OpenSEES
Model-Building Commands

Silvia Mazzoni
University of California, Berkeley

OpenSEES MiniWorkshop
Hybrid Simulation Workshop
Richmond, December 2005
Tcl & OpenSEES commands

- Command syntax:
  \begin{verbatim}
  command arg1 arg2 ...;  # comment
  \end{verbatim}

  example Tcl command:
  \begin{verbatim}
  set a 1;                 # assign value of 1 to a
  set b [expr 2*$a];
  \end{verbatim}

  example OpenSEES command:
  \begin{verbatim}
  node 1 10. 10. -mass 10 0 0
  \end{verbatim}
Using Variables in Tcl -- UNITS

- set in 1.; # define basic units
- set sec 1.; # define basic units
- set kip 1.; # define basic units
- set ft [expr 12.*$in]; # define engineering units
- set ksi [expr $kip/pow($in,2)];
- set psi [expr $ksi/1000.];
- set in2 [expr $in*$in]; # inch^2
- set in4 [expr $in*$in*$in*$in]; # inch^4
- set PI [expr 2*asin(1.0)]; # define constants
- set g [expr 32.2*$ft/pow($sec,2)]; # grav. acceleration
- set Lcol [expr 36*$ft]; # column length
- set Dcol [expr 6.5*$ft]; # circular-column Diameter
- node 1 0 0
- node 2 0 $Lcol
- set Weight [expr 1000*$kip]; # weight
- set Mass [expr $Weight/$g]; # mass
- mass 2 $Mass 0 0; # assign mass to node
ModelBuilder Objects

- model Command
- node Command
- mass Command
- Constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- element Command
- block Command
- region Command
- Geometric Transformation Command
- Time Series
- pattern Command
### model-building commands

<table>
<thead>
<tr>
<th>Command</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>model Command</td>
<td>analysis.tcl</td>
</tr>
<tr>
<td>node Command</td>
<td>nodalmesh.tcl</td>
</tr>
<tr>
<td>mass Command</td>
<td></td>
</tr>
<tr>
<td>Constraints objects</td>
<td></td>
</tr>
<tr>
<td>uniaxialMaterial Command</td>
<td>materials.tcl</td>
</tr>
<tr>
<td>nDMaterial Command</td>
<td></td>
</tr>
<tr>
<td>section Command</td>
<td>elements.tcl</td>
</tr>
<tr>
<td>element Command</td>
<td></td>
</tr>
<tr>
<td>Geometric Transformation</td>
<td></td>
</tr>
</tbody>
</table>
http://opensees.berkeley.edu
or, OpenSeesManual.chm
sample command

**node Command**

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

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

- **$nodeTag**: integer tag identifying node
- **$coords**: nodal coordinates (*ndm* arguments)
- **$MassValues**: nodal mass corresponding to each DOF (*ndf* 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.
nodes and boundary conditions

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

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

fix $nodeTag (ndf $ConstrValues)

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

1. # Define nodes; # frame is in X-Y plane
2. node 1 0.0 0.0 0.0
3. node 2 $Lbeam 0.0 0.0
4. node 3 0.0 $Lcol 0.0 -mass $Mnode 0.0 0.0 0.0 0.0 0.0
5. node 4 $Lbeam $Lcol 0.0 -mass $Mnode 0.0 0.0 0.0 0.0 0.0

6. # Boundary conditions; # node DX DY DZ RX RY RZ ! 1: fixed, 0: released
7. fix 1 1 1 1 1 1 1;
8. fix 2 1 1 1 1 1 1
9. fix 3 0 0 1 1 1 0
10. fix 4 0 0 1 1 1 0

11. #
12. #
13. #
14. #

3------------------------4
|   |   |
|   |   |
--1-- --2--
materials

**uniaxialMaterial Elastic** $matTag$ E <$\eta$>

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

**uniaxialMaterial Concrete01** $matTag$ $f_{pc}$ $\varepsilon_{spc0}$ $f_{pcu}$ $\varepsilon_{spcu}$ $\varepsilon_{spcu}$

