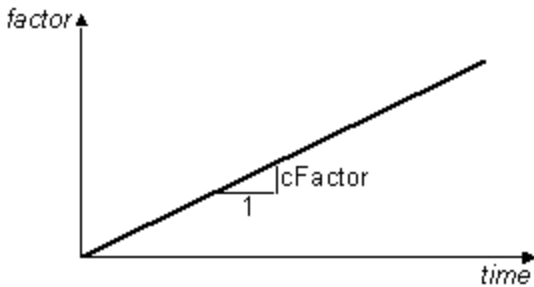


Figure 33: Linear Time Series

Rectangular Time Series

This command creates a RectangularSeries *TimeSeries* (page 101) object and associates it to the *LoadPattern* (page 106) 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**

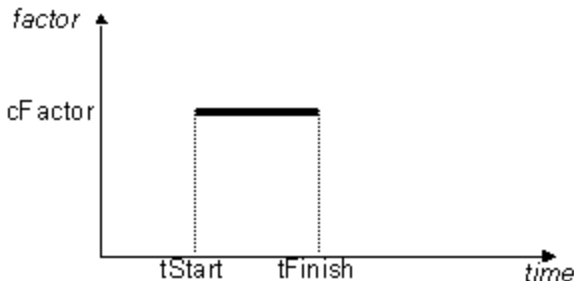


Figure 34: Rectangular
Time Series

Sine Time Series

This command creates a TrigSeries *TimeSeries* (page 101) object and associates it to the *LoadPattern* (page 106) 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:

$$cFactor \cdot \sin \left[\frac{2 \cdot \pi \cdot (\text{time} - tStart)}{\text{period}} + \text{shift} \right]$$

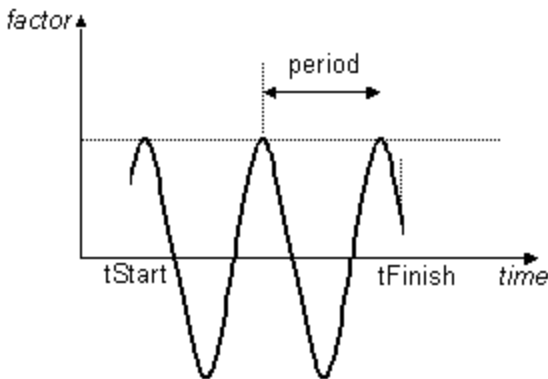


Figure 35: Sine Time Series

Path Time Series

This command associates a *TimeSeries* (page 101) object of the type *PathSeries* or *PathTimeSeries* (if the increment is not constant) to a *LoadPattern* (page 106) 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 13

pattern Command

The pattern command is used to construct a *LoadPattern* object, its associated with the *TimeSeries* (page 101) object and the *Load* (page 107) and *Constraint* (page 115) objects for the pattern.

In This Chapter

plain Pattern.....	106
UniformExcitation Pattern.....	108
MultipleSupport Pattern	109

plain Pattern

This command is used to construct an ordinary *LoadPattern* (page 106) object in the *Domain* (page 18).

```
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 <i>TimeSeries</i> (page 101) object associated with the <i>LoadPattern</i> object.
load ...	list of commands to construct nodal loads -- the <i>NodalLoad</i> (page 107) object
sp ...	list of commands to construct single-point constraints -- the <i>SP_Constraint</i> (page 108) object
eleLoad ...	list of commands to construct element loads -- the <i>eleLoad</i> (page 108) 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 <i>ndf</i> (page 21), 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 <i>ndf</i> (page 21), the number of nodal degrees-of-freedom.
\$DOFvalue	reference value of the constraint to be applied to the DOF at the node.

EXAMPLE

```
load 3 0.1 0. 0. 0. 0. 0.;      # 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}	<i>TimeSeries</i> (page 101) 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 109) 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 111) 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 111) constraints in a *MultipleSupportExcitation* (page 109) object.

Plain GroundMotion

This command is used to construct a plain GroundMotion object. Each GroundMotion object is associated with a number of *TimeSeries* (page 101) 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 109) object tag

<-accel {accelSeriesType accelArgs}>

TimeSeries (page 101) objects defining the acceleration record (optional).

<-vel {velSeriesType velArgs}>

TimeSeries (page 101) objects defining the velocity record (optional)

<-disp {dispSeriesType dispArgs}>

TimeSeries (page 101) 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 119) 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 109) object tag

\$gmTag1 \$gmTag2 ... ground motions which have already been added to the *MultipleSupportExcitation* (page 109) 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 109) 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 21), the number of nodal degrees-of-freedom.

\$gMotionTag pre-defined *GroundMotion* (page 109) object tag

NOTE: The *GroundMotion* (page 109) object must be added to the *MultipleSupportExcitation* (page 109) pattern before the *ImposedMotionSP* constraint.

