Structural Modeling With Examples

Silvia Mazzoni, PhD

Structural Consultant
Degenkolb Engineers

OpenSees Days 2010

OpenSees

Degenkolb

Tcl & OpenSees commands

- Comand 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

page | discussion | view source | history

Command_Manual

navigation
- OpenSees Home
- Message Board
- OpenSees Wiki
- User Documetation
- Download Page
- Source Code

search

Go | Search

toolbox
- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

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

Code Developed by: fmk

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 analyst the option of associating nodal mass with the node

nodes and boundary conditions

copy and paste from manual:

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

Nodal coordinates and BC

coordinates & mass

```
# Define nodes;                 # frame is in X-Y plane
node  1   0.0      0.0          0.0
node  2   100.    0.0           0.0
node  3   0.0      $Lcol        0.0         -mass 4.7 0.0 0.0 0.0 0.0 0.0
node  4   $Lbeam  $Lcol         0.0         -mass $Mnode 0.0 0.0 0.0 0.0 0.0
# Boundary conditions;          # node DX DY DZ RX RY RZ ! 1: fixed, 0: released
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
#                                 3----------------------4
#                                  |                      |
#                                  |                      |
#                               --1--                  --2--
```

boundary conditions

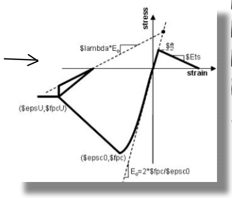
ModelBuilder Objects

- 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

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

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


materials

concrete

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

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

if {$ConcreteMaterialType == "elastic"} {
    uniaxialMaterial Elastic $IDcore $Ec
    uniaxialMaterial Elastic $IDcover $Ec
} else {;         # $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
}
```

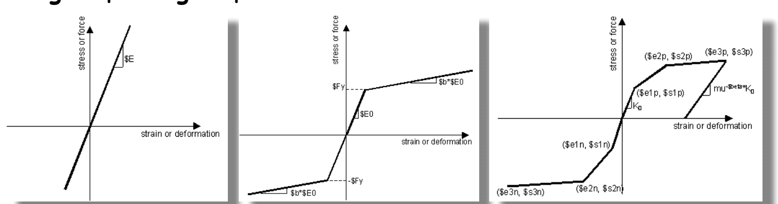
uniaxialMaterial Hysteretic $matTag $s1p $e1p $s2p $e2p <$s3p $e3p> $s1n $e1n $s2n $e2n <$s3n $e3n> $pinchX $pinchY $damage1 $damage2 <$beta>

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

materials

reinforcing steel

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



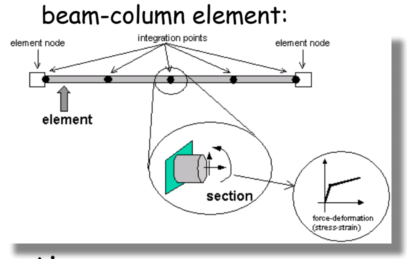
ModelBuilder Objects

- 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

beam-column element:



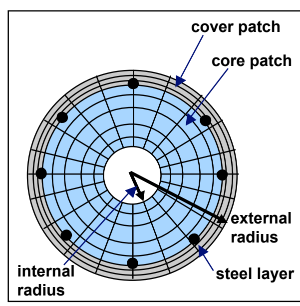
- Elastic Membrane Plate Section
- Plate Fiber Section
- Bidirectional Section
- Isolator2spring Section: Model to include buckling behavior of an elastomeric bearing

Uniaxial Beam-Column Sections:

- Elastic Section
  - linear-elastic moment-curvature relationship
- Uniaxial Section
  - user-defined moment-curvature relationship (use unaxialMaterial)
  - 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)

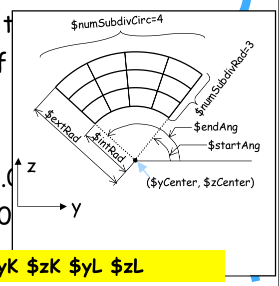
$A area of fiber

$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 t
$yCenter $zCenter	y & z-coordinates of the center of
$intRad	internal radius
$extRad	external radius
$startAng	starting angle (optional. default=0.
$endAng	ending angle (optional. default=360

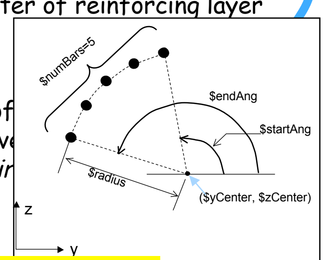


patch quad $matTag $numSubdivIJ $numSubdivJK $yI $zI $yJ $zJ $yK $zK $yL $zL

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 $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, respective *(Optional, Default: a full cir is assumed 0-360)*



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

tcl procedure

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

```
proc procName {input variables} {
   … series of commands
}
```

to execute:

```
procName (input variables)
```

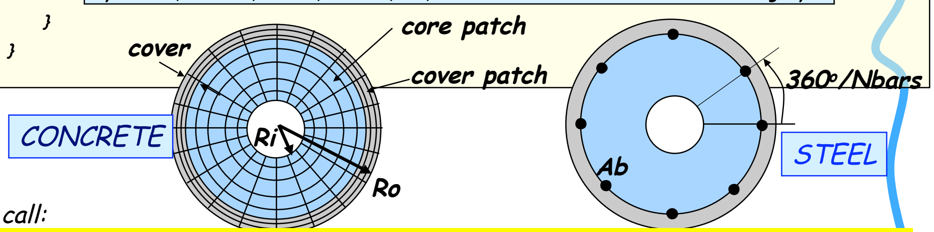
```
proc multiply {a b} {
   set c [expr $a*$b]
   return $c
}
set a 3; set b 5
set result [multiply $a $b]
```


tcl proc: define fiber section

