ANEXO A

Implementación en Matlab

Contenido

- Datos Iniciales
- Precalculos
- Matriz de Rigidez Global
- Datos Iniciales del NSP
- Algoritmo del analisis NSP
- Fuerza en los resortes - comportamiento del material

Datos Iniciales

```
function [Pv,Rotulas,uf,rotaciones,Maplt,cr,u_lt,cv_hy,Ugeneral,CONNP,XYP,L]=

NSP_Pushover(tol,umax,CONN,XY,PROP,elemenrotulas,columnasportico,cm,Lp,dut)

PROP(:,3)=PROP(:,3)*10000;   % Área de las Secciones Ocupadas
g=3;                         % Grados de libertad de cada nodo
E=PROP(:,2);                 % Módulo de Elasticidad del Material
A=PROP(:,3);                 % Área de la Sección
PROP(:,3)=PROP(:,3)*10000;   % Área Infinitamente rígida
I=PROP(:,4);                 % Inercia de la Sección
z=PROP(:,5);                 % Módulo de Sección Plástica
% Comprueba si el usuario coloco cargas gravitacionales o no
if size(Lp,1)==size(find(Lp(:,1)==0));
    Lp=[];                   % Lp: Vector de cargas gravitacionales del sistema
end

[CONEX,Mp, Mpl,equalDOF,CONN,XY,kr,cr,masterpiso]=
rotulas_equaldof1(CONN,XY,PROP,columnasportico,elemenrotulas,cm);
% CONEX: Matriz de desface de conexiones. Mp: Vector de Momentos plasticos de los elementos
```

```
% Mpl: Momentos plasticos en los resortes rotacionales. CONN:Matriz de conecciones de los elementos
% XY: Matriz de cordenadas de los nodos. kr: Rigidez de los resortes. cr: comportamiento de los reso
% equalDOF: Matriz de nodos Master y Slave
Mapl=zeros(size(Mpl));      % Vector de los Momentos Aplicados
db=CONN(:,4);               % Desface Inicial
de=CONN(:,5);               % Desface final
fs=PROP(:,7);               % Factor de Forma de la Sección
v=PROP(:,6);                % Módulo de Poisson
As=(A./fs)*10^15;           % Área de Corte
alfa=PROP(:,8);             % Coeficiente Térmico
rb=CONEX(:,2);              % Parámetro de rigidez inicial
re=CONEX(:,3);              % Parámetro de rigidez final
Fext=[];                    % Vector de Fuerzas Externas
Rest=[];                    % Vector de Fuerzas Resistentes
for i=1:size(XY,1);
Fext=[Fext;(XY(i,7:9))'];   %Los valores XY(i,7:9) contienen los valores de fuerzas
Rest=[Rest;(XY(i,4:6))'];   %Los valores XY(i,4:6) contienen las restricciones del nodo
end

hmax=max(XY(:,3));          % hmax:Obtiene la altura máxima del pórtico
p_hmax=find(XY(:,3)==hmax); % p_hmax:Obtiene las posiciones de todos los nodos con altura máxima
p_hmax_p=find(XY(p_hmax,10)==0);% p_hmax_p: Obtiene las posiciones de los nodos principales
t=p_hmax(p_hmax_p(1,1),1)*3-2;  % t: Encuentra el grado de libertad en x del Nodo de Control
```

Precálculos

```
for i=1:size(CONN,1)
    NI=find(CONN(i,2)==XY(:,1));  % Posición del nodo inicial
    NJ=find(CONN(i,3)==XY(:,1));  %Posición del nodo final
    Lo(i)=sqrt((XY(NI,2)-XY(NJ,2))^2+(XY(NI,3)-XY(NJ,3))^2);% Longitud inicial de los elementos
    L(i)=Lo(i)-de(i)-db(i);                                  % Longitud libre de los elementos
    a(i)=atan((XY(NJ,3)-XY(NI,3))/(XY(NJ,2)-XY(NI,2)));     % Ángulo de dirección de los elementos
    if XY(NJ,3)-XY(NI,3)>0 && XY(NJ,2)-XY(NI,2)<0           % Segundo cuadrante
        a(i)  = a(i)+pi();
    elseif XY(NJ,3)-XY(NI,3)<0 && XY(NJ,2)-XY(NI,2)<0       % Tercer cuadrante
        a(i)  = a(i)-pi();
    end
end
L=L';                                                       % Vector de longitudes
a=a';                                                       % Vector de ángulos
```

Matriz de Rigidez Global

```