FMK: ADD TIME-SERIES INTEGRATORS

CHAPTER 14

analysis Command

This command is used to construct the *Analysis object* (page 19).

All currently-available analysis objects employ incremental solution strategies.

In This Chapter

Static Analysis	112
Transient Analysis	113
VariableTransient Analysis	113

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
<i>SolutionAlgorithm</i> (page 126), <i>StaticIntegrator</i> (page 121, page 119, page 120, page 122)	<i>NewtonRaphson</i> (page 126) <i>EquiSolnAlgo</i> with a <i>CTestNormUnbalance</i> (page 129) with a tolerance of 1e-6 and a maximum of 25 iterations
<i>ConstraintHandler</i> (page 115)	<i>PlainHandler</i> (page 116) <i>ConstraintHandler</i>
<i>DOF_Numberer</i> (page 132)	<i>RCM</i> (page 133) <i>DOF_Numberer</i>
<i>LinearSOE</i> (page 134), <i>LinearSolver</i> (page 134)	<i>profiled symmetric positive definite</i> (page 135) <i>LinearSOE</i> and <i>Linear Solver</i>

Integrator	<i>LoadControl StaticIntegrator</i> (page 120) 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
<i>SolutionAlgorithm</i> (page 126), <i>TransientIntegrator</i> (page 122, page 124, page 119, page 123)	<i>NewtonRaphson</i> (page 126) <i>EquiSolnAlgo</i> with a <i>CTestNormUnbalance</i> (page 129) with a tolerance of 1e-6 and a maximum of 25 iterations
<i>ConstraintHandler</i> (page 115)	<i>PlainHandler</i> (page 116) <i>ConstraintHandler</i>
<i>DOF_Numberer</i> (page 132)	<i>RCM</i> (page 133) <i>DOF_Numberer</i>
<i>LinearSOE</i> (page 134), <i>LinearSolver</i> (page 134)	<i>profiled symmetric positive definite</i> (page 135) <i>LinearSOE</i> and <i>Linear Solver</i>
Integrator	<i>Newmark TransientIntegrator</i> (page 123) with $\gamma=0.5$ and $\beta=0.25$

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
<i>SolutionAlgorithm</i> (page 126), <i>TransientIntegrator</i> (page 122, page 124, page 119, page 123)	<i>NewtonRaphson</i> (page 126) <i>EquiSolnAlgo</i> with a <i>CTestNormUnbalance</i> (page 129) with a tolerance of 1e-6 and a maximum of 25 iterations
<i>ConstraintHandler</i> (page 115)	<i>PlainHandler</i> (page 116) <i>ConstraintHandler</i>
<i>DOF_Numberer</i> (page 132)	<i>RCM</i> (page 133) <i>DOF_Numberer</i>
<i>LinearSOE</i> (page 134), <i>LinearSolver</i> (page 134)	<i>profiled symmetric positive definite</i> (page 135) <i>LinearSOE</i> and <i>Linear Solver</i>
<i>Integrator</i>	<i>Newmark TransientIntegrator</i> (page 123) with $\gamma=0.5$ and $\beta=0.25$

CHAPTER 15

constraints Command

This 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.

The constraint equations may take the following forms:

$$U_c = C_{rc} U_r \quad T U_r = [U_r \ U_c]^T$$

$$C U = 0 \quad [C_r \ C_c]^T [U_r \ U_c] = 0$$

where

$$C_{rc} = -C_c^{-1} C_r \quad T = [I \ C_{rc}]^T$$

U represents the degrees of freedom. The matrix C contains constants, and the matrix I is the identity matrix. The matrix T is the transformation matrix, defined above. The subscripts indicate the following: r =retained, c =condensed.

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)

In This Chapter

Plain Constraints	116
Penalty Method	116
Lagrange Multipliers.....	117
Transformation Method	117

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 $\{\mathbf{t}\}^T [\boldsymbol{\alpha}] \{\mathbf{t}\} / 2$

where $[\boldsymbol{\alpha}]$ is a diagonal matrix of "penalty numbers". The resulting system of equations is of the form:

$$[\mathbf{K} + \mathbf{C}^T \boldsymbol{\alpha} \mathbf{C}] \mathbf{U} = [\mathbf{R} + \mathbf{C}^T \boldsymbol{\alpha} \mathbf{Q}]$$

Where $\mathbf{C}^T \boldsymbol{\alpha} \mathbf{C}$ can be called the penalty matrix. \mathbf{C} and \mathbf{Q} are matrices containing constants, \mathbf{K} is the stiffness matrix, \mathbf{U} represents the DOF and \mathbf{R} the restoring forces. If $\boldsymbol{\alpha} = 0$ the constraints are ignored. As $\boldsymbol{\alpha}$ grows, \mathbf{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.