```
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
    }
}
```

CONCRETE

STEEL

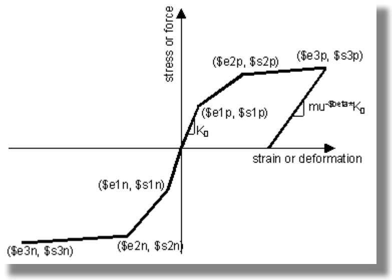


call:

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

UniaxialSection

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

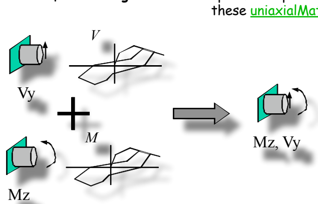


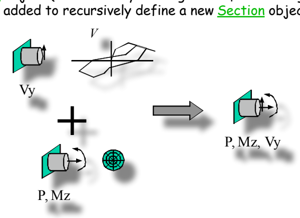
section aggregator

- 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>		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





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

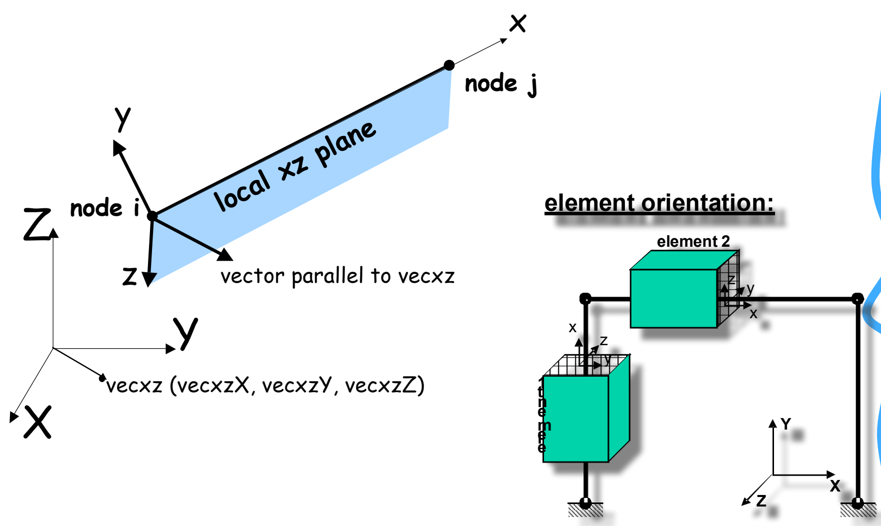
geometric transformation

- 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
$vecxzX **$vecxzY** **$vecxzZ**	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 $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



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
```

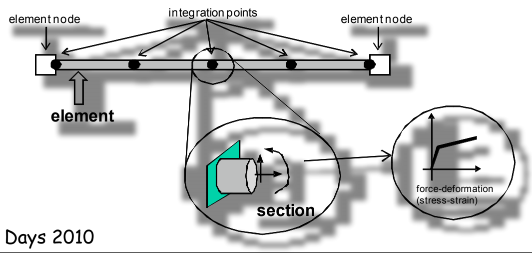
- 3D:

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

Nonlinear Beam Column Element

element 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";                                              Tcl procedure
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
```

COLUMN

BEAM

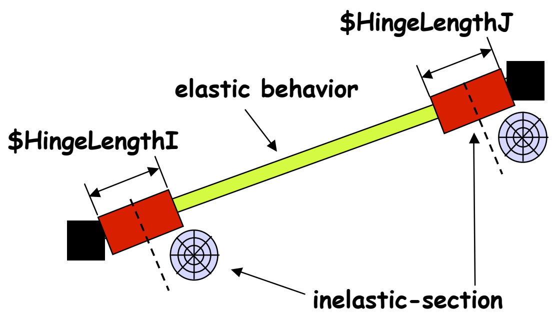
beamWithHinges Element

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>
```

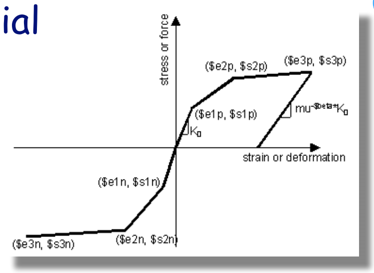


beamWithHinges Element

$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})*

ZeroLength Element

- DOF spring
- Use uniaxialMaterial to define "force"-"deformation" response.
  - Translational dof: force-deformation
  - Rotational dof: moment-rotation
  - e.g. Hysteretic 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

```
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 {
    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:

```
eleLoad -ele $eleTag1 <$eleTag2 ....> -type -beamUniform $Wz <$Wx>
```

•Point load:

```
eleLoad -ele $eleTag1 <$eleTag2 ....> -type -beamPoint $Pz $xL <$Px>
```

3D:

•Uniformly-distributed load:

```
eleLoad -ele $eleTag1 <$eleTag2 ....> -type -beamUniform $Wy $Wz <$Wx>
```

•Point load:

```
eleLoad -ele $eleTag1 <$eleTag2 ....> -type -beamPoint $Py $Pz $xL <$Px>
```

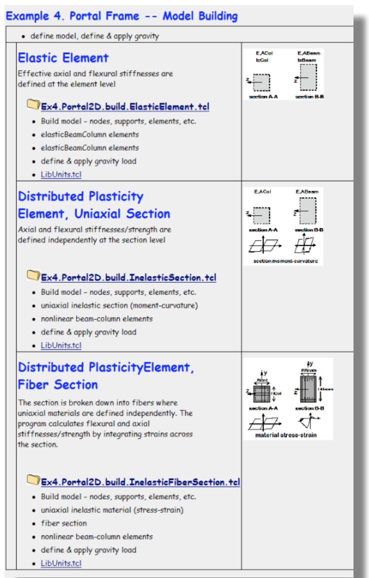
```
pattern Plain 1 Linear {
    sp    1 2 -0.001
    eleLoad -ele 3 -type -beamUniform [expr -$Weight/$LBeam]
}
```


Structural Example – Reinforced-Concrete Frame:

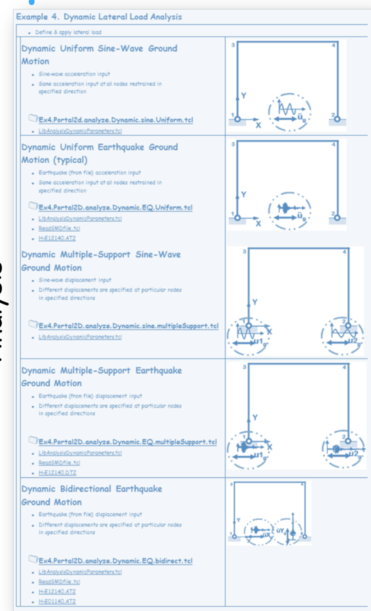
Building the Model

Example 4 in examples manual

Model Building

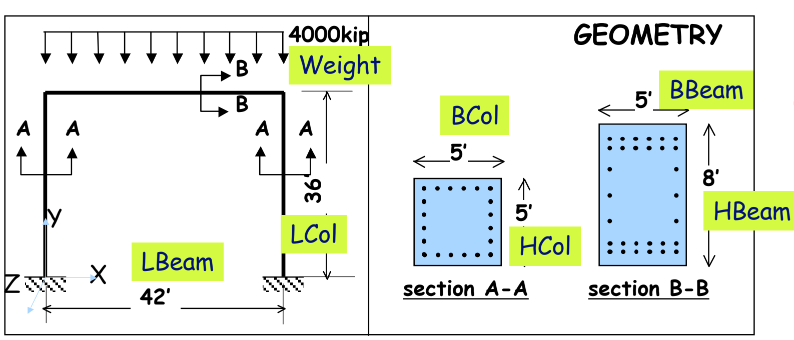


Analysis



problem statement

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



.....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
#
#    ^Y
#    |
#    3_________(3)________4      __
#    |                    |      |
#    |                    |      |
#    |                    |      |
#   (1)                  (2)    LCol
#    |                    |      |
#    |                    |      |
#    |                    |      |
#   =1=                  =2=    _|_  -------->X
#    |----------LBeam------------|
#

# SET UP ----------------------------------------------------------------------------------
wipe;                              # clear memory of all past model definitions
model BasicBuilder -ndm 2 -ndf 3;  # Define the model builder, ndm=#dimension, ndf=#dofs
set dataDir Data;                  # set up name of data directory
file mkdir $dataDir;               # create data directory
set GMdir "GMfiles";               # ground-motion file directory
source LibUnits.tcl;               # define basic and system units
```

