Structural Modeling
With Examples

Silvia Mazzoni, PhD

Structural Consultant
Degenkolb Engineers

OpenSees Days 2010

Tcl & OpenSees commands

• Command syntax:
  command arg1 arg2 ...; # comment

example Tcl command:
  set a 1; # assign value of 1 to a
  set b [expr 2*$a];
example OpenSees command:
  node 1 10. 10. -mass $Mnode 0 0
ModelBuilder Objects

- model Command
  - node Command
  - mass Command
  - constraints objects
  - uniaxialMaterial Command
  - nDMaterial Command
  - section Command
  - Geometric Transformation Command
  - element Command
  - block Command
  - region Command
  - Time Series
  - pattern Command

Model Command

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)

```
model BasicBuilder -ndm 3 -ndf 6
```
ModelBuilder Objects

- model Command
- node Command
- mass Command
- constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- Geometric Transformation Command
- element Command
- block Command
- region Command
- Time Series
- pattern Command

Nodal Coordinates

Node command

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

```node $nodeTag (nrm $coords) <-mass (nrm $massValues)>
```

- $nodeTag: integer tag identifying node
- $coords: nodal coordinates (nrm arguments)
- $massValues: nodal mass corresponding to each DOF (nrm argument) (optional)

The optional -mass string slices allows the option of associating nodal mass with the node.

Example:

```node 1 0 0 0.0 0.0 0.0
node 2 00 120.0 0.0
```

Code Developed by: *tek*
sample command

```
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 analysts the option of associating nodal mass with the node.

nodes and boundary conditions

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

$nodeTag: integer tag identifying node
$coords: nodal coordinates (ndm arguments)

```
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
## Nodal coordinates and BC

1. # Define nodes;  
   node 1 0.0 0.0 0.0  
   node 2 100.0 0.0 0.0  
   node 3 0.0 $L_{col}$ 0.0 -mass 4.7 0.0 0.0 0.0 0.0 0.0 0.0  
   node 4 $L_{beam}$ $L_{col}$ 0.0 -mass $M_{node}$ 0.0 0.0 0.0 0.0 0.0  

2. # frame is in X-Y plane coordinates & mass  

3. # Boundary conditions;  
   fix 1 1 1 1 1 1 1;  
   fix 2 1 1 1 1 1 1  
   fix 3 0 0 1 1 1 0  
   fix 4 0 0 1 1 1 0  

4. # boundary conditions

---

## ModelBuilder Objects

- model Command  
- node Command  
- mass Command  
- constraints objects  
  - uniaxialMaterial Command  
- nDMaterial Command  
- section Command  
- Geometric Transformation Command  
- element Command  
- block Command  
- region Command  
- Time Series  
- pattern Command
uniaxialMaterials: stress-strain (force-deformation) behavior

**Core Materials**
- Elastic Material
- Elastic-Perfectly Plastic Material
- Elastic-Perfectly Plastic Gap Material
- Elastic-No Tension Material
- Parallel Material
- Series Material
- Hardening Material
- Concrete01 Material
- Concrete02 Material
- Concrete03 Material
- Steel01 Material
- Steel02 Material
- Hardening
- Hysteretic Material
- Viscous Material

**Contributed Materials**
- BARSLIP Material
- Bond_SP01
- Concrete04 Material - Popovics
- Concrete07 - Chang & Mander's
- Fatigue Material
- Hyperbolic Gap Material
- Limit State Material
- PINCHING4 Material
- PyTzQz Uniaxial Materials
- Reinforcing Steel Material
- SelfCentering Material

---

**materials**

*copy and paste from manual:*

```
uniaxialMaterial Elastic $matTag $E <$eta>
  $matTag  unique material object integer tag
  $E      tangent
  $eta    damping tangent (optional, default=0.0)

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.
```
tcl if statement

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

```tcl
set a 7
if {$a==7} {
    puts $a
}
```

materials

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

1. set ConcreteMaterialType "elastic" # options: "elastic","inelastic"