"Guideline for choice of $\boldsymbol{\alpha}$: If computer words carry approximately p decimal digits, experience has shown that $\boldsymbol{\alpha}$ 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. 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".

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 defined due to the introduction of zeroes on the diagonal by the constraint equations:

$$\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} U \\ \lambda \end{bmatrix} = \begin{bmatrix} R \\ Q \end{bmatrix}$$

Transformation Method

This command is used to construct a TransformationConstraintHandler 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 16

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 (??).

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:





$$R(U, I) = \lambda P^* - F_R(U)$$

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 112) or **transient analysis** (page 113):

STATIC ANALYSIS*

 <i>LoadControl</i> (page 120)	$\lambda_n = \lambda_{n-1} + d\lambda$
 <i>DisplacementControl</i> (page 121)	$U_{j_n} = U_{j_{n-1}} + dU_j$
 <i>MinUnbalDispNorm</i> (page 122)	$d/d\lambda (dU_n^T dU_n) = 0$
 <i>ArcLength</i> (page 122)	$dU_n^T dU_n + \alpha^2 d\lambda_n = ds^2$

TRANSIENT ANALYSIS

-  *Newmark* (page 123)
-  *Hilbert-Hughes-Taylor Method (HHT)* (page 124))

***NOTE:** static integrators should only be used with a *Linear TimeSeries* (page 102) object with a factor of 1.0.

In This Chapter

Load Control	120
Displacement Control.....	121
Minimum Unbalanced Displacement Norm	122
Arc-Length Control	122
Newmark Method.....	123
Hilbert-Hughes-Taylor.....	124

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 <i>analysis</i> (page 112) 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 , $d\text{Lambda}(i)$, is related to the load increment at $(i-1)$, $d\text{Lambda}(i-1)$, and the number of iterations at $(i-1)$, $J(i-1)$, by the following:

$$d\text{Lambda}(i) = d\text{Lambda}(i-1) * Jd / J(i-1)$$

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 <i>ndf</i> (page 21), 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:

$$dU(i) = dU(i-1) * Jd / J(i-1)$$

$$U_j = U_{j-1} + dU_j$$

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 <i>analysis</i> (page 112) 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 , $d\Lambda_1(i)$, is related to the load increment at $(i-1)$, $d\Lambda_1(i-1)$, and the number of iterations at $(i-1)$, $J(i-1)$, by the following:

$$d\Lambda_1(i) = d\Lambda_1(i-1) * Jd / J(i-1)$$

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 form:

$$g(p, \lambda) = q_i(p) - \lambda \cdot q_{ef} = 0$$

the arc-length method aims to find the intersection of the above equation with $s = \text{constant}$, 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:

$$a = \left(\psi^T \cdot p + \psi^2 \cdot q_{ef}^T \cdot q_{ef} \right) - \psi^2 l^2 = 0$$

$$a = \left(\Delta p^T \cdot \Delta p + \Delta \lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef} \right) - \Delta l^2 = 0$$

$$a = \left(\Delta p^T \cdot \Delta p + \Delta \lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef} \right) - \Delta l^2 = 0$$

$$a = \left(\Delta p^T \cdot \Delta p + \Delta \lambda^2 \cdot \psi^2 \cdot q_{ef}^T \cdot q_{ef} \right) - \Delta l^2 = 0$$

where ψl 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 ?
\$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 17

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.....	126
Newton Algorithm	126
Newton with Line Search Algorithm	127
Modified Newton Algorithm	127
Krylov-Newton Algorithm	128
BFGS Algorithm.....	128
Broyden Algorithm	128

Linear Algorithm

This command is used to construct a Linear algorithm object which takes one iteration to solve the system of equations.

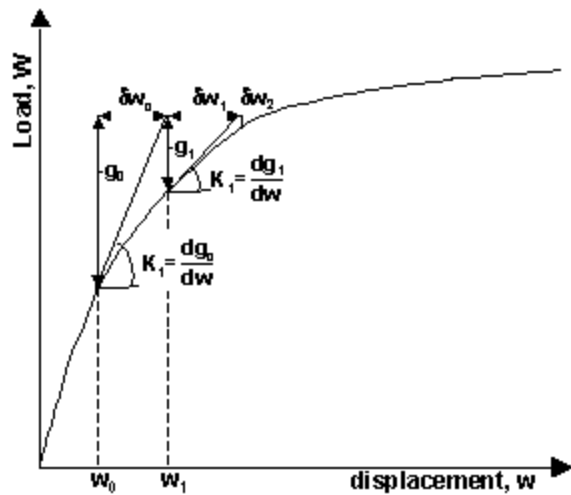
algorithm Linear

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.