Put unit definitions into a file

LibUnits.tcl

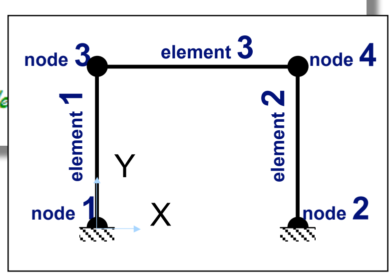
```
# define UNITS ----------------------------------------------------------------
set in 1.;                                # define basic units -- output units
set kip 1.;                               # define basic units -- output units
set sec 1.;                               # define basic units -- output units
set LunitTXT "inch";                      # define basic-unit text for output
set FunitTXT "kip";                       # define basic-unit text for output
set TunitTXT "sec";                       # define basic-unit text for output
set ft [expr 12.*$in];                    # define engineering units
set ksi [expr $kip/pow($in,2)];
set psi [expr $ksi/1000.];
set lbf [expr $psi*$in*$in];              # pounds force
set pcf [expr $lbf/pow($ft,3)];           # pounds per cubic foot
set psf [expr $lbf/pow($ft,3)];           # pounds per square foot
set in2 [expr $in*$in];                   # inch^2
set in4 [expr $in*$in*$in*$in];           # inch^4
set cm [expr $in/2.54];                   # centimeter
set PI [expr 2*asin(1.0)];                # define constants
set g [expr 32.2*$ft/pow($sec,2)];        # gravitational acceleration
set Ubig 1.e10;                           # a really large number
set Usmall [expr 1/$Ubig];                # a really small number
```

Geometry, Weight, Mass

```
# define GEOMETRY -------------------------------------------------------------
set LCol [expr 36*$ft];          # column length
set LBeam [expr 42*$ft];         # beam length
# define section geometry
set HCol [expr 5.*$ft];          # Column Depth
set BCol [expr 5.*$ft];          # Column Width
set HBeam [expr 8.*$ft];         # Beam Depth
set BBeam [expr 5.*$ft];         # Beam Width
# superstructure weight
set Weight [expr 2000.*$kip];

# calculated parameters
set PCol [expr $Weight/2];                                   # nodal dead-load weigh
set Mass [expr $PCol/$g];                                    # nodal mass
set MCol [expr 1./12.*($Weight/$LBeam)*pow($LBeam,2)];       # beam-end moment due
# calculated geometry parameters
set ACol [expr $BCol*$HCol];                                 # cross-sectional area
set ABeam [expr $BBeam*$HBeam];
set IzCol [expr 1./12.*$BCol*pow($HCol,3)];                  # Column moment of iner
set IzBeam [expr 1./12.*$BBeam*pow($HBeam,3)];               # Beam moment of inert
```


Nodes: Coords, BC, Mass

```
# nodal coordinates:
node 1 0 0;            # node#, X, Y
node 2 $LBeam 0
node 3 0 $LCol
node 4 $LBeam $LCol
# Single point constraints -- Boundary Conditions
fix 1 1 1 0;           # node DX DY RZ
fix 2 1 1 0;           # node DX DY RZ
fix 3 0 0 0
fix 4 0 0 0
# nodal masses:
mass 3 $Mass  0. 0.;       # node
mass 4 $Mass  0.  0.
```



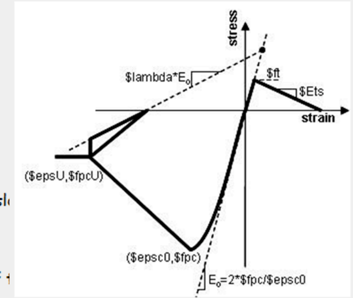
Concrete Material: Concrete02

Concrete02 Material -- Linear Tension Softening

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

```
uniaxialMaterial Concrete02 $matTag $fpc $epsc0 $fpcu $epsU $lambda $ft $Ets
```

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



Concrete02 Material -- Material parameters

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

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

Steel Material: Steel02

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

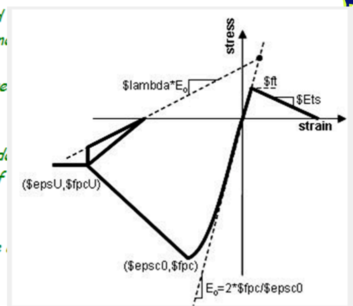
This command is used to construct a uniaxial Giuffre-Menegotto-Pinto steel material object with isotropic strain hardening.

```
uniaxialMaterial Steel02 $matTag $Fy $E $b $R0 $cR1 $cR2 <$a1 $a2 $a3 $a4 $sigInit>
```

$matTag	unique material object integer tag
$Fy	yield strength
$E	initial elastic tangent
$b	strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
$R0, $cR1, $cR2	control the transition from elastic to plastic branches. Recommended values: $R0=between 10 and 20, $cR1=0.925, $cR2=0.15
$a1, $a2, $a3, $a4	isotropic hardening parameters: (optional, default: no isotropic hardening)

Materials

```
# MATERIAL parameters -------------------------------------------------------------------
set IDconcU 1;                     # material ID tag -- unconfined
set IDreinf 2;                     # material ID tag -- reinforcem
# nominal concrete compressive strength
set fc [expr -4.0*$ksi];           # CONCRETE Compressive Stre
set Ec [expr 57*$ksi*sqrt(-$fc/$psi)];  # Concrete Elastic Modulus
# unconfined concrete
set fc1U $fc;                      # UNCONFINED concrete (tod
set eps1U -0.003;                  # strain at maximum strength of
set fc2U [expr 0.2*$fc1U];         # ultimate stress
set eps2U -0.05;                   # strain at ultimate stress
set lambda 0.1;                    # ratio between unloading slope
# tensile-strength properties
set ftU [expr -0.14*$fc1U];        # tensile strength +tension
set Ets [expr $ftU/0.002];         # tension softening stiffness
# -----------
set Fy [expr 66.8*$ksi];           # STEEL yield stress
set Es [expr 29000.*$ksi];         # modulus of steel
set Bs 0.01;                       # strain-hardening ratio
set R0 18;                         # control the transition from elastic to plastic branches
set cR1 0.925;                     # control the transition from elastic to plastic branches
set cR2 0.15;                      # control the transition from elastic to plastic branches

uniaxialMaterial Concrete02 $IDconcU $fc1U $eps1U $fc2U $eps2U $lambda $ftU $Ets;  # build cover concrete
uniaxialMaterial Steel02 $IDreinf $Fy $Es $Bs $R0 $cR1 $cR2;        # build reinforcement material
```



Concrete02 Material -- Material parameters

Section Command

section Command

This command is used to construct a SectionForceDeformation object, hereto referred to as Section, which represents force-de

What is a section?