2. if {ConcreteMaterialType == "elastic"} {
3.    uniaxialMaterial Elastic $Idcore $Ec
4.    uniaxialMaterial Elastic $Idcover $Ec
5. } else {
6.    # ConcreteMaterialType == "inelastic"
7.    # uniaxial Kent-Scott-Park concrete model w/ linear unload/reload, no T strength (-ve comp)
8.    uniaxialMaterial Concrete01 $IDcore  $fc1C  $eps1C  $fc2C  $eps2C; # Core
9.    uniaxialMaterial Concrete01 $IDcover  $fc1U  $eps1U  $fc2U $eps2U; # Cover
10. }
```
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 $e2n**: stress and strain (or force & deformation) at second point of the envelope in the negative direction (optional)
- **$s3n $e3n**: stress and strain (or force & deformation) at third point of the envelope in the negative direction (optional)
- **$pinchX**: pinching factor for strain (or deformation) during reloading
- **$pinchY**: pinching factor for stress (or force) during reloading
- **$damage1**: damage due to ductility: $D_1(\mu-1)$
- **$damage2**: damage due to energy: $D_2(E_{ii}/E_{ult})$
- **$beta**: power used to determine the degraded unloading stiffness based on ductility, $\mu^{beta}$ (optional, default=0.0)

---

**materials**

1. set SteelMaterialType "hysteretic";
2. if ($SteelMaterialType == "elastic") {
3. uniaxialMaterial Elastic $IDsteel $E
4. }
5. elseif ($SteelMaterialType == "bilinear") {
6. uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs
7. }
8. else {
9. uniaxialMaterial Hysteretic $IDsteel $Fy $epsY $Fy1 $epsY1 $Fu $epsU -$Fy -$epsY -$Fy1 -$epsY1 -$Fu -$epsU $pinchX $pinchY $damage1 $damage2 $betaMUsteel
10. }

---

Silvia Mazzoni, OpenSees Days 2010
ModelBuilder Objects

- `model Command`
- `node Command`
- `mass Command`
- `constraints objects`
- `uniaxialMaterial Command`
- `nDMaterial Command` ← Presented in Geotech
- `section Command`
  - `Geometric Transformation Command`
  - `element Command`
  - `block Command`
  - `region Command`
  - `Time Series`
  - `pattern Command`

OpenSees Sections:

- Elastic Section
- Uniaxial Section
- Fiber Section
- Section Aggregator

- Elastic Membrane Plate Section
- Plate Fiber Section
- Bidirectional Section
- Isolator2spring Section: Model to include buckling behavior of an elastomeric bearing
Uniaxial Beam-Column Sections:

- **Elastic Section**
  - linear-elastic moment-curvature relationship

- **Uniaxial Section**
  - user-defined moment-curvature relationship (use uniaxialMaterial)
  - uncoupled P-M and anything else

- **Fiber Section**
  - user-defined section geometry/materials via fibers
  - coupled P-M interaction
  - coupled bi-directional response

- **Section Aggregator**
  - combine all uncoupled responses (e.g., Uniaxial flexure + Uniaxial Axial, Fiber flexure/axial + shear)

---

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.
- **$numSubdivRad**: number of subdivisions (fibers) in the radial direction.
- **$yCenter** and **$zCenter**: y & z-coordinates of the center of the patch.
- **$intRad** and **$extRad**: internal and external radius.
- **$startAng** and **$endAng**: starting and ending angle of the patch, respectively. (Optional, Default: a full circle is assumed 0-360)

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

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

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

*Silvia Mazzoni, OpenSees Days 2010*
tcl procedure

• used for repeated series of commands on a set of input variables

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

to execute:
```
procName (input variables)
```

```tcl
proc multiply {a b} {
  set c [expr $a*$b]
  return $c
}
```

```
set a 3; set b 5
set result [multiply $a $b]
```

---

tcl proc: define fiber section

```tcl
proc RCcircSection {id  Ri Ro cover 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
  }
}
```

```
RCcircSection $id $Ri $Ro cover $coreID coverID steelID $Nbars $Ab $nfCoreR $nfCoreT $nfCoverR $nfCoverT
```

Silvia Mazzoni, OpenSees Days 2010
**UniaxialSection**

- Use uniaxialMaterial to define section moment-curvature relationship
  - Hysteretic Material

**section aggregator**

- groups previously-defined uniaxialMaterial objects, or sections, 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>** specified 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.
ModelBuilder Objects

- model Command
- node Command
- mass Command
- constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- Geometric Transformation Command
- element Command
- block Command
- region Command
- Time Series
- pattern Command

geometric transformation

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

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

-Type Transformation Type: Linear, PDelta or Corotational
-transfTag unique identifier for CrdTransf object
-vecxzX ONLY IN 3D. 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 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)
-dZi
-dXj $dYj 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)
-dZj
local coordinate system

vecxz (vecxzX, vecxzY, vecxzZ)

ModelBuilder Objects

- model Command
- node Command
- mass Command
- constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- Geometric Transformation Command
  - element Command
    - block Command
    - region Command
    - Time Series
    - pattern Command
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

\[
element \text{nonlinearBeamColumn} \ <\ \ $eleTag \ $iNode \ $jNode \ $numIntgrPts \ $secTag \ $transfTag \ <\ -\mass \ $massDens \ <-\ iter \ $maxIters \ $tol> \ 
\]

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

---

**elements**