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* (see <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 18

test Command

This command is used to construct a `ConvergenceTest` object. Certain *SolutionAlgorithm* (page 126) 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	$\sqrt{R^T R} < \text{tol}$
 Norm Displacement Increment	$\sqrt{\Delta U^T \Delta U} < \text{tol}$
 Energy Increment	$\frac{1}{2} (\Delta U^T R) < \text{tol}$

In This Chapter

Norm Unbalance Test	129
Norm Displacement Increment T	130
Energy Increment Test.....	131

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 134) 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:

$$\sqrt{R^T R} < \text{tol}$$

Norm Displacement Increment T

This command is used to construct a CTestNormDisplncr object which tests positive force convergence if the 2-norm of the x vector (the displacement increment) in the *LinearSOE* (page 134) object is less than the specified tolerance.

test NormDisplncr \$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)
	<ul style="list-style-type: none"> 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:

$$\sqrt{\Delta U^T \Delta U} < \text{tol}$$

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 134) 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:



$$\frac{1}{2} (\Delta U^T R) < \text{tol}$$

CHAPTER 19

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.

The types of numberer objects available:

-  **Plain** (page 132) nodes are assigned degrees-of-freedom arbitrarily
-  **RCM** (page 133) nodes are assigned degrees-of-freedom using the Reverse Cuthill-McKee algorithm (REF???)

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.....	132
RCM Numberer	133

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 20

system Command

This command is used to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.

- Profile Symmetric Positive Definite (SPD)



system ProfileSPD

- Banded Symmetric Positive Definite



system BandSPD

- Sparse Symmetric Positive Definite



system SparseSPD

- Banded General



system BandGeneral

- Sparse Symmetric



system SparseGeneral

system Umfpack

Figure 36: *system* command

In This Chapter

BandGeneral SOE	135
BandSPD SOE	135
ProfileSPD SOE.....	135
SparseGeneral SOE	135
UmfPack SOE.....	136
SparseSPD SOE	136

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 21

analyze Command

This command is invoked on the Analysis object constructed with the *analysis command* (page 112).

```
analyze $numIncr <$dt> <$dtMin $dtMax $Jd>
```

\$numIncr	number of load steps
\$dt	time-step increment. Required if a transient analysis (page 113) or variable time step transient analysis (page 113) was specified.
\$dtMin \$dtMax	minimum and maximum time steps Required if a variable time step transient analysis (page 113) was specified.
\$Jd	number of iterations performed at each step Required if a variable time step transient analysis (page 113) was specified.

This command RETURNS:

0 successful
<0 unsuccessful

CHAPTER 22

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: $K - \omega^2 M$

generalized solve: $K - \omega^2 M$ (default)

standard solve: $K - \omega^2 I$

\$numEigenvalues

number of first eigenvalues (?? to be determined)

The eigenvectors are stored at the nodes and can be printed out using *Node Recorder* (page 94) or the *Print command* (page 141).

CHAPTER 23

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..... 139

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 140) 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 139) command was invoked

CHAPTER 24

Miscellaneous Commands

These are a few additional miscellaneous command used in OpenSees

In This Chapter

print Command.....	141
reset Command	142
wipe Command	142
wipeAnalysis Command.....	142
loadConst Command	143
getTime Command	143
nodeDisp Command	143
video Command	144

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 18) 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 15).

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 67), *nodes* (page 22), *materials* (page 46, page 29), etc. It does destroy the solution strategies: the *algorithm* (page 126), *analysis* (page 112), *equation solver* (page 134), *constraint handler* (page 115), etc.

loadConst Command

This command is used to invoke `setLoadConst()` on all *LoadPattern* (page 106) 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 21), 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 99) command.

```
video -file $fileName -window $windowName
```

\$fileName fileName of images file created by the recorder *display* (page 99) command

\$windowName name of window to be created

The images are displayed by invoking the *play* (page 144) command.

play Command

This command is used to play the `TclVideoPlayer` object created by the *video* (page 144) command.

```
play
```


CHAPTER 25

How To....

In this chapter, some examples on how to generate input for OpenSees for specific tasks will be presented.