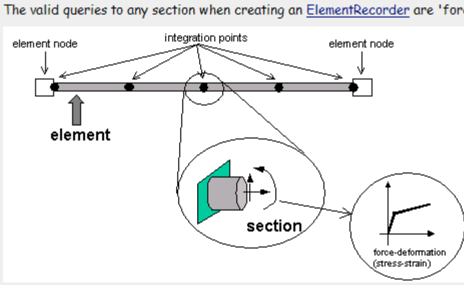
- A section defines the stress resultant force-deformation response at a cross section of a beam-column or plate element
- Types of sections:

• **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 b

The valid queries to any section when creating an ElementRecorder are 'force' and 'deformation.'



- 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

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

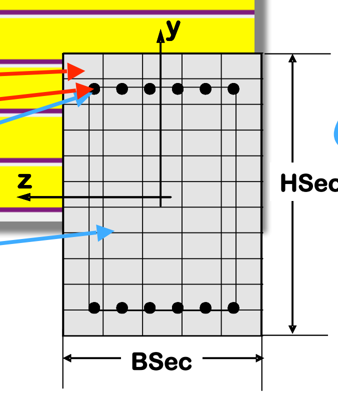
Fiber Section

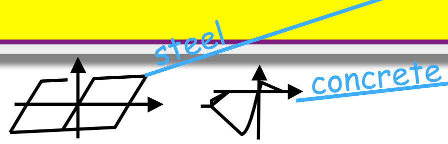
The FiberSection object is composed of Fiber objects.

A fiber section has a general geometric configuration formed by subregions of simpler, regular shapes (e.g. qua patch and layer (Circular Layer Command, Straight Layer Command) are used to define the discretization of the associated with uniaxialMaterial 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>
}
```





material stress-strain

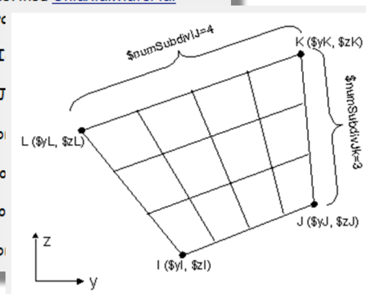
Patch Command: Concrete

Quadrilateral Patch Command

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

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

$matTag		material integer tag of the previously-defined UniaxialMaterial object used to represent the stress-stra
$numSubdivIJ		number of subdivisions (fibers) in the I
$numSubdivJK		number of subdivisions (fibers) in the J
$yI	**$zI**	y & z-coordinates of vertex I (local coo
$yJ	**$zJ**	y & z-coordinates of vertex J (local coo
$yK	**$zK**	y & z-coordinates of vertex K (local coo
$yL	**$zL**	y & z-coordinates of vertex L (local coo



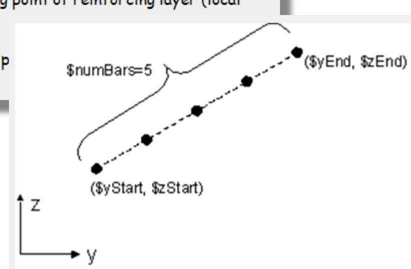
Layer Command: Steel

Straight Layer Command

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

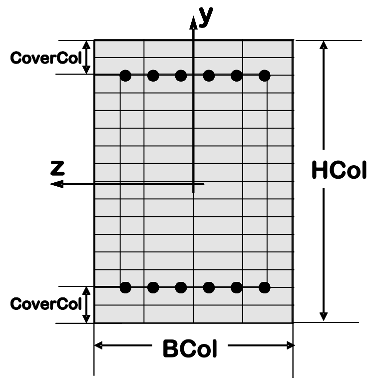
layer straight $matTag $numBars $areaBar $yStart $zStart $yEnd $zEnd

$matTag		material integer tag of the previously-defined UniaxialMaterial 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 p coordinate system)



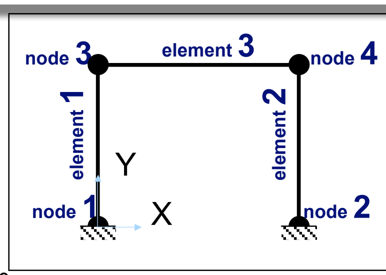
Column & Beam Sections

```
# Define ELEMENTS & SECTIONS -----------------------------
# symmetric section
set ColSecTag 1;                 # assign a tag number to the co
set BeamSecTag 2;                # assign a tag number to the be
# define section geometry
set coverCol [expr 6.*$in];      # Column cover to reinforcing
set numBarsCol 10;               # number of longitudinal-reinf
set barAreaCol [expr 2.25*$in2]; # 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 fiberSec $ColSecTag  {;     # Define the fiber section
   patch quadr $IDconcU $nfZ $nfY -$coverY $coverZ -$coverY -$coverZ $coverY -$coverZ $coverY $coverZ;
   layer straight $IDreinf $numBarsCol $barAreaCol -$coreY $coreZ -$coreY -$coreZ;   # top layer reinfocement
   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  ;

# element connectivity:
set numIntgrPts 5;                    # number of integration points for force-based element
element nonlinearBeamColumn 1 1 3 $numIntgrPts $ColSecTag $ColTransfTag;  # self-explanatory when using variables
element nonlinearBeamColumn 2 2 4 $numIntgrPts $ColSecTag $ColTransfTag;
element nonlinearBeamColumn 3 3 4 $numIntgrPts $BeamSecTag $BeamTransfTag;
```

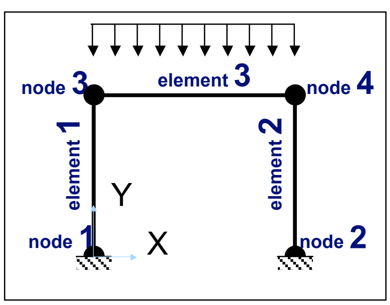


Recorders