[kg,B,T,R]=MATRICES_K(rb,re,L,E,I,a,db,de,v,A,As);
% kg: Matriz de los elementos. B: Matriz de Transformación de Ejes
```

```
% T: Matriz de Transformación de conexiones de desface. R: Parámetro de rigidez
[K]= MATRIZ_K(XY,CONN,kg,g);    % Matriz de rigidez global elástica de la estructura

%Efectos P-Delta
if size(Lp,1)~=0
    masterpiso(1)=1;
    Kg=zeros(size(K));                          % Matriz de rigidez Geométrica
    for i=1:(size(masterpiso,1))-1

    pd{i}= Matriz_Pdelta(Lp(i),L(i),E(i),A(i)); % Coeficientes de rigidez-matriz geométrica
    NI=masterpiso(i);                           % NI: Posición del nodo inicial
    NJ=masterpiso(i+1);                         % NJ: Posición del nodo final
    Kg((3*NI-2):(3*NI),(3*NI-2):(3*NI))=Kg((3*NI-2):(3*NI),(3*NI-2):(3*NI))+pd{i}((1:3),(1:3));
    Kg((3*NJ-2):(3*NJ),(3*NJ-2):(3*NJ))=Kg((3*NJ-2):(3*NJ),(3*NJ-2):(3*NJ))+pd{i}((4:6),(4:6));
    Kg((3*NI-2):(3*NI),(3*NJ-2):(3*NJ))=Kg((3*NI-2):(3*NI),(3*NJ-2):(3*NJ))+pd{i}((1:3),(4:6));
    Kg((3*NJ-2):(3*NJ),(3*NI-2):(3*NI))=Kg((3*NJ-2):(3*NJ),(3*NI-2):(3*NI))+pd{i}((4:6),(1:3));

    end
    K=K+Kg;                                 % K: Matriz de rigidez global de la estructura
end

[K2,Fext2,gS,gM,KS,uf]= MATRIZ_EQUAL1(K,Fext,equalDOF);
% Fext2:Vector de Fuerzas puntuales.gS:Grados Slave.gM:Grados Master.KS: Matriz de los resortes
% uf:Vector de conexión de los resortes. K2:Matriz de rigidez global simplificada
p2=p;                                       % Grados de libertad desconocidos del equalDOF
for i=1:length(gS)
    p2(find(gS(i)==p2))=[];                 % Vector con los grados de libertad
end
cro={};                                     % Comportamiento completo de los resortes rotacionales
for i=1:size(KS,2)
    FSac{i}=zeros(2,1);
    yteta=cr{i}(:,1);
    xteta=cr{i}(:,2);
    %Opensees
    x_op=zeros(size(yteta,1)*2-1,1);
    y_op=zeros(size(yteta,1)*2-1,1);
    cont=1;
    for j=size(yteta,1):-1:1
        x_op(cont,1)=-xteta(j,1);
        y_op(cont,1)=-yteta(j,1);
        if cont~=size(yteta,1);
            x_op(cont+size(yteta,1),1)=xteta(cont+1,1);
            y_op(cont+size(yteta,1),1)=yteta(cont+1,1);
```

```
        end
        cont=cont+1;
    end
    cro{i}(:,1)=y_op;
    cro{i}(:,2)=x_op;
end
ks=100000;              % Rigidedz del resorte en el nodo de control
K2(t,t)=K2(t,t)+ks;     % Se coloca el resorte en la matriz de rigidez
F2=Fext2;               % Fuerzas Externas simplificadas por equalDOF
F=F2(p2,1);             % Patron de fuerzas externas unitarias simplificadas por equalDOF
a=find(F==Fmax);        % Posición del grado de libertad longitudinal del Nodo de Control
```

Datos Iniciales del NSP

```
cr=cro;               % Comportamiento de las rotulas del material completo
n=1;                  % Contador del vector de resultados Pv
Pv=[];                % Vector de resultados Fuerza/Desplazamiento/Subiteraciones
Pv(1,:)=[0,0,0];      %Vector de resultados del pushover
m=0;                  % Contador de todas las iteraciones del programa
Rotulas={};           % Vector de posición de Resortes
rotaciones=[];        % Vector que guarda las rotaciones de todos los resortes
Maplt=[];             % Vector que guarda los momentos de todos los resortes
% Vectores que indican el cambio de pendiente en el comportamiento del material
% Cposant: Comportamiento anterior, Cposact: Comportamiento actual
Cposant=ones(size(uf,2),1);Cposact=ones(size(uf,2),1);
diract=ones(size(uf,2),1);% Indicador si el resorte esta en tensión o compresión