In This Chapter

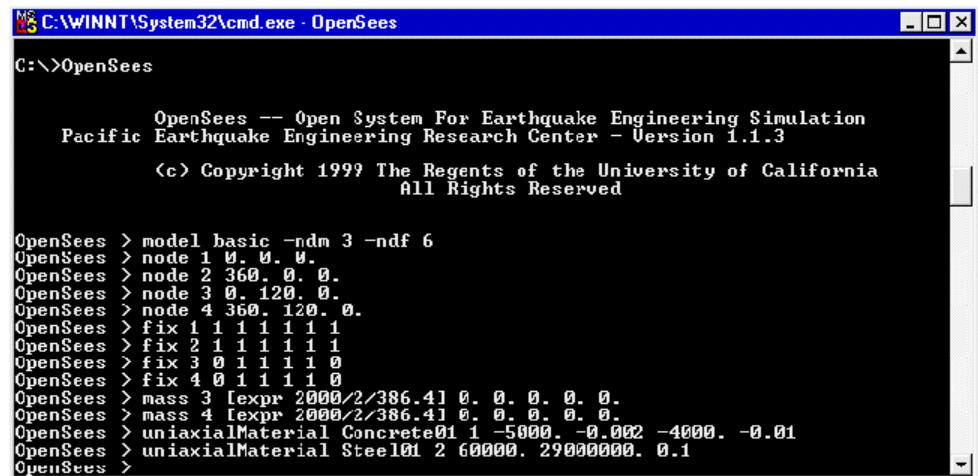
...Run OpenSees.....	146
...Define Units & Constants	148
...Generate Matlab Commands	149
...Define Tcl Procedure	149
...Read External files	151
Building The Model	152
Defining Output.....	158
Gravity Loads.....	159
Static Analysis	160
Dynamic Analysis	162
...Combine Input-File Components	163
...Run Parameter Study.....	164
...Run Moment-Curvature Analysis on Section	165
...Determine Natural Period & Frequency	167

...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):



```

C:\WINNT\System32\cmd.exe - OpenSees
C:\>OpenSees

OpenSees -- Open System For Earthquake Engineering Simulation
Pacific Earthquake Engineering Research Center - Version 1.1.3
(c) Copyright 1999 The Regents of the University of California
All Rights Reserved

OpenSees > model basic -ndm 3 -ndf 6
OpenSees > node 1 0. 0. 0.
OpenSees > node 2 360. 0. 0.
OpenSees > node 3 0. 120. 0.
OpenSees > node 4 360. 120. 0.
OpenSees > fix 1 1 1 1 1 1
OpenSees > fix 2 1 1 1 1 1
OpenSees > fix 3 0 1 1 1 0
OpenSees > fix 4 0 1 1 1 0
OpenSees > mass 3 lexpr 2000/2/386.41 0. 0. 0. 0.
OpenSees > mass 4 lexpr 2000/2/386.41 0. 0. 0. 0.
OpenSees > uniaxialMaterial Concrete01 1 -5000. -0.002 -4000. -0.01
OpenSees > uniaxialMaterial Steel01 2 60000. 29000000. 0.1
OpenSees >

```

Figure 37: 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):



```

C:\openSees.exe

OpenSees -- Open System For Earthquake Engineering Simulation
Pacific Earthquake Engineering Research Center - Version 1.1.3

(c) Copyright 1999 The Regents of the University of California
All Rights Reserved

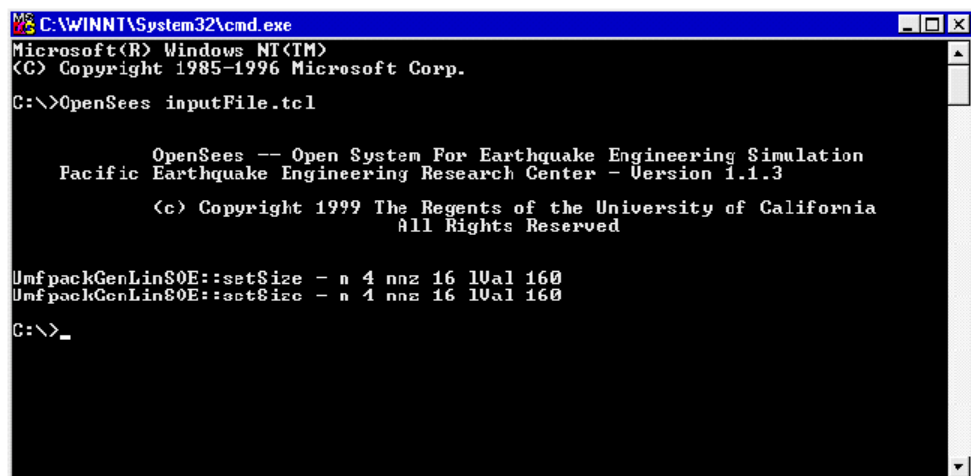
OpenSees > source inputFile.tcl
UmfpackGenLinS0E::setSize - n 4 nnz 16 lUal 160
UmfpackGenLinS0E::setSize - n 4 nnz 16 lUal 160
OpenSees > _

```

Figure 38: Run
OpenSees -- Source
File

Batch Mode

The previously-created input file containing the Tcl script commands necessary to execute the analysis can also be executed at the MS-DOS/Unix prompt, as shown in the figure (Win32 version):



```

C:\WINNT\System32\cmd.exe
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>OpenSees inputFile.tcl

OpenSees -- Open System For Earthquake Engineering Simulation
Pacific Earthquake Engineering Research Center - Version 1.1.3

(c) Copyright 1999 The Regents of the University of California
All Rights Reserved

UmfpackGenLinS0E::setSize - n 4 nnz 16 lUal 160
UmfpackGenLinS0E::setSize - n 4 nnz 16 lUal 160
C:\>_

```

Figure 39: 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:

```
# ----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 26

...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.