```
# Define RECORDERS -------------------------------------------------------------
recorder Node -file $dataDir/DFree.out -time -node 3 4 -dof 1 2 3 disp;          # displacements of free node
recorder Node -file $dataDir/DBase.out -time -node 1 2 -dof 1 2 3 disp;        # displacements of support noo
recorder Node -file $dataDir/RBase.out -time -node 1 2 -dof 1 2 3 reaction;       # support reaction
recorder Drift -file $dataDir/Drift.out -time -iNode 1 2 -jNode 3 4 -dof 1 -perpDirn 2 ;   # lateral drift
recorder Element -file $dataDir/FCol.out -time -ele 1 2 globalForce;          # element forces -- column
recorder Element -file $dataDir/FBeam.out -time -ele 3 globalForce;           # element forces -- beam
recorder Element -file $dataDir/ForceColSec1.out -time -ele 1 2 section 1 force;     # Column section forces, d
recorder Element -file $dataDir/DefoColSec1.out -time -ele 1 2 section 1 deformation;    # section deforma
recorder Element -file $dataDir/ForceColSec$numIntgrPts.out -time -ele 1 2 section $numIntgrPts force;    # se
recorder Element -file $dataDir/DefoColSec$numIntgrPts.out -time -ele 1 2 section $numIntgrPts deformation;
recorder Element -file $dataDir/ForceBeamSec1.out -time -ele 3 section 1 force;      # Beam section forces,
recorder Element -file $dataDir/DefoBeamSec1.out -time -ele 3 section 1 deformation;     # section deforma
recorder Element -file $dataDir/ForceBeamSec$numIntgrPts.out -time -ele 3 section $numIntgrPts force;    # se
recorder Element -file $dataDir/DefoBeamSec$numIntgrPts.out -time -ele 3 section $numIntgrPts deformation;  #
```


Gravity Load

```
# define GRAVITY -------------------------------------------------------------
set WzBeam [expr $Weight/$LBeam];
pattern Plain 1 Linear {
   eleLoad -ele 3 -type -beamUniform -$WzBeam ; # distributed superstructure-weight on beam
}
```

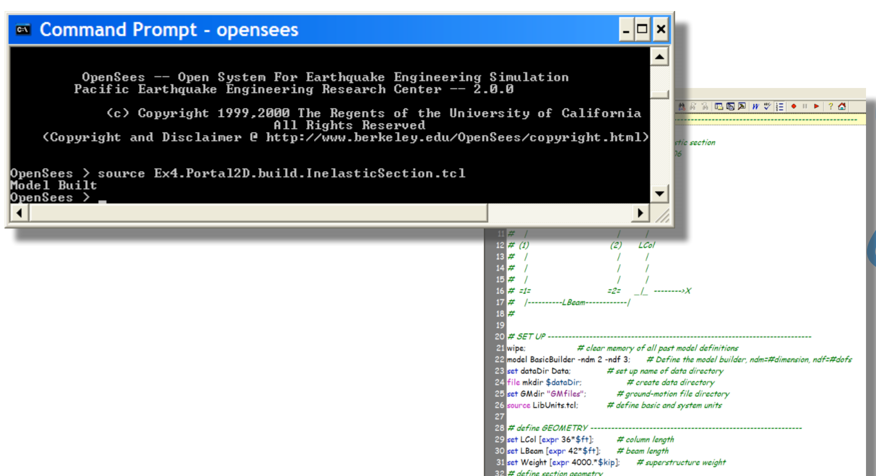


Gravity Analysis

```
# Gravity-analysis parameters -- load-controlled static analysis
set Tol 1.0e-8;            # convergence tolerance for test
constraints Plain;             # how it handles boundary conditions
numberer Plain;          # renumber dof's to minimize band-width (optimization), if you want to
system BandGeneral;      # how to store and solve the system of equations in the analysis
test NormDispIncr $Tol 6 ;       # determine if convergence has been achieved at the end of an iteration step
algorithm Newton;          # use Newton's solution algorithm: updates tangent stiffness at every iteration
set NstepGravity 10;       # apply gravity in 10 steps
set DGravity [expr 1./$NstepGravity];  # first load increment;
integrator LoadControl $DGravity;    # determine the next time step for an analysis
analysis Static;         # define type of analysis static or transient
analyze $NstepGravity;       # apply gravity
# ------------------------------------------------ maintain constant gravity loads and reset time to zero
loadConst -time 0.0

puts "Model Built"
```


Run OpenSees

source Ex4.Portal2D.build.InelasticSection.tcl

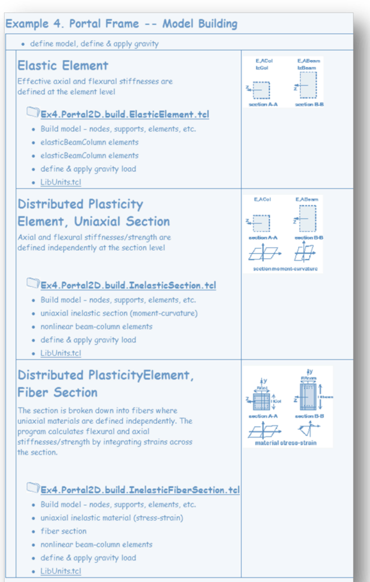


Structural Example – Reinforced-Concrete Frame:

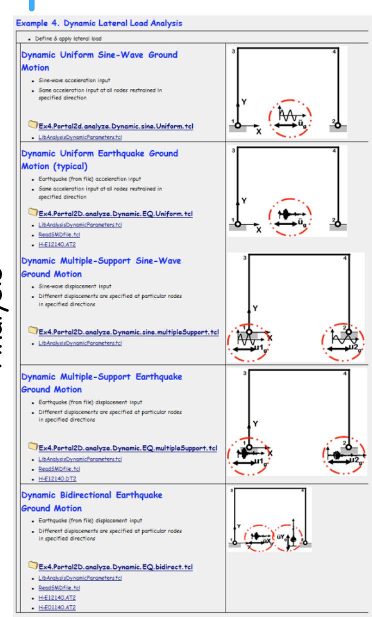
Lateral-Load Analysis

Example 4 in examples manual

Model Building



Analysis



LibAnalysisStaticParameters.tcl

```
# ---------------------------------------------------------------
# static analysis parameters
# I am setting all these variables as global variables (using varia
#    so that these variables can be uploaded by a procedure
#                                   Silvia Mazzoni & Frank McKenna, 2006
#
#

# CONSTRAINTS handler -- Determines how the constraint eq
#       Plain Constraints -- Removes constrained degrees of fre
#       Lagrange Multipliers -- Uses the method of Lagrange mu
#       Penalty Method -- Uses penalty numbers to enforce cons
#       Transformation Method -- Performs a condensation of c
variable constraintsTypeStatic Plain;     # default;
if {  [info exists RigidDiaphragm] == 1} {
   if {$RigidDiaphragm=="ON"} {
      variable constraintsTypeStatic Lagrange;   #     for large
   };  # if rigid diaphragm is on
};  # if rigid diaphragm exists
constraints $constraintsTypeStatic