- **$matTag$** unique material object integer tag
- **$f_{pc}$** compressive strength*
- **$\varepsilon_{spc0}$** strain at compressive strength*
- **$f_{pcu}$** crushing strength*
- **$\varepsilon_{spcu}$** strain at crushing strength*
tcl if statement

if {logical statement} {
    ....series of commands.....
}

```
set ConcreteMaterialType "inelastic" # options: "elastic","inelastic"

if {$ConcreteMaterialType == "elastic"} {
    uniaxialMaterial Elastic $IDcore $Ec
    uniaxialMaterial Elastic $IDcover $Ec
}

if {$ConcreteMaterialType == "inelastic"} {
    # uniaxial Kent-Scott-Park concrete model w/ linear unload/reload, no T strength (-ve comp.)
    uniaxialMaterial Concrete01 $IDcore $fc1C $eps1C $fc2C $eps2C; # Core
    uniaxialMaterial Concrete01 $IDcover $fc1U $eps1U $fc2U $eps2U; # Cover
}
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\texttt{matTag}$</td>
<td>unique material object integer tag</td>
</tr>
<tr>
<td>$s1p$</td>
<td>stress and strain (or force &amp; deformation) at first point of the envelope</td>
</tr>
<tr>
<td>$e1p$</td>
<td>in the positive direction</td>
</tr>
<tr>
<td>$s2p$</td>
<td>stress and strain (or force &amp; deformation) at second point of the envelope</td>
</tr>
<tr>
<td>$e2p$</td>
<td>in the positive direction</td>
</tr>
<tr>
<td>$s3p$</td>
<td>stress and strain (or force &amp; deformation) at third point of the envelope</td>
</tr>
<tr>
<td>$e3p$</td>
<td>in the positive direction</td>
</tr>
<tr>
<td>$s1n$</td>
<td>stress and strain (or force &amp; deformation) at first point of the envelope</td>
</tr>
<tr>
<td>$e1n$</td>
<td>in the negative direction</td>
</tr>
<tr>
<td>$s2n$</td>
<td>stress and strain (or force &amp; deformation) at second point of the envelope</td>
</tr>
<tr>
<td>$e2n$</td>
<td>in the negative direction</td>
</tr>
<tr>
<td>$s3n$</td>
<td>stress and strain (or force &amp; deformation) at third point of the envelope</td>
</tr>
<tr>
<td>$e3n$</td>
<td>in the negative direction</td>
</tr>
<tr>
<td>$\texttt{pinchX}$</td>
<td>pinching factor for strain (or deformation) during reloading</td>
</tr>
<tr>
<td>$\texttt{pinchY}$</td>
<td>pinching factor for stress (or force) during reloading</td>
</tr>
<tr>
<td>$\texttt{damage1}$</td>
<td>damage due to ductility: $D_1(\mu-1)$</td>
</tr>
<tr>
<td>$\texttt{damage2}$</td>
<td>damage due to energy: $D_2\left(\frac{E_{ii}}{E_{ult}}\right)$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>power used to determine the degraded unloading stiffness based on ductility, $\mu^{-\beta}$ (optional, default=0.0)</td>
</tr>
</tbody>
</table>
1. set SteelMaterialType "hysteretic";

2. if {$SteelMaterialType == "elastic"} {
   uniaxialMaterial Elastic $IDsteel $Es
}

3. if {$SteelMaterialType == "bilinear"} {
   uniaxialMaterial Steel01 $Idsteel $Fy $Es $Bs
}

4. }

5. if {$SteelMaterialType == "hysteretic"} {
   uniaxialMaterial Hysteretic $IDsteel $Fy $epsY $Fy1 $epsY1 $Fu $epsU -$Fy -$epsY -$Fy1 -$epsY1 -$Fu -$epsU $pinchX $pinchY $damage1 $damage2 $betaMUsteel
}

6. }

7. else {
   uniaxialMaterial $IDsteel $Es
}

8. }
tcl procedure

proc procName {input variables} {
    ... series of commands
}

to execute:

procName (input variables)
**fiber section command**

```
section Fiber $secTag {
  fiber <fiber arguments>
  patch <patch arguments>
  layer <layer arguments>
}
```