```
# -----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 function/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).

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 executed 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 preceded 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 $PI*pow($Rcol,2)]; # column cross-sectional area
set cover     [expr $Hcol/15];   # column cover width
set IgCol     [expr $PI*pow($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 aggregating 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 displacement 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 ground 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 155). These commands are typically placed at the beginning of the input file, after the header remarks.

```

# construct model builder using the model Command (page 21)
wipe; # clear data from past analysis
model basic -ndm 3 -ndf 6; # modelbuilder: basic (page 21), ndm= no. dimensions, ndf= no.

```

```

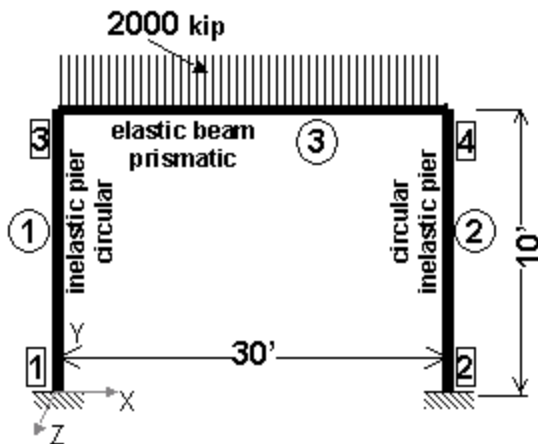
# Define nodes ----- frame is in X-Y plane (X-horizontal, Y-vertical) using the node Command (page 22)
node 1 0. 0. 0.; # base of left column
node 2 360. 0. 0.; # base of right column
node 3 0. 120. 0.; # top of left column
node 4 360. 120. 0.; # top of right column

# Define Boundary Conditions and nodal mass using the fix Command (page 24) ! 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 23):
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 155) -- 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 22)
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 24) ! 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 23):
mass 4 $Mass 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.

...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 $seps1C $fc2C $seps2C; # Core concrete
    uniaxialMaterial Concrete01 $IDcover $fc1U $seps1U $fc2U $seps2U; # Cover concrete
}
# STEEL
if {$SteelMaterialType=="elastic"} {
    uniaxialMaterial Elastic $IDsteel $Es
}
if {$SteelMaterialType=="bilinear"} {
    uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs
}
if {$SteelMaterialType=="hysteretic"} {

```

```

        uniaxialMaterial Hysteretic $IDsteel $Fy $sepsY $Fy1 $sepsY1 $Fu $sepsU -$Fy -$sepsY -$Fy1 -$sepsY1 -
        $Fu -$sepsU $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 $IDctrlNode -dof 1
}; # end of dynamic analysis

```

...Define Data-Plot During Analysis

```

# -----RecorderPlot.tcl-----
set pfile "Data/node.out";
recorder Node $pfile disp -time -node $IDctrlNode -dof 1
set title PushFrame$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 NormDisplncr 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 NormDisplncr 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 attempts

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 $DmaxPush
    set controlDisp 0.0;
    test NormDisplncr 1.0e-8 20 0
    while {$controlDisp < $maxU && $ok == 0} {
        set ok [analyze 1]
        set controlDisp [nodeDisp $IDctrlNode 1]
        if {$ok != 0} {
            puts "Trying Newton with Initial Tangent .."
            test NormDisplncr 1.0e-8 1000 1
            algorithm Newton -initial
        }
    }
}

```

```

        set ok [analyze 1]
        test NormDisplncr 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 NormDisplncr 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

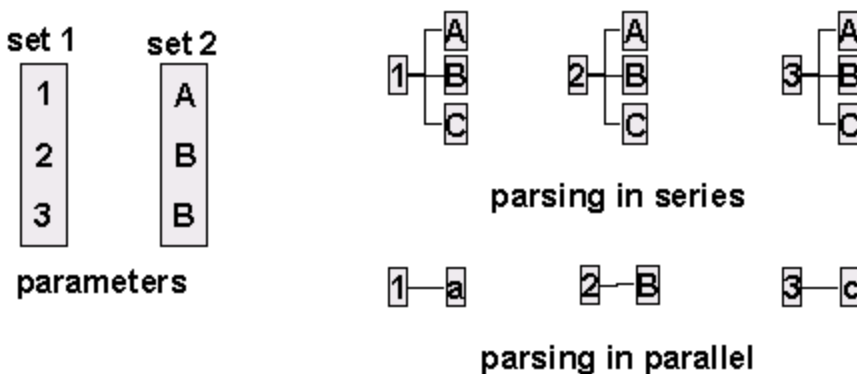
```