Fr=zeros(size(p2));  % Fuerzas Resistentes
Fa=zeros(size(p2));  % Fuerzas Actuantes
u2=zeros(3*size(XY,1),1); % Vector de desplazamientos por iteración de todos los
                          % grados de libertad del sistema
utot=zeros(size(XY,1)*3,1);% Vector de desplazamientos acomuados de todos los
                           % grados del sistema
giro=zeros(size(uf,2),1); % Obtiene las rotaciones de los resortes de la viga
                          % y columnas
U=zeros(size(p2));        % Vector de desplazamientos de todos los grados de
                          % libertad Principales (Master) del sistema
Ut=0;                     % Desplazamiento en el nodo de control
lambda=0;                 % Coeficiente de relación entre fuerzas externas y resistentes
est_rot=zeros(size(cr,2),1);% Vector que evita que el código considere nuevamente la
                            % formación de la rótula plástica en el mismo resorte
comp_est_rot=est_rot;
cv_hy={};                 % Comportamiento de cada resorte rotacional de la estructura
Ugeneral=XY(:,1);         % Denominación de los resortes y desplazamientos durante cada
                          % iteración/ guarda cada iteración
```

```
NGDL=size(XY,1)*3;       %Variable de almacenamiento de desplazamientos
```

Algoritmo del analisis NSP

```
 while Ut<umax
    Ut=Ut+dut;
    Fs=ks*dut;           % Fuerza resistente del resorte
    Fa(a,1)=Fa(a,1)+Fs;  % Fuerzas Actuantes en la Estructura
    deltaF=Fa+Fr;        % Vector de fuerzas desbalanceado
    pos1=[];             % Guardan los resortes que fallan
    w=0;                 % No. iteraciones para igualarse las fuerzas internas
                         % con las externas
    %Se obtiene la norma del vector desbalaceado y se compara con la tolerancia.
     while norm(deltaF)>tol

        w=w+1;           % Acumulador de iteraciones
        Kpp=K2(p2,p2);   % Matriz de rigidez de los grados de libertad desconocidos del
                         % equalDOF (Master)
        Uu=inv(Kpp)*deltaF;  % Vector de desplazamientos debido al desbalance
        Uf=inv(Kpp)*F;       % Vector de desplazamientos debido a las fuerzas unitarias
        Ru=ks*((U(a,1)+Uu(a,1))-Ut); % Fuerzas que absorbe la estructura en el nodo de control
        Rf=ks*(Uf(a,1));             % Fuerzas que actuan en el nodo de control
        dlambda=-(Ru/Rf);            % Factor de relación de fuerzas
        lambda=lambda+dlambda;       % Variable acomulativa del factor escalar debido a la
                                     % relación de fuerzas
        Uu=Uu+dlambda*Uf;
        U=U+Uu;
        Fa=lambda*F;
        Fa(a,1)=Fa(a,1)+ks*Ut;
        u2(p2,1)=Uu;
        for i=size(gM,2):-1:1
            u2(gS(i),1)=u2(gM(i),1);
        end
        utot=utot+u2;
        fele2={}; pele2={}; fele2x={}; pele2x={};
        Fint=zeros(NGDL,1);
        for i=1:size(CONN,1)
            NI=CONN(i,2);
            NJ=CONN(i,3);
            % Fuerzas internas de los elementos en coordenadas globales con el desface
            fele2x{i}=kg{i}*[utot(3*NI-2:3*NI);utot(3*NJ-2:3*NJ)];
            % Fuerzas internas de los elementos en coordenadas locales con el desface
            pele2x{i}=B{i}*fele2x{i};
            % Fuerzas internas de los elementos en coordenadas locales sin el desface
            pele2{i}=inv(T{i})*pele2x{i};
```

```
    % Fuerzas internas de los elementos en coordenadas globales sin el desface
    fele2{i}=inv(B{i})*pele2{i};