# DOF NUMBERER (number the degrees of freedom in the dom
#   determines the mapping between equation numbers and degre
#       Plain -- Uses the numbering provided by the user
#       RCM -- Renumbers the DOF to minimize the matrix ban
set numbererTypeStatic RCM
numberer $numbererTypeStatic

# SYSTEM (http://opensees.berkeley.edu/OpenSees/manuals/u
#   Linear Equation Solvers (how to store and solve the system o
#   -- provide the solution of the linear system of equations Ku = P. Each solver is tailored
#       ProfileSPD -- Direct profile solver for symmetric positive definite matrices
#       BandGeneral -- Direct solver for banded unsymmetric matrices
#       BandSPD -- Direct solver for banded symmetric positive definite matrices
#       SparseGeneral -- Direct solver for unsymmetric sparse matrices
#       SparseSPD -- Direct solver for symmetric sparse matrices
#       UmfPack -- Direct UmfPack solver for unsymmetric matrices
set systemTypeStatic BandGeneral;      # try UmfPack for large model
system $systemTypeStatic
```

```
# TEST: # convergence test to
# Convergence TEST (http://opensees.berkeley.edu/OpenSees/manuals/usermanua
#   -- Accept the current state of the domain as being on the converged solution pat
#   -- determine if convergence has been achieved at the end of an iteration step
#       NormUnbalance -- Specifies a tolerance on the norm of the unbalanced load
#       NormDispIncr -- Specifies a tolerance on the norm of the displacement inc
#       EnergyIncr-- Specifies a tolerance on the inner product of the unbalanced
#       RelativeNormUnbalance --
#       RelativeNormDispIncr --
#       RelativeEnergyIncr --
variable TolStatic 1.e-8;                  # Convergence Test: tolerance
variable maxNumIterStatic 6;               # Convergence Test: maximum number of i
variable printFlagStatic 0;             # Convergence Test: flag used to print inform
variable testTypeStatic EnergyIncr ;    # Convergence-test type
test $testTypeStatic $TolStatic $maxNumIterStatic $printFlagStatic;
# for improved-convergence procedure:
   variable maxNumIterConvergeStatic 2000;
   variable printFlagConvergeStatic 0;

# Solution ALGORITHM: -- Iterate fr
#       Linear -- Uses the solution at t
#       Newton -- Uses the tangent at t
#       ModifiedNewton -- Uses the ta
#       NewtonLineSearch --
#       KrylovNewton --
#       BFGS --
#       Broyden --
variable algorithmTypeStatic Newton
algorithm $algorithmTypeStatic;
```

```
# Static INTEGRATOR: -- determine the next time step for an analysi
#       LoadControl -- Specifies the incremental load factor to be appl
#       DisplacementControl -- Specifies the incremental displacement
#       Minimum Unbalanced Displacement Norm -- Specifies the incre
#       Arc Length -- Specifies the incremental arc-length of the load-
# Transient INTEGRATOR: -- determine the next time step for an anal
#       Newmark -- The two parameter time-stepping method develope
#       HHT -- The three parameter Hilbert-Hughes-Taylor time-stepp
#       Central Difference -- Approximates velocity and acceleration b
integrator DisplacementControl  $IDctrlNode   $IDctrlDOF $Dincr

# ANALYSIS  -- defines what type of analysis is to be performed (htt
#       Static Analysis -- solves the KU=R problem, without the mass o
#       Transient Analysis -- solves the time-dependent analysis. The ti
#       variableTransient Analysis -- performs the same analysis type
#           there are convergence problems with the Transient Analys
set analysisTypeStatic Static
analysis $analysisTypeStatic
```


LibAnalysisDynamicParameters.tcl

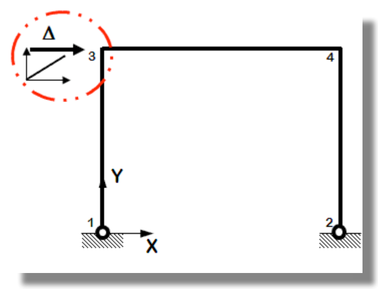
```
# ---------------------------------------------------------------
# dynamic-analysis parameters
# I am setting all these variables as global variable
#    so that these variables can be uploaded by a pr
#                                   Silvia Mazzoni & Frank Mc

# Set up Analysis Parameters ----------------------
# CONSTRAINTS handler -- Determines how the c
#       Plain Constraints -- Removes constrained de
#       Lagrange Multipliers -- Uses the method of
#       Penalty Method -- Uses penalty numbers to
#       Transformation Method -- Performs a cond
variable constraintsTypeDynamic Transformation;
constraints $constraintsTypeDynamic ;

# DOF NUMBERER (number the degrees of freedo
#   determines the mapping between equation numbe
#       Plain -- Uses the numbering provided by the
#       RCM -- Renumbers the DOF to minimize th
variable numbererTypeDynamic RCM
numberer $numbererTypeDynamic

# SYSTEM (http://opensees.berkeley.edu/OpenSe
#   Linear Equation Solvers (how to store and solve
#   -- provide the solution of the linear system of e
#       ProfileSPD -- Direct profile solver for sym
#       BandGeneral -- Direct solver for banded un
#       BandSPD -- Direct solver for banded symm
#       SparseGeneral -- Direct solver for unsymm
#       SparseSPD -- Direct solver for symmetric
#       UmfPack -- Direct UmfPack solver for unsymmetric matrices
variable systemTypeDynamic BandGeneral;  # try UmfPack for large problems
system $systemTypeDynamic
```

```
# TEST: # convergence test to
# Convergence TEST (http://opensees.berkeley.edu/OpenSees/manuals/usermanu
#   -- Accept the current state of the domain as being on the converged solution p
#   -- determine if convergence has been achieved at the end of an iteration step
#       NormUnbalance -- Specif
#       NormDispIncr -- Specifi
#       EnergyIncr-- Specifies a
#       RelativeNormUnbalance -
#       RelativeNormDispIncr --
#       RelativeEnergyIncr --
variable TolDynamic 1.e-8;
variable maxNumIterDynamic 10;
variable printFlagDynamic 0;
variable testTypeDynamic Energy
test $testTypeDynamic $TolDyna
# for improved-convergence proc
   variable maxNumIterConvergeD
   variable printFlagConvergeDyn

# Solution ALGORITHM: -- Itera
#       Linear -- Uses the solutio
#       Newton -- Uses the tange
#       ModifiedNewton -- Uses
#       NewtonLineSearch --
#       KrylovNewton --
#       BFGS --
#       Broyden --
variable algorithmTypeDynamic ModifiedNewton
algorithm $algorithmTypeDynamic;
```