```
set ColumnType "inelastic";
source RCcircSection.tcl;                          # proc to define circular fiber section- flexure

RCcircSection $IDcolFlex  $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol  $AbCol  $nfCoreR $nfCoreT $nfCoverR $nfCoverT

uniaxialMaterial Elastic $IDcolTors $GJ;          # Define torsion

section Aggregator $IDcolSec $IDcolTors T -section $IDcolFlex; # attach torsion & flex

geomTransf Linear $IDcolTrans 0 0 1;              # no 2nd-order effects, define element normal

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
} else {

# $ColumnType == "inelastic"

# element element type  ID, node I, node J, no. int pts, section ID, transf. ID

element nonlinearBeamColumn 1 1 3 $np $IDcolSec $IDcolTrans

element nonlinearBeamColumn 2 2 4 $np $IDcolSec $IDcolTrans

geomTransf Linear $IDbeamTrans 0 0 1; # BEAM transformation, element normal

element elasticBeamColumn 3 3 4 $Abeam $Ec $G $J $IyBeam $IzBeam $IDbeamTrans
```

---

Silvia Mazzoni, OpenSees Days 2010
beamWithHinges Element

2D:

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

3D:

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

Silvia Mazzoni, OpenSees Days 2010

beamWithHinges Element

$eleTag unique element object tag
$iNode $jNode end nodes
$secTagI identifier for previously-defined section object corresponding to node I
$HingeLengthI hinge length at node I
$secTagJ identifier for previously-defined section object corresponding to node J
$HingeLengthJ hinge length at node J
$E Young's Modulus
$A area of element cross-section
$Iz section moment of inertia about the section local z-axis
$Iy section moment of inertia about the section local y-axis
$G Shear Modulus
$J torsional moment of inertia of cross section
$transfTag identifier for previously-defined coordinate-transformation (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)

Silvia Mazzoni, OpenSees Days 2010
ZeroLength Element

- DOF spring
- Use uniaxialMaterial to define “force”-“deformation” response.
  - Translational dof: force-deformation
  - Rotational dof: moment-rotation
  - e.g. Hysteric Material
- 1M applications

ModelBuilder Objects

- model Command
- node Command
- mass Command
- constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- Geometric Transformation Command
- element Command
- block Command
- region Command
- Time Series
- pattern Command
Loads - pattern command

(patternPlain $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.

(patternPlain 1 Linear {
  load 3 0.0 -$Pdl 0.0 0.0 0.0 -$Mdl
  load 4 0.0 -$Pdl 0.0 0.0 0.0 +$Mdl
  sp 1 2 -0.001
})
pattern command (cont.)

2D:
• Uniformly-distributed load:
  \texttt{eleLoad -ele \$eleTag1 \$eleTag2 \ldots \ -type -beamUniform \$Wz \$Wx}

• Point load:
  \texttt{eleLoad -ele \$eleTag1 \$eleTag2 \ldots \ -type -beamPoint \$Pz \$xL \$Px}

3D:
• Uniformly-distributed load:
  \texttt{eleLoad -ele \$eleTag1 \$eleTag2 \ldots \ -type -beamUniform \$Wy \$Wz \$Wx}

• Point load:
  \texttt{eleLoad -ele \$eleTag1 \$eleTag2 \ldots \ -type -beamPoint \$Py \$Pz \$xL \$Px}

pattern Plain 1 Linear { 
  sp 1 2 -0.001 
  \texttt{eleLoad -ele 3 -type -beamUniform [expr -$Weight/$LBeam]}
}

Structural Example – Reinforced-Concrete Frame:

Building the Model
Example 4 in examples manual

Model Building

- Reinforced-Concrete Portal Frame
- Use kip, inch and sec as basic units

Analysis
…..remember what we told you about Tcl?

- Tcl is a string based scripting language
- Enables variables and variable substitution (use variables to define units!!)
- Expression evaluation
- Array management
- Basic control structures (if, while, for, foreach)
- Procedures
- File manipulation

Tcl enables variables & variable substitution:

example Tcl command:

set a 1;
set b [expr 2*$a];

Define Units:

set in 1.0;
set ft [expr 12.*$in]
set Hcol [expr 3.*$ft + 6.*$in]