    % Vector de fuerzas internas generales de los elementos
    Fint(3*NI-2,1)=Fint(3*NI-2,1)+fele2x{i}(1,1);
    Fint(3*NI-1,1)=Fint(3*NI-1,1)+fele2x{i}(2,1);
    Fint(3*NI,1)=Fint(3*NI,1)+fele2x{i}(3,1);
    Fint(3*NJ-2,1)=Fint(3*NJ-2,1)+fele2x{i}(4,1);
    Fint(3*NJ-1,1)=Fint(3*NJ-1,1)+fele2x{i}(5,1);
    Fint(3*NJ,1)=Fint(3*NJ,1)+fele2x{i}(6,1);
end
```

Fuerza en los resortes - comportamiento del material

```
for i=1:size(uf,2)
        giro(i)=utot(uf{i}(1,1))-utot(uf{i}(2,1));% Obtiene el estado de giro
        dgiro(i)=u2(uf{i}(1,1))-u2(uf{i}(2,1));   % Incremental del giro de los resortes
        if dgiro(i)>0;
            diract(i)=1;       % Si el giro va hacia la derecha se coloca 1 Tensión
        else
            diract(i)=-1;      % Si el giro va hacia la izquierda se coloca -1 Compresión
        end
        if n==1
            Cposact(i)=Cposact(i)*diract(i);
            dirant(i)=diract(i);
            Cposant(i)=Cposact(i);
        end
        % Indica la posición del punto 0 dentro del comportamiento de los resortes
        O=(((size(cr{i},1))-1)/2)+1;
        if dirant(i)~=diract(i) && abs(Cposant(i))==2
            if diract(i)==-1
                ingiro=giro_a(i)-cro{i}(O+1,2);  %Incremento positivo
                %Incremento negativo si la pendiente disminuye
                inMom=(ingiro)*(kr{i}(abs(Cposant(i)),1));
                Cposact(i)=1;
            end
            if diract(i)==1
                ingiro=giro_a(i)-cro{i}(O-1,2);  %Incremento negativo
                %Incremento positivo si la pendiente disminuye
                inMom=(ingiro)*(kr{i}(abs(Cposant(i)),1));
                Cposact(i)=-1;
            end
        % cro: Comportamiento del resorte con desface cuando hay un ciclo de descarga
        % cr: Comportamiento del resorte.
            cr{i}(:,1)=cro{i}(:,1)+ones(size(cr{i},1),1)*inMom;
```

```
                cr{i}(:,2)=cro{i}(:,2)+ones(size(cr{i},1),1)*ingiro;
            end

            for j=1:(size(cr{i},1)-1)
                if giro(i)>cr{i}(j,2) && giro(i)<cr{i}(j+1,2)
                    Cposact(i)=j-0;
                    if Cposact(i)>=0
                       Cposact(i)=Cposact(i)+1;
                    end
                    j=(size(cr{i},1)-1);
                end
            end
            % Momentos dependiendo del tramo
            M=cr{i}(0+(Cposact(i))-(abs(Cposact(i))/(Cposact(i))),1);
            if abs(Cposact(i))==2
              g=((Cposact(i))/(Cposact(i)))*(giro(i)-cr{i}(0+(Cposact(i))-
              (abs(Cposact(i))/(Cposact(i))),2));
            else
                g=(giro(i)-cr{i}(0+(Cposact(i))-(abs(Cposact(i))/(Cposact(i))),2));
            end
            % Tramo del comportamiento del resorte
            C=abs(Cposact(i));
            % Pendiente dependiendo tramo del resorte
            kp=kr{i}(C,1);
            % Momento aplicado en el resorte
            Mapl(i)=M+(g)*kp;
            % Momentos generados en el grado de libertad rotacional del resorte
            FS{i}=[Mapl(i);-Mapl(i)];
    % Coloca los momentos generados en el vector de fuerzas internas de la estructura
            Fint(uf{i}(1,1),1)=Fint(uf{i}(1,1),1)+FS{i}(1,1);
            Fint(uf{i}(2,1),1)=Fint(uf{i}(2,1),1)+FS{i}(2,1);

            if Cposact(i)~=Cposant(i)
                % Pendiente anterior dependiendo tramo del resorte
                Knant=kr{i}(abs(Cposant(i)),1);
                % Pendiente actual dependiendo tramo del resorte
                Kn=kr{i}(abs(Cposact(i)),1);