```
# Static INTEGRATOR: -- determine the next time step for an analysis  (htt
#       LoadControl -- Specifies the incremental load factor to be applied to
#       DisplacementControl -- Specifies the incremental displacement at a s
#       Minimum Unbalanced Displacement Norm -- Specifies the incremental
#       Arc Length -- Specifies the incremental arc-length of the load-displa
# Transient INTEGRATOR: -- determine the next time step for an analysis i
#       Newmark -- The two parameter time-stepping method developed by N
#       HHT -- The three parameter Hilbert-Hughes-Taylor time-stepping m
#       Central Difference -- Approximates velocity and acceleration by cen
variable NewmarkGamma 0.5;   # Newmark-integrator gamma parameter (als
variable NewmarkBeta 0.25;  # Newmark-integrator beta parameter
variable integratorTypeDynamic Newmark;
integrator $integratorTypeDynamic $NewmarkGamma $NewmarkBeta

# ANALYSIS  -- defines what type of analysis is to be performed (http://op
#       Static Analysis -- solves the KU=R problem, without the mass or damp
#       Transient Analysis -- solves the time-dependent analysis. The time ste
#       variableTransient Analysis -- performs the same analysis type as the
#           there are convergence problems with the Transient Analysis obje
variable analysisTypeDynamic Transient
analysis $analysisTypeDynamic
```

Static Pushover: define load

```
# ------------------------------------------------------------
# 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 m
set IDctrlNode 3;          # node where displacement is read
set IDctrlDOF 1;           # degree of freedom of displaceme
# characteristics of pushover analysis
set Dmax [expr 0.1*$LCol];        # maximum displacement of
set Dincr [expr 0.001*$LCol];       # displacement incremen

# create load pattern for lateral pushover load
set Hload [expr $Weight/2];        # define the lateral loac
set iPushNode "3 4";         # define nodes where lateral loa
pattern Plain 200 Linear {;        # define load pattern --
   foreach PushNode $iPushNode {
      load $PushNode $Hload 0.0 0.0 0.0 0.0 0.0
   }
}

# ---------- set up analysis parameters
source LibAnalysisStaticParameters.tcl; # constraintsHand
```



Static Pushover: analyze

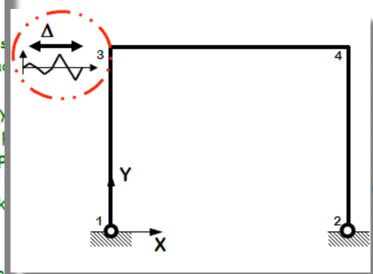
```
# ---------- set up analysis parameters
source LibAnalysisStaticParameters.tcl;
```

```
# --------------------------------     perform Static
set Nsteps [expr int($Dmax/$Dincr)];      # number
set ok [analyze $Nsteps];            # this will retur
set fmt1 "%s Pushover analysis: CtrlNode %.3i, dof %
if {$ok != 0} {
   # if analysis fails, we try some other stuff, perfor
   set Dstep 0.0;
   set ok 0
   while {$Dstep <= 1.0 && $ok == 0} {
      set controlDisp [nodeDisp $IDctrlNode $IDctr
      set Dstep [expr $controlDisp/$Dmax]
      set ok [analyze 1 ]
```

```
   # if analysis fails, we try some other stuff
   # performance is slower inside this loop global maxNumIterStatic;    #
   if {$ok != 0} {
      puts "Trying Newton with Initial Tangent .."
      test NormDispIncr   $Tol 2000 0
      algorithm Newton -initial
      set ok [analyze 1]
      test $testTypeStatic $TolStatic      $maxNumIterStatic    0
      algorithm $algorithmTypeStatic
   }
   if {$ok != 0} {
      puts "Trying Broyden .."
      algorithm Broyden 8
      set ok [analyze 1 ]
      algorithm $algorithmTypeStatic
   }
   if {$ok != 0} {
      puts "Trying NewtonWithLineSearch .."
      algorithm NewtonLineSearch 0.8
      set ok [analyze 1]
      algorithm $algorithmTypeStatic
   }

 };  # end while loop
};     # end if ok !0
```

Cyclic Pushover: define Load

```
# ------------------------------------------------------------------------------------------------
# Example4. 2D Portal Frame--  Static Reversed-Cyclic Analysis
#                          Silvia Mazzoni & Frank McKenna, 2006
# execute this file after you have built the model, and after you appl
#

# we need to set up parameters that are particular to the model.
set IDctrlNode 3;            # node where displacement is read for dis
set IDctrlDOF 1;             # degree of freedom of displacement read
# characteristics of cyclic analysis
set iDmax "0.005  0.01 0.035 0.1";        # vector of displacement-cy
set Dincr [expr 0.001*$LCol];           # displacement increment for p
set Fact $LCol;              # scale drift ratio by storey height for disp
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 Hload [expr $Weight/2];        # define the lateral load as a pro
set iPushNode "3 4";          # define nodes where lateral load is appli
pattern Plain 200 Linear {;         # define load pattern -- generali
   foreach PushNode $iPushNode {
      load $PushNode $Hload 0.0 0.0 0.0 0.0 0.0
   }
}
```

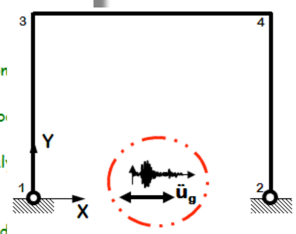


Cyclic Pushover: analyze

```
# --------------------------------      perform Static Cyclic Displacements Analysis
source LibGeneratePeaks.tcl
set fmt1 "%s Cyclic analysis: CtrlNode %.3i, dof %.1i, Disp=%.4f %s";  # format for screen/file
foreach Dmax $iDmax {
   set iDstep [GeneratePeaks $Dmax $Dincr $CycleType $Fact];    # this proc is defined above
   for {set i 1} {$i <= $Ncycles} {incr i 1} {
      set zeroD 0
      set D0 0.0
      foreach Dstep $iDstep {
         set D1 $Dstep
         set Dincr [expr $D1 - $D0]
         integrator DisplacementControl  $IDctrlNode $IDctrlDOF $Dincr
         analysis Static
         # ------------------------------------------first analyze command-------------------
         set ok [analyze 1]
         # ------------------------------------------if convergence failure-------------------
         if {$ok != 0} {
            # if analysis fails, we try some other stuff
            # performance is slower inside this loop global maxNumIterStatic;     # max no. of i
            if {$ok != 0} {
               puts "Trying Newton with Initial Tangent .."
               test NormDispIncr   $Tol 2000 0
               algorithm Newton -initial
               set ok [analyze 1]
               test $testTypeStatic $TolStatic      $maxNumIterStatic    0
```

Uniform Excitation 1D: define