Ex4.Portal2D.build.InelasticFiberSection.tcl

# Example4 2D Portal Frame - Build Model
# nonlinearBeamColumn element, inelastic fiber section
# Silvia Mazzoni & Frank McKenna, 2006
#
# put unit definitions into a file
#
# wipe
#
# model BasisBuilder -ndm 2 -ndf 3
# define the model builder: ndm=dimension, ndf=ndef
# set dataDir: Data
# setup name of data directory
# file mkdir $dataDir
# create data directory
# ground-motion file directory
# source LibUnits.tcl
# define basic and system units

 LibUnits.tcl

# define UNITS -----------------------------
# define basic units -- output units
# define basic units -- output units
# define basic units -- output units
# define basic-unit text for output
# define basic-unit text for output
# define basic-unit text for output
# define basic-unit text for output
# define engineering units
# pounds force
# pounds per cubic foot
# pounds per square foot
# inch^2
# inch^4
# centimeter
# define constants
# gravitational acceleration
# a really large number
# a really small number

Silvia Mazzoni, OpenSees Days 2010
Geometry, Weight, Mass

28  # define GEOMETRY
29  set LCol [expr 36*$ft];   # column length
30  set LBeam [expr 42*$ft]; # beam length
31  # define section geometry
32  set HCol [expr 5.5*$ft]; # Column Depth
33  set BCol [expr 5.5*$ft]; # Column Width
34  set HBeam [expr 8.5*$ft]; # Beam Depth
35  set BBeam [expr 5.5*$ft]; # Beam Width
36  # superstructure weight
37  set Weight [expr 2000.*$kip];
38  # calculated parameters
39  set PCol [expr $Weight/2]; # nodal dead load weight
40  set Mass [expr $PCol/$g];  # nodal mass
41  set MC0l [expr 1./12.*(weight/$lBeam)^pow($LBeam,2)]; # beam-end moment due
c43  # calculated geometry parameters
44  set AC0l [expr $BCol*$HCol]; # cross-sectional area
45  set AB0am [expr $BBeam*$HBeam];
46  set Ic0l [expr 1./12.*$BC0l^pow($HCol,3)]; # Column moment of inertia
47  set IM0am [expr 1./12.*$BBeam^pow($HBeam,3)]; # Beam moment of inertia

Nodes: Coords, BC, Mass

49  # nodal coordinates:
50  node 1 0 0:  # node#, X, Y
51  node 2 $LBeam 0
52  node 3 0 $LCol
53  node 4 $LBeam $LCol
54  # Single point constraints -- Boundary Conditions
55  fix 1 1 1 0;  # node DX DY DZ
56  fix 2 1 1 0;  # node DX DY DZ
57  fix 3 0 0 0
58  fix 4 0 0 0
59  # nodal masses:
60  mass 3 $Mass 0 0;  # node
61  mass 4 $Mass 0 0.
Concrete Material: Concrete02

Concrete02 Material -- Linear Tension Softening
This command is used to construct a uniaxial concrete material object with tensile strength and linear tension softening.

uniaxialMaterial Concrete02 $matTag $fpc $epso0 $fpeu $epsoU $lambda $ft $Ets

$matTag unique material object integer tag
$fpc compressive strength*
$epso0 strain at compressive strength*
$fpeu crushing strength*
$epsoU strain at crushing strength*
$lambda ratio between unloading slope at $epsoU and initial slope
$ft tensile strength
$Ets tension softening stiffness (absolute value) (slope of tension softening branch)

*NOTE: Compressive concrete parameters should be input as negative values.
The initial slope for this model is (2*$fpc/$epso0)

Silvia Mazzoni, OpenSees Days 2010

Steel Material: Steel02

Steel02 Material -- Giuffrè-Menegotto-Pinto Model with Isotropic Strain Hardening
This command is used to construct a uniaxial Giuffrè-Menegotto-Pinto steel material object with isotropic strain hardening.

uniaxialMaterial Steel02 $matTag $Fy $E $b $R0 $R1 $R2 <$a1 $a2 $a3 $a4 $sigInt>

$matTag unique material object integer tag
$Fy yield strength
$E initial elastic tangent
$b strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
$R0, $R1, $R2 control the transition from elastic to plastic branches.
Recommended values:
$R0 between 10 and 20, $R1 = 0.925, $R2 = 0.15
$a1, $a2, $a3, $a4 isotropic hardening parameters (optional, default no isotropic hardening)

Silvia Mazzoni, OpenSees Days 2010
Materials