                % Rigidez de los resortes rotacionales
                KS{i}=[Kn,-Kn;-Kn,Kn];
% Incorporación de la rigidez de los resortes en la matriz de rigidez de la estructura
                K2(uf{i}(1,1),uf{i}(1,1))=K2(uf{i}(1,1),uf{i}(1,1))- Knant+Kn;
                K2(uf{i}(1,1),uf{i}(2,1))=K2(uf{i}(1,1),uf{i}(2,1))+ Knant-Kn;
                K2(uf{i}(2,1),uf{i}(1,1))=K2(uf{i}(2,1),uf{i}(1,1))+ Knant-Kn;
```

```
                    K2(uf{i}(2,1),uf{i}(2,1))=K2(uf{i}(2,1),uf{i}(2,1))- Knant+Kn;
% Selecciona solo la primera ves que el resorte rotacional se convierte en una rótula plástica
                    if abs(Cposact(i))==2
                        idct=1;
                        if est_rot(i)==0;
                            est_rot(i)=idct;
                            pos1=[pos1,i];
                        end
                    end
                end
            end
            %Se suman en los gdl master las fuerzas de los gdl slave
            for i=1:size(gM,2)
                Fint(gM(i),1)=Fint(gM(i),1)+Fint(gS(i),1); Fint(gS(i),1)=0;
            end
            %Efectos P-Delta-Vector de cargas gravitacionales
            if size(Lp,1)~=0
                for i=1:size(masterpiso,1)
                    if i==1

                    NI=(masterpiso(i))*3-2;
                    NJ=(masterpiso(i+1))*3-2;
                    Fint(NI,1)=Fint(NI,1)+Kg(NI,NI)*utot(NI)+Kg(NI,NJ)*utot(NJ);

                    elseif i==size(masterpiso,1)

                    NH=(masterpiso(i-1))*3-2;
                    NI=(masterpiso(i))*3-2;
                    Fint(NI,1)=Fint(NI,1)+Kg(NI,NH)*utot(NH)+Kg(NI,NI)*utot(NI);

                    else

                    NH=(masterpiso(i-1))*3-2;
                    NI=(masterpiso(i))*3-2;
                    NJ=(masterpiso(i+1))*3-2;
                    Fint(NI,1)=Fint(NI,1)+Kg(NI,NH)*utot(NH)+Kg(NI,NI)*utot(NI)
                    +Kg(NI,NJ)*utot(NJ);

                    end
                end
            end
            Fr=-Fint(p2,1);
            Fr(a,1)=Fr(a,1)-ks*U(a,1);
```

```
        deltaF=Fa+Fr;

        Mapl_a=Mapl;
        giro_a=giro;
        dirant=diract;
        Cposant=Cposact;

    end
    giros(n,:)=[giro'];
    momentos(n,:)=[Mapl'];
    Cpos(n,:)=[Cposact'];
    d(n,:)=[diract'];
    posll=[];
    if size(pos1,2)~=0
        for j=1:size(pos1,2)
           if comp_est_rot(pos1(1,j),1)==0;
               comp_est_rot(pos1(1,j),1)=1;
               posll=[posll,pos1(1,j)];
           end
        end
        pos1=posll;
        m=m+1;
        Rotulas{m}=[Ut,pos1];
        rotaciones=[rotaciones,(giro)];
        Maplt=[Maplt,(Mapl)];
    end
    Paux=lambda;               % Fuerza longitudinal en el nodo de control

%Suma cada incremento de fuerza
%Suma cada incremento en el nodo de control
    n=n+1;Pv(n,:)=[Paux,Ut,w]; % Se agrega tanto la fuerza como el desplazamiento en un nodo
    Xx=[];                     % Desplazamiento en x de cada nodo
    Yy=[];                     % Desplazamiento en y de cada nodo
    Zz=[];                     % Rotación de cada nodo
  for i=1:size(XY,1)
        Xx=[Xx;utot((i*3)-2,1)];
        Yy=[Yy;utot((i*3)-1,1)];
        Zz=[Zz;utot((i*3),1)];
  end
    Ugeneral=[Ugeneral,Xx,Yy];

    if Paux<0
        Ut=umax;
    end
```

```
 end
 u_lt=0;
 cr=cro;
for i=1:size(cro,2);
    cv_hy{i}(:,1)=giros(:,i);
    cv_hy{i}(:,2)=momentos(:,i);
end
XYP=XY;
CONNP=CONN;

end
```

ANEXO B

Implementacción en Opensees