```
fiber $yLoc $zLoc $A $matTag
```

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

- **$zLoc**  
  z coordinate of the fiber in the section (local coordinate system)

- **$A**  
  area of fiber

- **$matTag**  
  material tag of the pre-defined `UniaxialMaterial` object used to represent the stress-strain for the area of the fiber
section command (cont.)

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

- **$matTag** material integer tag of the previously-defined `UniaxialMaterial` object used to represent the stress-strain for the area of the fiber
- **$numSubdivCirc** number of subdivisions (fibers) in the circumferential direction.
- **$numSubdivRad** number of subdivisions (fibers) in the radial direction.
- **$yCente** y & z coordinates of the center of the section
- **$zCente** y & z coordinates of the center of the section
- **$intRad** internal radius
- **$extRad** external radius
- **$startAng** starting angle (optional. default=0)
- **$endAng** ending angle (optional. default=360)
section command (cont.)

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

$matTag
material integer tag of the previously-defined
UniaxialMaterial 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

$startAn $endAn
starting and ending angle of
reinforcing layer, respectively
(Optional, Default: a full circle is assumed 0-360)
tcl proc: define fiber section

```
proc RCcircSection {id Ri Ro coverID coreID coverID steelID Nbars Ab 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; # Define the core patch
    patch circ $coverID $nfCoverT $nfCoverR 0 0 $Rc $Ro 0 360; # Define the cover patch
    if {$Nbars<= 0} { return }set theta [expr 360.0/$Nbars]; # angle increment between bars
    layer circ $steelID $Nbars $Ab 0 0 $Rc $theta 360; # Define the reinforcing layer
  }
}
```
section aggregator

• groups previously-defined UniaxialMaterial objects into a single section force-deformation model

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

$secTag  unique section object integer tag
$matTag1, $matTag2 ... previously-defined UniaxialMaterial objects
$string1, $string2 .... the force-deformation quantities corresponding to each section object. One of the following strings is used:

P  Axial force-deformation
Mz  Moment-curvature about section local z-axis
Vy  Shear force-deformation along section local y-axis
My  Moment-curvature about section local y-axis
Vz  Shear force-deformation along section local z-axis
T  Torsion Force-Deformation

<-section $sectionTag> specifies a previously-defined Section object (identified by the argument $sectionTag) to which these UniaxialMaterial objects may be added to recursively define a new Section object.
geometric transformation

• performs a linear geometric transformation of beam stiffness and resisting force from the basic system to the global-coordinate system