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 response):

```

# -----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
# 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; # 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 138) 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 27

Questions/Things missing

 *recorder Element 202 -time -file Data/mat202.out defoANDforce*

I have no documentation on defoANDforce

 *integrator LoadControl [expr 1./\$GravSteps] 1 [expr 1./\$GravSteps] [expr 1./\$GravSteps]*

I have that the last 3 arguments are optional, but they are not. -- they are in the future version.

Index

-
- ...Build Model and Define Nodes • 21, 22, 23, 25, 154
- ...Build Model and Define Nodes using Variables • 155
- ...Combine Input-File Components • 163
- ...Define Analysis-Output Generation • 158
- ...Define Data-Plot During Analysis • 159
- ...Define Dynamic Ground-Motion Analysis • 162
- ...Define Elements • 157
- ...Define Gravity Loads • 159
- ...Define Materials • 156
- ...Define Static Pushover Analysis • 160
- ...Define Tcl Procedure • 149
- ...Define Units & Constants • 148
- ...Define Variables and Parameters • 152
- ...Determine Natural Period & Frequency • 167
- ...Generate Matlab Commands • 149
- ...Read External files • 151
- ...Run Dynamic Ground-Motion Analysis • 163
- ...Run Gravity Analysis • 160
- ...Run Moment-Curvature Analysis on Section • 165
- ...Run OpenSees • 146
- ...Run Parameter Study • 164
- ...Run Static Pushover Analysis • 161

A

- algorithm Command • 19, 112, 113, 114, 126, 129, 142
- analysis Command • 19, 112, 121, 122, 137, 142
- Analysis Object • 18, 19, 112
- analyze Command • 137
- Arc-Length Control • 113, 114, 120, 122

B

- BandGeneral SOE • 135
- BandSPD SOE • 135

- Basic Model Builder • 10, 21, 22, 23, 56, 58, 59, 60, 61, 99, 154
- Bbar Brick Element • 46, 79, 87
- Bbar Plane Strain Quadrilateral Element • 46, 76, 85
- Beam With Hinges Element • 70, 71, 98
- BFGS Algorithm • 128
- Bidirectional Section • 66
- block Command • 17, 22, 84
- block2D Command • 84, 85
- block3D Command • 84, 87
- Bond01 Material • 44
- Bond02 Material • 45
- Brick Elements • 77
- Broyden Algorithm • 128
- build Command • 22
- Building The Model • 152

C

- Circular Layer Command • 55, 61
- Circular Patch Command • 59
- Concrete01 Material • 37, 41
- Concrete02 Material • 41
- Concrete03 Material • 42
- Constant Time Series • 102
- Constraints • 24
- constraints Command • 17, 19, 22, 106, 112, 113, 114, 115, 142
- Copyright • 11
- Corotational Transformation • 92
- Corotational Truss Element • 68

D

- dataBase Commands • 139
- Defining Output • 158
- Displacement Control • 112, 120, 121
- Displacement-Based Beam-Column Element • 70, 72, 98
- Display Recorder • 99, 144
- Domain Object • 18, 95, 96, 97, 106, 142
- Drift Recorder • 96
- Dynamic Analysis • 162

E

eigen Command • 138, 167
Eight Node Brick Element • 80
Elastic Beam Column Element • 69, 98
Elastic Isotropic Material • 46
Elastic Material • 29
Elastic Membrane Plate Section • 65
Elastic Section • 53
Elastic-No Tension Material • 38
Elastic-Perfectly Plastic Gap Material • 31
Elastic-Perfectly Plastic Material • 30
eleLoad Command • 106, 108
element Command • 17, 22, 67, 98, 142
Element Recorder • 29, 46, 52, 68, 69, 71, 72, 73, 74, 75, 76, 77, 80, 81, 83, 96
Energy Increment Test • 131
Enhanced Strain Quadrilateral Element • 46, 77, 85
EPState • 48, 51
equalDOF Command • 27
Evolution Law • 48, 50
Example Tcl Commands • 14

F

Fedeas Materials • 41
Fiber Command • 55, 56, 98
Fiber Section • 55
FileDataStore Command • 139
Fix Command • 24, 155, 156
fixX Command • 25
fixY Command • 25
fixZ Command • 26

G

Geometric Transformation Commands • 17, 22, 69, 70, 72, 89