```
# --------------------------------------------------------------------------------------
# Ejemplo : Edificio de un Piso
# Tipo: Modelo con rotulas plasticas en la columna y en la viga de un piso
# Primeras Pruebas
# Nombres: Paola Mejia, Jorge Rivera
# Unidades: Toneladas, Centimetros.
##########################################################################################
#          Set Up & Source Definition
##########################################################################################
wipe all; # clear memory of past model definitions
model BasicBuilder -ndm 2 -ndf 3; # Define the model builder, ndm = #dimension, ndf = #dofs
#source DisplayModel2D.tcl; # procedure for displaying a 2D perspective of model
#source DisplayPlane.tcl; # procedure for displaying a plane in a model with very small stiffness
#logFile Summary.out # save the warnning and error messanges that the running scripts

##########################################################################################
#          Define Analysis Type
##########################################################################################
# Define type of analysis:  "pushover" = pushover
set analysisType "pushover";
if {$analysisType == "pushover"} {
set dataDir Un_Piso_Un_Vano_Ejemplo_Tesis; # name of output folder
file mkdir $dataDir; # create output folder
}

##########################################################################################
#          Define Building Geometry, Nodes, and Constraints
##########################################################################################
```

```
## define structure-geometry parameters
set NStories 1; # Numero de pisos
set NBays 1; # Numero de banos
set WBay      [expr 20.0*30.48];                  # Ancho del bano en cm
set HStory1   [expr 15.0*30.48];                 # Altura del piso en cm
set HBuilding [expr $HStory1*1.0]; # Altura del Edificio en cm

## calculate locations of beam/column joints:
set Pier1  0.0;                                  #Eje de referencia de la columna 1
set Pier2  [expr $Pier1 + $WBay]; #Eje de referencia de la columna 2
set Pier3  [expr $Pier2 + $WBay]; # P-delta column line
set Floor1 0.0;                                  #Altura de Partida
set Floor2 [expr $Floor1 + $HStory1]; #Altura de referencia del piso 1

## calculate joint offset distance for beam plastic hinges
set phlat23 [expr 0.0]; #lateral dist from beam-col joint to loc of hinge on Floor 2

## calculate nodal masses -- lump floor masses at frame nodes
set g 980.7; # acceleration due to gravity
set Floor2Weight 270.0; # weight of Floor 2 in ton
set WBuilding  [expr $Floor2Weight]; # total building weight
set NodalMass2 [expr ($Floor2Weight/$g) / (2.0)]; # mass at each node on Floor 2
set Negligible 1e-15; # a very smnumber to avoid problems with zero
## define nodes and assign masses to beam-column intersections of frame
# command:  node nodeID xcoord ycoord -mass mass_dof1 mass_dof2 mass_dof3
# nodeID convention:  "xy" where x = N.colum # and y = Floor #
##Nodos de la base
node 11 $Pier1 $Floor1;
node 21 $Pier2 $Floor1;
node 31 $Pier3 $Floor1; #nodo de la columna fantasma
##Nodos del piso 1
node 12 $Pier1 $Floor2 -mass $Negligible $Negligible $Negligible;
node 22 $Pier2 $Floor2 -mass $Negligible $Negligible $Negligible;
node 32 $Pier3 $Floor2; #nodo de la columna fantasma

## define extra nodes for plastic hinge rotational springs
# nodeID convention:  "xya" where x = No. Column #, y = Floor #,
# a = location relative to beam-column joint
# "a" convention: 2 = left; 3 = right;
# "a" convention: 6 = below; 7 = above;

# column hinges at bottom of Story 1 (base)
node 117 $Pier1 $Floor1;
```

```
node 217 $Pier2 $Floor1;

# column hinges at bottom of Story 2 (base)
node 126 $Pier1 $Floor2;
node 226 $Pier2 $Floor2;

# column hinges at top of Story 1
node 326 $Pier3 $Floor2; # zero-stiffness spring will be used on p-delta column

# beam hinges at Floor 2
node 122 $Pier1 $Floor2;
node 223 $Pier2 $Floor2;