```
geomTransf Linear $transfTag $vecxzX $vecxzY $vecxzZ <-jntOffset $dXi $dYi $dZi $dXj $dYj $dZj>
```

- **$transfTag** unique identifier for CrdTransf object
- **$vecxzX** X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. The local y-axis is defined by taking the cross product of the x-axis and the vecxz vector. These components are specified in the global-coordinate system X,Y,Z and define a vector that is in a plane parallel to the x-z plane of the local-coordinate system. These items need to be specified for the three-dimensional problem.
- **$dXi** $dYi** $dZi** joint offset values -- absolute offsets specified with respect to the global coordinate system for element-end node i (the number of arguments depends on the dimensions of the current model) (optional)
- **$dXj** $dYj** $dZj** joint offset values -- absolute offsets specified with respect to the global coordinate system for element-end node j (the number of arguments depends on the dimensions of the current model) (optional)
local coordinate system

node i

node j

local xz plane

vector parallel to vecxz

vecxz (vecxzX, vecxzY, vecxzZ)
elements

- Truss Element
- Corotational Truss Element
- Elastic Beam Column Element
- NonLinear Beam-Column Elements
  - Nonlinear Beam Column Element
  - Beam With Hinges Element
  - Displacement-Based Beam-Column Element
- Zero-Length Elements
- Quadrilateral Elements
- Brick Elements
- FourNodeQuadUP Element
- BeamColumnJoint Element
Elastic Beam Column Element

- **2D:**
  
  ```
  element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag
  ```

- **3D:**
  
  ```
  element elasticBeamColumn $eleTag $iNode $jNode $A $E $G $J $Iy $Iz $transfTag
  ```
Nonlinear Beam Column Element

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

- `$eleTag`: unique element object tag
- `$iNode` $`$jNode`: end nodes
- `$numIntgrPts`: number of integration points along the element.
- `$secTag`: identifier for previously-defined `section` object
- `$transfTag`: identifier for previously-defined `coordinate-transformation` (CrdTransf) object
- `$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})*
1. set ColumnType “inelastic”;
2. source RCcircSection.tcl; # proc to define circular fiber section—flexure
3. RCCircSection $IDcolFlex $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol $AbCol $nfCoreR $nfCoreT $nfCoverR $nfCoverT
4. uniaxialMaterial Elastic $IDcolTors $GJ; # Define torsion
5. section Aggregator $IDcolSec $IDcolTors T -section $IDcolFlex; # attach torsion & flex
6. geomTransf Linear $IDcolTrans 0 0 1; # no 2nd-order effects, define element normal
7. if {$ColumnType == “elastic”} {
8. element elasticBeamColumn 1 1 3 $Acol $Ec $G $J $IyCol $IzCol $IDcolTrans
9. element elasticBeamColumn 2 2 4 $Acol $Ec $G $J $IyCol $IzCol $IDcolTrans }
10. if {$ColumnType == “inelastic”} {
11. # element element type ID, node I, node J, no. int pts, section ID, transf. ID
12. element nonlinearBeamColumn 1 1 3 $np $IDcolSec $IDcolTrans
13. element nonlinearBeamColumn 2 2 4 $np $IDcolSec $IDcolTrans }
14. geomTransf Linear $IDbeamTrans 0 0 1; # BEAM transformation, define element normal
15. element elasticBeamColumn 3 3 4 $Abeam $Ec $G $J $IyBeam $IzBeam $IDbeamTrans
region command

- label a group of nodes and elements.
- This command is also used to assign rayleigh damping parameters to the nodes and elements in this region.
- The region is specified by either elements or nodes, not both. If elements are defined, the region includes these elements and the all connected nodes. If nodes are specified, the region includes these nodes and all elements whose external nodes are prescribed.

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

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

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

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

$regTag
$node1 $node2 ...
$startNode $endNode
all
$alphaM $betaK
$betaKinit
$betaKcomm

unique integer tag
node tags -- select nodes in domain
(optional, default: all)
tag for start and end nodes -- range of
nodes in domain (optional, default: all)
all nodes in domain (optional & default)
Arguments to define Rayleigh damping matrix (optional, default: zero)
region command (damping)

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

\[ C = \alphaM M + \betaK K + \betaKinit K_{initial} + \betaKcomm K_{committed} \]

- \( M \): mass matrix used to calculate Rayleigh Damping
- \( K_{current} \): stiffness matrix at current state determination used to calculate Rayleigh Damping
- \( K_{init} \): stiffness matrix at initial state determination used to calculate Rayleigh Damping
- \( K_{lastCommit} \): stiffness matrix at last-committed state determination used to calculate Rayleigh Damping
Recorder Objects

- Node Recorder
- EnvelopeNode Recorder
- MaxNodeDisp Recorder
- Drift Recorder
- Element Recorder
- EnvelopeElement Recorder
- Display Recorder
- Plot Recorder
- playback Command
recorder Node <-file $fileName> <-time> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-region $RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType

$fileName file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)

-time this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)

$node1 $node2 ... tags nodes where response is being recorded -- select nodes in domain (optional, default: all)

$startNode $endNode tag for start and end nodes where response is being recorded -- range of nodes in domain (optional, default: all)

$RegionTag tag for previously-defined selection of nodes defined using the Region command. (optional)

all where response is being recorded -- all nodes in domain (optional & default)

$dof1 $dof2 ... degrees of freedom of response being recorded. Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

$respType defines response type to be recorded. The following response types are available:
disp displacement
vel velocity
accel acceleration
incrDisp incremental displacement
eigen eigenvector

Silvia Mazzoni OpenSEES mini-Workshop 2005
recorder EnvelopeNode <-file $fileName> <-time> <-node ($node1 $node2 ...) <-nodeRange $startNode $endNode> <-region $RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType

records the envelope of displacement, velocity, acceleration and incremental displacement at the nodes (translational & rotational). The envelope consists of the following: minimum, maximum and maximum absolute value of specified response type

$fileName

file where results are stored. Each line of the file contains the result for a committed state of the domain (optional, default: screen output)

-time
this argument will place the pseudo time of the as the first entry in the line. (optional, default: omitted)

$node1

tags nodes where response is being recorded -- select nodes in domain (optional, default: all)

$node2 ...

$startNode
tag for start and end nodes where response is being recorded -- range of nodes in domain (optional, default: all)

$endNode

$RegionTag
tag for previously-defined selection of nodes defined using the Region command. (optional)

all

where response is being recorded -- all nodes in domain (optional & default)

$dof1

degrees of freedom of response being recorded.

$dof2 ...

Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

..... same arguments as node recorder
recorder MaxNodeDisp $dof $node1 $node2
...

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

$dof$ displacement degree-of-freedom direction. Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

$node1$ nodes where maximum displacement is being recorded

$node2$ ...
recorder Element <-file $fileName> <-time> <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-region $regTag> <-ele all> ($arg1 $arg2 ...)
element recorder (output arguments)

All:

globalForce - element resisting force in global coordinates (does not include inertial forces)
recorder Element -file ele1global.out -time -ele 1 globalForce

localForce - element resisting force in local coordinates (does not include inertial forces)
recorder Element -file ele1local.out -time -ele 1 localForce

Section:

section $secNum - request response quantities from a specific section along the element length
$secNum refers to the integration point whose data is to be output

force - section forces
example: recorder Element -file ele1sec1Force.out -time -ele 1 section 1 force
defformation - section deformations
example: recorder Element -file ele1sec1Force.out -time -ele 1 section 1 deformation
stiffness - section stiffness
example: recorder Element -file ele1sec1Force.out -time -ele 1 section 1 stiffness
stressStrain - record stress-strain response.
example: recorder Element -file ele1sec1Force.out -time -ele 1 section 1 fiber $y$ $z$
stressStrain

$y$ local y coordinate of fiber to be monitored*
$z$ local z coordinate of fiber to be monitored*
# Record nodal displacements -NODAL DISPLACEMENTS
# ALL displacements at node 1
recorder Node -file Dnode1.out -time -node 1 -dof 1 2 3 disp;

# Record vertical-y displacement of ALL nodes
recorder Node -file DNodeALL.out -time -node all -dof 2 disp;

# Record REACTION FORCES - (=forces in element 1)
recorder Element -file Fel1.out -time -ele 1 localForce
Loads - pattern command