getTime Command • 143
Gravity Loads • 159
groundMotion Command • 101, 102, 109, 110, 111

H

Hardening Material • 35
Hilbert-Hughes-Taylor • 113, 114, 120, 124
How To.... • 145
Hysteretic Material • 39

I

imposedMotion Command • 109, 111

integrator Command • 19, 110, 112, 113, 114, 119

Interpolated GroundMotion • 110

Introduction • 9

J

J2 Plasticity Material • 47

K

Krylov-Newton Algorithm • 128

L

Lagrange Multipliers • 117
Linear Algorithm • 126
Linear Time Series • 102, 120
Linear Transformation • 89, 91, 92
load Command • 106, 107
Load Control • 112, 113, 120
loadConst Command • 143

M

Mass Command • 17, 22, 23, 155, 156
MaxNodeDisp Recorder • 95
Minimum Unbalanced Displacement Norm • 112, 120, 122
Miscellaneous Commands • 141
Model Building Commands • 21
Model Command • 21, 26, 27, 28, 92, 93, 95, 96, 107, 108, 111, 121, 143, 154
ModelBuilder Object • 17, 18, 22, 24, 25, 69, 71
Modified Newton Algorithm • 127
MultipleSupport Pattern • 109, 111
Multi-Point Constraints • 27

N

nDMaterial Command • 17, 22, 46, 75, 76, 77, 78, 79, 81, 142
Newmark Method • 113, 114, 120, 123
Newton Algorithm • 112, 113, 114, 126
Newton with Line Search Algorithm • 127
Node Command • 10, 17, 22, 95, 96, 142, 155
Node Recorder • 94, 138
nodeDisp Command • 143
Nonlinear Beam Column Element • 70, 98
NonLinear Beam-Column Elements • 53, 70
Norm Displacement Increment T • 130
Norm Unbalance Test • 112, 113, 114, 129
Notation • 9

numberer Command • 19, 112, 113, 114, 132

O

Open System for Earthquake Engineering Simulation • 16

OpenSees Interpreter • 15, 142

P

Parallel Material • 32

Path Time Series • 105

pattern Command • 17, 22, 101, 102, 103, 104, 105, 106, 143

P-Delta Transformation • 91

Penalty Method • 116

Plain Constraints • 112, 113, 114, 116

Plain GroundMotion • 110

Plain Numberer • 132

plain Pattern • 106

Plane Stress Material • 47

Plate Fiber Material • 48, 65

Plate Fiber Section • 65, 76

play Command • 144

playback Command • 100

Plot Recorder • 99

Potential Surface • 48, 49

print Command • 138, 141

ProfileSPD SOE • 112, 113, 114, 135

Q

Quad Element • 46, 75, 85

Quadrilateral Elements • 75

Quadrilateral Patch Command • 55, 57

Questions/Things missing • 169

R

RCM Numberer • 112, 113, 114, 132, 133

recorder Command • 20, 94, 100

Recorder Object • 18, 20

Rectangular Time Series • 103

reset Command • 142

restore Command • 140

rigidDiaphragm Command • 27

rigidLink Command • 28

S

save Command • 139, 140

Section Aggregator • 62

section Command • 17, 22, 52, 62, 63, 67, 68, 70, 71, 72, 74, 76, 98

Series Material • 33

Shell Element • 46, 76, 85

Sine Time Series • 104

Single-Point Constraints • 24

sp Command • 106, 108

SparseGeneral SOE • 135

SparseSPD SOE • 136

Standard Brick Element • 46, 77, 87

Static Analysis • 19, 112, 120, 160

Steel01 Material • 36, 41

Steel02 Material • 44

Straight Layer Command • 55, 60

system Command • 19, 112, 113, 114, 129, 130, 131, 134, 142

T

Tcl Basics • 11

Tcl Commands • 12, 15

Template Elasto-Plastic Material • 48

test Command • 129

Time Series • 17, 22, 101, 102, 103, 104, 105, 106, 108, 110

Transformation Method • 117

Transient Analysis • 19, 113, 120, 137

Truss Element • 67

Twenty Node Brick Element • 81

U

UmfPack SOE • 136

Uniaxial Section • 54

uniaxialMaterial Command • 17, 22, 29, 32, 33, 54, 55, 56, 57, 59, 60, 61, 62, 63, 67, 68, 73, 142

UniformExcitation Pattern • 108

u-p-U element • 82

V

VariableTransient Analysis • 19, 113, 137

video Command • 144

Viscous Material • 40

W

wipe Command • 142

wipeAnalysis Command • 142

Y

Yield Surface • 48, 49

Z

Zero-Length Element • 73

Zero-Length Elements • 73

Zero-Length Section Element • 74