```plaintext
# MATERIAL parameters
set IDconcu 1;  # material ID tag -- unconfined
set IDreinf 2;  # material ID tag -- reinforcement
# nominal concrete compressive strength
set fc  [expr -4.0*$ksi];  # CONCRETE Compressive Stren
set Ec [expr 57*$ksi*exp(-$fcs/$psi)];  # Concrete Elastic Modulus
# unconfined concrete
set fcu $fc;
set epscu -0.003;  # strain at maximum strength of
set fcu [expr 0.2*$fcu];  # ultimate stress
set epscu -0.005;  # strain at ultimate stress
set lambda 0.1;  # ratio between unloading slope
set fhu [expr -0.14*$fcu];  # tensile strength tension
set Ecu [expr $fhu/0.002];  # tension softening stiffness
# ---------
set fy [expr 66.8*$ksi];  # STEEL yield stress
set Ex [expr 29000*$ksi];  # modulus of steel
set Eo 0.1;  # strain-hardening ratio
set ro 18;  # control the transition from elastic to plastic branches
set s1 0.925;  # control the transition from elastic to plastic branches
set s2 0.15;  # control the transition from elastic to plastic branches
# build cover concrete
set IDconcu 1;  # building concrete
set IDreinf 2;  # building reinforcement
```

Silvia Mazzoni, OpenSees Days 2010

---

Section Command

This command is used to construct a SectionForceDeformation object, hence referred to as Section, which represents force-deformation behavior of a geometrical object. The command is of the form:

```plaintext
section Command
```

What is a section?

- A section defines the stress resultant force-deformation response of 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 force-deformation.

The valid sections to any section when creating an ElementFibers are 'force' and 'deformation.'

Silvia Mazzoni, OpenSees Days 2010
Beam Section: Elastic

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)

Column Section: Fiber Section

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. quadratic, cubic, hexagonal, or rectangular) and layers (General Layer Command, Straight Layer Command) are used to define the discretization of the associated with the structural 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>
  
  concrete
  steel
  
  material stress-strain
}
Patch Command: Concrete

Quadrilateral Patch Command

This command is used to construct a Patch object with a quadrilateral shape. The geometry of the patch is specified in sequence — counter-clockwise.

patch quad $matTag $numSubdivI $numSubdivJ $yI $xI $yJ $xJ $yK $xK $yL $xL

$matTag: material integer tag of the previously-defined UniaxialMaterial object used to represent the stress-strain curve.
$numSubdivI: number of subdivisions (fibers) in the I direction.
$numSubdivJ: number of subdivisions (fibers) in the J direction.
$yI, $xI: y & z-coordinates of vertex I (local coordinates).
$yJ, $xJ: y & z-coordinates of vertex J (local coordinates).
$yK, $xK: y & z-coordinates of vertex K (local coordinates).
$yL, $xL: y & z-coordinates of vertex L (local coordinates).

Layer Command: Steel

Straight Layer Command

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

layer straight $matTag $numBars $areaBar $yStart $xStart $yEnd $xEnd

$matTag: material integer tag of the previously-defined UniaxialMaterial object used to represent the stress-strain curve for the area of the fiber.
$numBars: number of reinforcing bars along layer.
$areaBar: area of individual reinforcing bar.
$yStart, $xStart: y & z-coordinates of starting point of reinforcing layer (local coordinate system).
$yEnd, $xEnd: y & z-coordinates of ending point of reinforcing layer (local coordinate system).
Column & Beam Sections