# constrain beam-column joints in a floor to have the same lateral
displacement using the "equalDOF" command
# command: equalDOF $MasterNodeID $SlaveNodeID $dof1 $dof2...
#equalDOF 12 22 1; # Floor 2:  Pier 1 to Pier 2
equalDOF 12 22 1;
#equalDOF 12 32 1;
equalDOF 11 117 1 2;
equalDOF 21 217 1 2;

equalDOF 12 126 1 2;
equalDOF 12 122 1 2;

equalDOF 22 226 1 2;
equalDOF 22 223 1 2;

equalDOF 12 32 1;
equalDOF 12 326 1;

equalDOF 32 326 2;

# assign boundary condidtions
# command:  fix nodeID dxFixity dyFixity rzFixity
# fixity values: 1 = constrained; 0 = unconstrained
# fix the base of the building; pin P-delta column at base
fix 11 1 1 1;
fix 21 1 1 1;
fix 31 1 1 0; # P-delta column is pinned

##########################################################################################
#          Define Section Properties and Elements
```

```
##############################################################################################
# ACERO
set Fy   3.5
set E    2100
set Roty 0.005
set b    -0.1
##Viga
# W27x102
set Z_W18x35 [expr 305.0*2.54*2.54*2.54]
set A_W18x35 [expr 30.0*2.54*2.54*10000000000000000]
set I_W18x35 [expr 3620*2.54*2.54*2.54*2.54]

##Columna
# W21x111
set Z_W24x84 [expr 279.0*2.54*2.54*2.54]
set A_W24x84 [expr 32.7*2.54*2.54*100000000000000000]
set I_W24x84 [expr 2670*2.54*2.54*2.54*2.54]

# uniaxialMaterial Steel01 Tag Fy E b
set Kb_W18x35  [expr $b*(($Z_W18x35*$Fy)/$Roty)]
set Kb_W24x84  [expr $b*(($Z_W24x84*$Fy)/$Roty)]

#Viga
#uniaxialMaterial ElasticMultiLinear 1 -strain 0 0.005 0.055 -stress 0 [expr $Z_W18x35*$Fy] [expr $Z
uniaxialMaterial Steel01 1 [expr ($Z_W18x35*$Fy)]   [expr ($Z_W18x35*$Fy)/$Roty]   $b
#Columna
#uniaxialMaterial ElasticMultiLinear 2 -strain 0 0.005 0.055 -stress 0 [expr $Z_W24x84*$Fy] 0
uniaxialMaterial Steel01 2 [expr ($Z_W24x84*$Fy)]   [expr ($Z_W24x84*$Fy)/$Roty]   $b

##############################################################################################
#          Define Elementos Rotulas
##############################################################################################

#geomTransf Linear 1; #INCLUIR NO LINEALIDAD GEOMETRICA [Linear PDelta Corotational]
set PDeltaTransf 1;
geomTransf PDelta $PDeltaTransf;  # PDelta transformation

## ROTULAS
# Tag NI NF -mat Material -dir [X-1 Y-2 XY-3]
# Rotulas en las Columnas
element zeroLength  1175 11 117 -mat 2 -dir 6
element zeroLength  2175 21 217 -mat 2 -dir 6
```

```
# Rotulas en las Columnas
element zeroLength  1265 12 126 -mat 2 -dir 6
element zeroLength  2265 22 226 -mat 2 -dir 6

# Rotulas en las Vigas
element zeroLength  1225 12 122 -mat 1 -dir 6
element zeroLength  2235 22 223 -mat 1 -dir 6
# Rotulas PDelta
element zeroLength  326 32 326 -mat 2 -dir 6

## ELEMENTOS ENTRE ROTULAS
# Tag NI NF Area Elasticidad Inercia Transformacion -mass 0.0000000001
#Columnas
element elasticBeamColumn 111 117 126 $A_W24x84 $E $I_W24x84 1 -mass $Negligible; #Columna 1
element elasticBeamColumn 121 217 226 $A_W24x84 $E $I_W24x84 1 -mass $Negligible; #Columna 2
#Vigas
element elasticBeamColumn 212 122 223 $A_W18x35 $E $I_W18x35 1 -mass $Negligible; #Viga

# define p-delta columns and rigid links
set TrussMatID 600; # define a material ID
set Arigid 1000.0; # define area of truss section (make much larger than A of frame elements)
set Irigid 100000.0; # moment of inertia for p-delta columns
uniaxialMaterial Elastic $TrussMatID $E; # define truss material
# rigid links
# command: element truss $eleID $iNode $jNode $A $materialID
# eleID convention:  6xy, 6 = truss link, x = Bay #, y = Floor #
element truss  622 22 32 $Arigid $TrussMatID; # Floor 2

# p-delta columns
# eleID convention:  7xy, 7 = p-delta columns, x = Pier #, y = Story #
element elasticBeamColumn  731  31  326 $Arigid $E $Irigid $PDeltaTransf -mass $Negligible;
# Story 1

#########################################################################
#              Gravity Loads & Gravity Analysis
#########################################################################
# apply gravity loads
#command: pattern PatternType $PatternID TimeSeriesType
pattern Plain 101 Constant {

# point loads on leaning column nodes
# command: load node Fx Fy Mz
# point loads on frame column nodes
```

```
set P_F2 [expr -1.0*($Floor2Weight)]; # load on each frame node in Floor 2
# Floor 2 loads
load 32 0.0 $P_F2 0.0;
}

# Gravity-analysis: load-controlled static analysis
set Tol 1.0e-6;   # convergence tolerance for test
constraints Plain;   # how it handles boundary conditions
numberer Plain;   # renumber dof's to minimize band-width (optimization)
# how to store and solve the system of equations in the analysis (large model: try UmfPack)
system BandGeneral;
# determine if convergence has been achieved at the end of an iteration step
test NormDispIncr $Tol 6;
# use Newton's solution algorithm: updates tangent stiffness at every iteration
algorithm Newton;
set NstepGravity 10;   # apply gravity in 10 steps
set DGravity [expr 1.0/$NstepGravity];  # 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"


##########################################################################################
#                                Analysis Section                                        #
##########################################################################################
#              Pushover Analysis                          #
#############################################################################
if {$analysisType == "pushover"} {
puts "Running Pushover..."