```java
pattern Plain $patternTag (TimeSeriesType arguments) {
  load (load-command arguments)
  sp (sp-command arguments)
  eleLoad (eleLoad-command arguments)
}
```

- **$patternTag**: unique pattern object tag
- **TimeSeriesType arguments**: list which is parsed to construct the `TimeSeries` object associated with the LoadPattern object.
- **load ...**: list of commands to construct nodal loads -- the `NodalLoad` object
- **sp ...**: list of commands to construct single-point constraints -- the `SP_Constraint` object
- **eleLoad ...**: list of commands to construct element loads -- the `eleLoad` object
pattern command (cont.)

load $nodeTag (ndf $LoadValues)

$nodeTag node on which loads act
$LoadValues load values that are to be applied to the node. Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

sp $nodeTag $DOFtag $DOFvalue

$nodeTag node on which the single-point constraint acts
$DOFtag degree-of-freedom at the node being constrained. Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.
$DOFvalue reference value of the constraint to be applied to the DOF at the node.

pattern Plain 1 Linear {
    #       Fx   Fy  Fz  Mx  My  Mz
    load 3  0.0  -$Pdl 0.0 0.0 0.0  0.0  -$Mdl
    load 4  0.0  -$Pdl 0.0 0.0 0.0  0.0  +$Mdl
}

Silvia Mazzoni OpenSEES mini-Workshop 2005
Questions, or statements!

The OpenSees Community Forum:
http://opensees.berkeley.edu/phpBB2/index.php

which can be accessed from:
http://opensees.berkeley.edu
thank you!!!