```
# Define ELEMENTS & SECTIONS -----------------------------
# symmetric section
set ColSecTag 1, # assign a tag number to the col
set BeamSecTag 2, # assign a tag number to the b.
# define section geometry
set coverCol [expr 6.*[in]]; # Column cover to reinforcing
set numBarsCol 10; # number of longitudinal-reinforc
set barAreaCol [expr 2.25*in^2]; # area of longitudinal-reinforc
# RC section
set coverY [expr $HCol/2.0]; # The distance from the section z
set coverZ [expr $BCol/2.0]; # The distance from the section y
set coreY [expr $coverY-$coverCol]
set coreZ [expr $coverZ-$coverCol]
set nFy 16; # number of fibers for concrete in y-direct
set nFz 4; # number of fibers for concrete in z-direction
section fiber $SecColSecTag ; # Define the fiber section
patch quad $IDconU $nFZ $nFY -$coverY $coverZ -$coverY -$coverZ $coverY $coverZ $coverY $coverZ;
layer straight $IDreinf $numBarsCol $barAreaCol $coreY $coreZ $coreY $coreZ $coreY $coreZ. # top layer reinforcement
layer straight $IDreinf $numBarsCol $barAreaCol $coreY $coreZ $coreY $coreZ. # bottom layer reinforcement
); # end of fibersection definition
# BEAM section
section Elastic $BeamSecTag $Ec $ABeam $IzBeam, # elastic beam section
```

Transformations and Elements

```
# define geometric transformation performs a linear geometric transformation of
# beam stiffness and resisting force from the basic system to the global-coordinate system
set ColTransfTag 1; # associate a tag to column transformation
set BeamTransfTag 2; # associate a tag to beam transformation (good practice to keep col and beam separate)
set ColTransfType Linear ; # options, Linear PDelta Corotational
geomTransf $ColTransfType $ColTransfTag ; # only columns can have PDelta effects (gravity effects)
geomTransf Linear $BeamTransfTag ;
```

Silvia Mazzoni, OpenSees Days 2010
### Recorders

```plaintext
# Define RECORDERS

recorder Node -file $dataDir/DFree.out -time -node 3 -def 1 2 3 disp:  # displacements of free nodes
recorder Node -file $dataDir/DBase.out -time -node 1 2 -def 1 2 3 disp:  # displacements of support nodes
recorder Node -file $dataDir/RBase.out -time -node 1 2 -def 1 2 3 reaction:  # reaction forces
recorder Drift -file $dataDir/Drift.out -time -Node 12 -Node 3 4 -def 1 -perpDir 2:  # lateral drift
recorder Element -file $dataDir/EColl.out -time -ele 1 2 globalForce:  # element forces -- column
recorder Element -file $dataDir/DefoColl.out -time -ele 1 2 section 1 force:  # section forces
recorder Element -file $dataDir/DefoCollSec.out -time -ele 1 2 section 1 deformation:  # section deformations
```

### Gravity Load

```plaintext
# define GRAVITY

set WzBeam [expr $Weight/$LBeam]
pattern Plain 1 Linear:

elsLoad -ele 3 -type beamUniform -$WzBeam : # distributed superstructure weight on beam
```
Gravity Analysis

Gravity-analysis parameters -- load-controlled static analysis
set Tol 1e-8;  # convergence tolerance for tet
Constraints Plane;  # how it handles boundary conditions
Numberer Plain;  # numberer dof x to minimize band-width (optimization), if you want to
Memory BondGeneral;  # how to store and solve the system of equations in the analysis
set NormDispInc $Tol 6;  # determine if convergence has been achieved at the end of an iteration step
algorithm Newton;  # use Newton's solution algorithm: update tangent stiffness at every iteration
set NetepGravity 10;  # apply gravity in 10 steps
set DGravity [expr $NetepGravity];  # first load increment:
integrator LoadControl $DGravity;  # determine the next time step for an analysis
analysis Static;  # define type of analysis static or transient
analyze $NetepGravity;  # apply gravity
loadConst time 0.0
puts "Model built"

Run OpenSees

source Ex4.Portal2D.build.InelasticSection.tcl
Structural Example – Reinforced-Concrete Frame:

Lateral-Load Analysis

Example 4 in examples manual

Model Building

Analysis

Silvia Mazzoni, OpenSees Days 2010
Static Pushover: define load

```plaintext
# Example4. 2D Portal Frame -- Static Pushover Analysis
# Silvia Mazzoni & Frank McKenna, 200
# execute this file after you have built the model, and after
#
# we need to set up parameters that are particular to the n
set IdistrNode 3; # node where displacement is read
set IdistrDOF 1; # degree of freedom of displacement
# characteristics of pushover analysis
set Dmax [expr 0.1*$LCol]; # maximum displacement of
set Dincr [expr 0.001*$LCol]; # displacement increment
# create load pattern for lateral pushover load
set Hload [expr $Weight/2]; # define the lateral load
set PushNode "3 4"; # define nodes where lateral load
pattern Plain 200 Linear(); # define load pattern --
foreach PushNode $PushNode { # define load pattern --
    load $PushNode $Hload 0.0 0.0 0.0 0.0 0.0 0.0
}
#
# set up analysis parameters
source LibAnalysisStaticParameters.tcl; # constraint land
```

Silvia Mazzoni, OpenSees Days 2010

---

Static Pushover: analyze