# assign lateral loads and create load pattern:  use ASCE 7-10 distribution
set lat2 1; # force on each frame node in Floor 2
pattern Plain 200 Linear {
load 12 $lat2 0.0 0.0;
}


# displacement parameters
set NodeControl 12; # node where disp is read for disp control
set IDctrlDOF 1; # degree of freedom read for disp control (1 = x displacement)
```

```
set d_ult [expr 0.1*$HBuilding]; # maximum displacement of pushover: 10% roof drift
set Dincr [expr 0.01]; # displacement increment

# analysis commands
constraints Plain;   # how it handles boundary conditions
numberer Plain; # renumber dof's to minimize band-width (optimization)
# how to store and solve the system of equations in the analysis (large model: try UmfPack)
system BandGeneral;
test EnergyIncr 1.e-008 1000 0; # tolerance, max iterations
# use Newton's solution algorithm: updates tangent stiffness at every iteration
algorithm Newton -initial;
# use displacement-controlled analysis
integrator DisplacementControl  $NodeControl   $IDctrlDOF $Dincr;
# define type of analysis: static for pushover
analysis Static;
# number of pushover analysis steps
set Nsteps [expr int($d_ult/$Dincr)];
# this will return zero if no convergence problems were encountered
# set ok [analyze $Nsteps];
set ok 0   ;      # Flag que indica convergencia
set Vcurr [getTime] ;  # Cortante basal actual
set controlDisp [nodeDisp $NodeControl  1]  ;  # Controlando desplazamiento de nodo

while {$controlDisp <= $d_ult && $Vcurr >= 0} {   #$ok == 0
set controlDisp [nodeDisp $NodeControl  1]
set ok [analyze 1]
# Si el analisis falla se puede tratar de variar los parametros para que converja
if {$ok != 0} {
puts stdout ""
puts stdout "Intentar Newton with Initial Tangent .."
test NormDispIncr   1.e-006 1000 0
algorithm Newton -initial
set ok [analyze 1]
      if {$ok == 0} {puts stdout "***** Initial Newton funciono .. regreso a Newton"}
#eval "algorithm Newton"
puts stdout ""
}
if {$ok != 0} {
puts stdout ""
puts stdout "Intentar Broyden .."
algorithm Broyden 8
set ok [analyze 1 ]
if {$ok == 0} {puts stdout "***** Broyden funciono .. regreso a Newton"}
```

```
#eval "algorithm Newton"
puts stdout ""
}
if {$ok != 0} {
puts stdout ""
puts stdout "Intentar NewtonWithLineSearch .."
algorithm NewtonLineSearch -type InitialInterpolation -tol 0.8 -maxIter 100
set ok [analyze 1]
if {$ok == 0} {puts stdout "***** NewtonWithLineSearch funciono .. regreso a Newton"}
#eval "algorithm Newton"
puts stdout ""
}

# Check analysis progress

set Vcurr [getTime]  ;  # Retrieve base shear
}

if {$ok != 0} { puts stderr "El analisis fallo en converger al desplazamiento:

$controlDisp en el nodo $NodeControl con cortante basal $Vcurr"}
else { puts stdout "Pushover Analysis completado exitosamente"}  ;  # end if

}

wipe all;
```