```
# Uniform Earthquake ground motion (uniform acceleration input at all support nodes)
set GMdirection 1;            # ground-motion direction
set GMfile "H-e12140" ;            # ground-motion filenames
set GMfact 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 PEER strong motion database file, extracts dt from the header and
# to the format OpenSees expects for Uniform/multiple-support ground motions
source ReadSMDFile.tcl;  # read in procedure Multinition
# Uniform EXCITATION: acceleration input
set inFile $GMdir/$GMfile.at2
set outFile $GMdir/$GMfile.g3;  # set variable holding new filename (PEER files have
ReadSMDFile $inFile $outFile dt;   # call procedure to convert the ground-motion fil
set GMfatt [expr $g*$GMfact];     # data in input file is in g Unifts -- ACCELERAT
set AccelSeries "Series -dt $dt -filePath $outFile -factor  $GMfatt";  # time series
pattern UniformExcitation  $IDloadTag  $GMdirection -accel  $AccelSeries  ;     #
```



Uniform Excitation 1D: damping

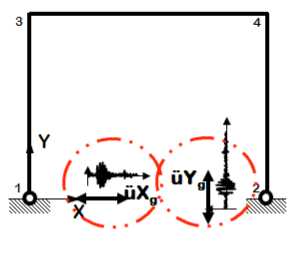
```
# define DAMPING----------------------------------------------------------------------------
# apply Rayleigh DAMPING from $xDamp
# D=$alphaM*M + $betaKcurr*Kcurrent + $betaKcomm*KlastCommit + $beatKinit*$Kinitia
set xDamp 0.02;          # 2% damping ratio
set lambda [eigen 1];        # eigenvalue mode 1
set omega [expr pow($lambda,0.5)];
set alphaM 0.;          # M-prop. damping; D = alphaM*M
set betaKcurr 0.;          # K-proportional damping;      +beatKcurr*KCurrent
set betaKcomm [expr 2.*$xDamp/($omega)];      # K-prop. damping parameter;   +betaKcom
set betaKinit 0.;          # initial-stiffness proportional damping      +beatKinit*Kini
rayleigh $alphaM $betaKcurr $betaKinit $betaKcomm;          # RAYLEIGH damping
```

Uniform Excitation 1D: analyze

```
set Nsteps [expr int($TmaxAnalysis/$DtAnalysis)];
set ok [analyze $Nsteps $DtAnalysis];          # actually perform
if {$ok != 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 == 0} {
    set controlTime [getTime]
    set ok [analyze 1 $DtAnalysis]
    if {$ok != 0} {
      puts "Trying Newton with Initial Tangent .."
      test NormDispIncr   $Tol 1000  0
      algorithm Newton -initial
      set ok [analyze 1 $DtAnalysis]
      test $testTypeDynamic $TolDynamic $maxNumIterDynam
      algorithm $algorithmTypeDynamic
    }
    if {$ok != 0} {
      puts "Trying Broyden .."
      algorithm Broyden 8
      set ok [analyze 1 $DtAnalysis]
      algorithm $algorithmTypeDynamic
```


Uniform Excitation 2D: define

```
# ------------------------------------------------------------
# Example4. 2D Portal Frame--  Dynamic EQ input analysis -- bidirectional
#                         Silvia Mazzoni & Frank McKenna, 2006
# execute this file after you have built the model, and after you apply gravit
#

# Bidirectional Uniform Earthquake ground motion (uniform acceleration inpu
set iGMfile "H-E01140 H-e12140" ;       # ground-motion filenames, should b
set iGMdirection "1 2";          # ground-motion direction
set iGMfact "1.5 0.25";          # 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
```




```
# define DAMPING--------------------------------------
# apply Rayleigh DAMPING from $xDamp
# D=$alphaM*M + $betaKcurr*Kcurrent + $betaKcomm
set xDamp 0.02;                # 2% damping ratio
set lambda [eigen 1];          # eigenvalue mode 1
set omega [expr pow($lambda,0.5)];
set alphaM 0.;              # M-prop. damping; D = alphaM
set betaKcurr 0.;                # K-proportional damping;
set betaKcomm [expr 2.*$xDamp/($omega)];       # K-pro
set betaKinit 0.;                # initial-stiffness proport
rayleigh $alphaM $betaKcurr $betaKinit $betaKcomm;
```

```
# the following commands are unique to the Uniform Earthquake excitation
set IDloadTag 400;   # for uniformSupport excitation
# read a PEER strong motion database file, extracts dt from the header and converts t
# to the format OpenSees expects for Uniform/multiple-support ground motions
source ReadSMDFile.tcl;   # read in procedure Multinition
# Uniform EXCITATION: acceleration input
foreach GMdirection $iGMdirection GMfile $iGMfile GMfact $iGMfact {
  incr IDloadTag;
  set inFile $GMdir/$GMfile.at2
  set outFile $GMdir/$GMfile.g3;          # set variable holding new filename (PEER
  ReadSMDFile $inFile $outFile dt;        # call procedure to convert the ground-mot
  set GMfatt [expr $g*$GMfact];           # data in input file is in g Unifts -- ACCEL
  set AccelSeries "Series -dt $dt -filePath $outFile -factor  $GMfatt";       # time
  pattern UniformExcitation  $IDloadTag  $GMdirection -accel  $AccelSeries  ;    #
}
```

Silvia Mazzoni, OpenSees Days 2010

Uniform Excitation 2D: analyze

```
# ---------- set up analysis parameters
source LibAnalysisDynamicParameters.tcl;  # constraintsHandler,DOFnum

set Nsteps [expr int($TmaxAnalysis/$DtAnalysis)];
set ok [analyze $Nsteps $DtAnalysis];          # actually perform analysis;

if {$ok != 0} {     ;                 # analysis was not successful.
   # -------------------------------------------------------------------
   # change some analysis parameters to achieve convergence
   # performance is slower inside this loop
   #    Time-controlled analysis
   set ok 0;
   set controlTime [getTime];
   while {$controlTime < $TmaxAnalysis && $ok == 0} {
      set controlTime [getTime]
      set ok [analyze 1 $DtAnalysis]
      if {$ok != 0} {
         puts "Trying Newton with Initial Tangent .."
         test NormDispIncr   $Tol 1000  0
         algorithm Newton -initial
         set ok [analyze 1 $DtAnalysis]
         test $testTypeDynamic $TolDynamic $maxNumIterDynamic  0
         algorithm $algorithmTypeDynamic
      }
      if {$ok != 0} {
         puts "Trying Broyden .."
         algorithm Broyden 8
         set ok [analyze 1 $DtAnalysis]
```


Questions, or statements:

The OpenSees Community Forum:

http://opensees.berkeley.edu/community/index.php

which can be accessed from:

http://opensees.berkeley.edu