```plaintext
# set up analysis parameters
source LibAnalysisStaticParameters.tcl;
#
# perform Static analysis
set Nsteps [expr int($Dmax/$Dincr)]; # number of
set ok [analyze $Nsteps]; # this will return
set hist1 "% Pushover analysis: CtrlNode %%.3; def % %.3; if ($sk <= 0) { # if analysis fails, we try some other stuff, perform:
    set Dynap 0.0
    set ok 0
    while ($Dstep <= 10 && $sk <= 0) {
        set controlDisp [getIdistrNode $IdistrDOF set Disp [expr $controlDisp/$Dmax]
        set ok [analyze 1]
    }
    # and while loop
    # if ok 10
```

Silvia Mazzoni, OpenSees Days 2010
Cyclic Pushover: define Load

```plaintext
# define load - load and steps
set DctrlNode 3; # node where displacement is read for displacements
set DctrlDOF 1; # degree of freedom of displacement read
set Dmax [exp expr 0.005 0.01 0.035 0.1]; # vector of displacement-coefficients
set Delta [exp expr 0.005 0.01 0.035 0.1]; # displacement increment for each load
set Fact $LCol; # scale drift ratio by story height for displacements
set cycleType Full; # you can do Full / Push / Half cycles
set Ncycles 1; # specify the number of cycles at each peak

# create load pattern for lateral pushover load
set $Head [exp expr $Weight/2]; # define the lateral load as a percentage
set $PushNode "3 4"; # define nodes where lateral load is applied
pattern Plain 200 Linear
for each PushNode $PushNode
    load $PushNode $Head 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
end
```

Silvia Mazzoni, OpenSees Days 2010

---

Cyclic Pushover: analyze

```plaintext
# perform Static Cyclic Displacement Analysis
source Lib GeneratePeaks.tcl
set fmt "%3i Cyclic analysis: CtrlNode %3i, dof %3i, Disp%3.4f %s"; # format for screen/file
for each DctrlNode $Dmax
    set $Dctrl $Dmax
    set $Delta [exp expr 0.005 0.01 0.035 0.1]; # vector of displacement-coefficients
    set $Fact $LCol; # scale drift ratio by story height for displacements
    set $cycleType Full; # you can do Full / Push / Half cycles
    set $Ncycles 1; # specify the number of cycles at each peak
    set $Head [exp expr $Weight/2]; # define the lateral load as a percentage
    set $PushNode "3 4"; # define nodes where lateral load is applied
    pattern Plain 200 Linear
    for each $PushNode
        load $PushNode $Head 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    end

# analyze command
set ok [analyze]
if {$ok == 0} {
    puts "Try another analysis..."
    test NormDispIncr $Tel 2000 0
    algorithm Newton -initial
    set ok [analyze]
    test $TelStatic $TelDynamic $maxNumIterStatic 0
end
```

Silvia Mazzoni, OpenSees Days 2010
Uniform Excitation 1D: define

```bash
# Uniform Earthquake ground motion (uniform acceleration input at all support nodes)
set 6MDir 1:   # ground-motion direction
set 6Mfile "H-e0140": # ground-motion filenames
set 6Mfact 1.5: # ground-motion scaling factor
# set up ground-motion-analysis parameters
set DtAnalysis [expr 0.01*$sec]: # time-step Dt for lateral analysis
set TmaxAnalysis [expr 10.*$sec]: # maximum duration of ground-motion
source LibAnalysisDynamicParameters.tcl: # constraintsHandler::DOFnumb
# ---------------------------------------- perform Dynamic Ground-Motion Anal
# the following commands are unique to the Uniform Earthquake excitation
set IDloadTag 400: # for uniformSupport excitation
# read a FEER strong motion database file, extracts $dt from the header and
# to the format OpenSees expects for Uniform/multiple-support ground motions
source ReadSMDBfile.tcl: # read in procedure MultiInit
# Uniform EXCITATION: acceleration input
set infile $MAdir/$MFile.at2
set outFile $MAdir/$MFile.g3: # set variable holding new filename (FEER files have
ReadSMDBfile $inFile $outFile $dt: # call procedure to convert the ground-motion fi
set GMfact [expr $g*$mKforce]: # data in input file is in g Units --> ACCELERAT
set AccelSeries "Series-dt $dt-filePath $outFiles-factor $GMfact": # time series
pattern UniformExcitation $IDloadTag $6MDir $6Mfact $6Mfile $6Mfile $6Mfile
```

Silvia Mazzoni, OpenSees Days 2010

---

Uniform Excitation 1D: damping

```bash
# define DAMPING
# apply Rayleigh DAMPING from $xDamp
# D = alphaM*M + betaKcurr*Kcurrent + $betaKcomm*KlastComm + $beatKinit*$Kinit
set xDamp 0.02: # 2% damping ratio
set lambda [expr 1]; # eigenvalue mode 1
set omega [expr pow($lambda,0.5)];
set alphaM: # M-prop. damping: D = alphaM*M
set betaKcurr 0.0: # K-proportional damping: +betaKcurr*Kcurrent
set betaKcomm [expr 2.*$xDamp/($omega)]: # K-prop. damping parameter: +betaKcomm
set betaKinit 0.0: # initial-stiffness proportional damping +beatKinit*Kini
rayleigh alphaM betaKcurr betaKinit betaKcomm: # RAYLEIGH damping
```

Silvia Mazzoni, OpenSees Days 2010
Uniform Excitation 1D: analyze

```plaintext
set Nsteps [expr int($TmaxAnalysis/$DTAnalysis)];
set ok [analyze $Nsteps $DTAnalysis]; # actually perform
if {$ok eq 0} {
    # analysis was not successful.
    # change some analysis parameters to achieve convergence
    # performance is slower inside this loop
    # Time-controlled analysis
    set ok 0;
    set controlTime [getTime];
    while {$controlTime < $TmaxAnalysis && $ok eq 0} {
        set controlTime [getTime];
        set ok [analyze 1 $DTAnalysis];
        if {$ok eq 0} {
            puts "Trying Newton with Initial Tangent ..."
            test NormDispInc $Tol 1000.0
            algorithm Newton -initial
            set ok [analyze 1 $DTAnalysis]
            test $testTypeDynamic $TolDynamic $maxNumIterDynamic
            algorithm $algorithmTypeDynamic
        }
        if {$ok eq 0} {
            puts "Trying Brayden ..."
            algorithm Brayden 8
            set ok [analyze 1 $DTAnalysis]
            algorithm $algorithmTypeDynamic
        }
    }
}
```

Silvia Mazzoni, OpenSees Days 2010

Uniform Excitation 2D: define

```plaintext
# Example: 2D Portal Frame: Dynamic EQ input analysis -- bilateral
# Silvia Mazzoni & Frank McKenna, 2006
# execute this file after you have built the model, and after you apply growth
#
# Bidirectional Uniform Earthquake ground motion (uniform acceleration input)
# gMfile "HEDGEO\Earthq.xml";
# gMfile "induced\Earthq.xml";
set gMfile "induced\Earthq.xml";
set gMfile "HEDGEO\Earthq.xml";
# ground-motion filenames should be:
set gMdirection "1 2"; # ground-motion direction
set gMfact "1.5 0.25"; # ground-motion scaling factor
# set up ground-motion analysis parameters
set DTAnalysis [expr 0.005*$sec]; # Time-step Dt for lateral analysis
set TmaxAnalysis [expr $DTAnalysis*$sec]; # maximum duration of ground-motion
```

Silvia Mazzoni, OpenSees Days 2010
Uniform Excitation 2D: analyze

```plaintext
# --------- set up analysis parameters
# constraintHandler DofNum = 1
set Nsteps (expr int($Tmax/Analysis+$Dt/Analysis));
set ok [analyze 0 $Nsteps $Dt/Analysis];  # actually perform analysis
if (!$ok) {
    return;
}  # analyze was not successful.
#
# change some analysis parameters to achieve convergence.
# performance is slower inside this loop.
#
# Time-controlled analysis
set nk 0;
set controlTime (getTime);
while ($controlTime < $Tmax && $nk != 0) {
set controlTime (getTime);
set ok [analyze 0 $Dt/Analysis];
if (!$ok) {
puts "Trying Newton with Initial Tangent ...
algorithm Newton initial
set ok [analyze 0 $Dt/Analysis];
} else {
puts "Trying Broyden ..."
algorithm Broyden 5
set ok [analyze 0 $Dt/Analysis];
}  # Newton converges.
}```

Questions, or statements:

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

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

Silvia Mazzoni, OpenSees Days 2010