APPENDIX F

MATLAB PROGRAM LISTING

This appendix provides a listing of seven MATLAB program components for this study.

(1) Generation of data structure of the steel SMRF and relevant information

create_SMRF; s_modify_2_5s4b_splice; compute_general_data; compute_bldg_weight; modify_s_Live; modify_s_DWE.

(2) Checking code-compliance of an SMRF design based on elastic analysis

LRFD_analysis; Kfactor; alignchart; alignchart2; LRFDcheck; LRFD_combo; column_comp_strength; B2_factor.

(3) Static pushover analysis of a valid SMRF design using DRAIN-2DX

create_DRAIN_INP_mode; get_DRAIN_mode; create_DRAIN_INP_pushover; get_DRAIN_pushover; fit_pushover_BILINEAR; BILINEAR; equivalent_SDOF; get_Sa_SAC; find_DriftRatioProfile; LNfit; LN.

(4) Nonlinear time history analysis using DRAIN-2DX

main_NTHA; read_EQdata; create_DRAIN_INP_NTHA; get_response_history.

(5) Quantification of merit objective functions of a valid SMRF design

compute_ SeismicDamageCost; compute_InitialCost.

(6) Implementation of multiobjective genetic algorithm

generate_initial_population; reproduce; cross_mutate; NSGAII.

(7) Main programs for multiobjective design optimization of SMRF

main_multiobj; objective_evaluation.

F.1 Generation of data structure of the steel SMRF and relevant information

```
function s=create_SMRF()

% generate the basic data structure of a steel SMRF structure used in this study

%----------------------------------- BASIC DATA INPUT -----------------------------------

s.num_stories=5;                                    % number of stories
s.num_bays=4;                                       % number of bays %
s.story_height=[15;13*ones(s.num_stories-1,1)]*12;  % story height [in]
s.bay_width=25*12;                                  % bay width [in]

s.yieldB=36*1000;     % nominal yield strength for beams   [psi]
s.yieldC=50*1000;     % nominal yield strength for columns [psi]

s.mod=29000000;       % steel modulus of elasticity [psi]

% define dimension info along the orthogonal direction
s.bay_widthY=s.bay_width;
s.num_baysY=6;
s.num_baysYframe=s.num_bays;

%-------------------------------- END OF BASIC DATA INPUT --------------------------------

s.trib=s.bay_width/2;                               % tributary width
s.nn=(s.num_stories+1)*(s.num_bays+1);              % number of nodes
s.nel=(2*s.num_bays+1)*s.num_stories;               % number of elements
s.numcol=(s.num_bays+1)*s.num_stories;              % number of columns
s.mps=ones(1,s.nel);                                % initialize material property

%-----------------------------------------------------------------------------------------
%                 compute nodal coordinates
%                 s.x is vector of x coordinates of each node (inches)
%                 s.y is vector of y coordinates of each node (inches)
%                 (node order is left to right by rows, bottom to top)
%-----------------------------------------------------------------------------------------

for i=0:s.num_stories
    if i==0
        for j=0:s.num_bays
            s.x(j+1)=j*s.bay_width;
            s.y(j+1)=0;
        end
    else
        for j=0:s.num_bays
            s.x(i*(s.num_bays+1)+j+1)=j*s.bay_width;
            s.y(i*(s.num_bays+1)+j+1)=sum(s.story_height(1:i));
        end
    end
end

%-----------------------------------------------------------------------------------------
%                 define i and j ends of columns
%                 s.i is vector of i-end node numbers for each column
%                 s.j is vector of j-end node numbers for each column
%                 (element order is left to right by rows, bottom to top)
%-----------------------------------------------------------------------------------------

for i=1:s.num_stories
   for j=0:s.num_bays
      s.i((i-1)*(s.num_bays+1)+j+1)=(i-1)*(s.num_bays+1)+j+1;
      s.j((i-1)*(s.num_bays+1)+j+1)=i*(s.num_bays+1)+j+1;
   end
end

%-----------------------------------------------------------------------------------------
%                 define i and j ends of beams
%                 s.i is vector of i-end node numbers for each beam
%                 s.j is vector of j-end node numbers for each beam
%                 (element order is left to right by rows, bottom to top,
%                 after all columns have been numbered)
%-----------------------------------------------------------------------------------------

for i=1:s.num_stories
   for j=1:s.num_bays
      s.i(s.numcol+(i-1)*(s.num_bays)+j)=i*(s.num_bays+1)+j;
```

```
      s.j(s.numcol+(i-1)*(s.num_bays)+j)=i*(s.num_bays+1)+j+1;
   end
end

%-------------------------------------------------------------------------------------------
%                 define design variables, assuming left-right symmetry;
%                 order is left to right by rows, bottom to top,
%                 for columns first, then for beams.
%-------------------------------------------------------------------------------------------

dv=1;
for i=1:s.num_stories                                  % assign DV's to columns
   for j=1:fix(s.num_bays/2)+1;
      s.sps((i-1)*(s.num_bays+1)+j)=dv;
      s.sps((i-1)*(s.num_bays+1)+s.num_bays-j+2)=dv;
      dv=dv+1;
   end
end
ncol_dv = dv-1;                                        % number of design variables for columns only
s.col_dv_per_floor = ncol_dv/s.num_stories;            % # of design variables per floor
s.num_col_dv = ncol_dv;
for i=1:s.num_stories                                  % assign DV's to beams, floor linking is considered
      s.sps(s.numcol+(i-1)*s.num_bays+(1:s.num_bays))=dv;
      dv=dv+1;
end
s.numdv=dv-1;                                          % total number of design variables

%-------------------------------------------------------------------------------------------
%                 section property type for each section property set:
%                 col=1, beam=2
%                 access using s.sptype(s.sps(elem#))
%-------------------------------------------------------------------------------------------

s.sptype=[ones(1,ncol_dv),2*ones(1,s.numdv-ncol_dv)];

%-------------------------------------------------------------------------------------------
%                 initialize pointers into the W-shapes column table
%                 (s.sptype=1) or W-shapes beam table (s.sptype=2).
%                 access using s.spdv(s.sps(elem#))
%-------------------------------------------------------------------------------------------

s.spdv=[70*ones(1,ncol_dv),120*ones(1,s.numdv-ncol_dv)];

%-------------------------------------------------------------------------------------------
%                 lists of members with common design variables.
%                 spsml(dv#) is the # of elements with design variable dv#.
%                 spsm(dv#,j) lists (by rows) the member numbers with dv#.
%                 access using elem#=s.spsm(dv#,i).
%-------------------------------------------------------------------------------------------

s.spsml=zeros(s.numdv,1);
for i=1:s.nel
   dv=s.sps(i);
   s.spsml(dv)=s.spsml(dv)+1;
   s.spsm(dv,s.spsml(dv))=i;
end

%-------------------------------------------------------------------------------------------
%                 create bounded dof array.
%                 s.nb is the total # of bounded (restrained) DOF.
%                 s.b contains node and directions (1=u,2=v,3=theta) of bounded DOF.
%                 access using node#=s.b(i,1), dir=s.b(i,2).
%-------------------------------------------------------------------------------------------

nb1=s.num_bays+1;
s.nb=3*nb1;
s.b=[1:nb1,1:nb1,1:nb1; ones(1,nb1),2*ones(1,nb1),3*ones(1,nb1)]';

%-------------------------------------------------------------------------------------------
%                 element lengths and direction cosines.
%                 access using length=s.l(elem#).
%-------------------------------------------------------------------------------------------

s.l=[];
for istory=1:s.num_stories
    s.l=[s.l,s.story_height(istory)*ones(1,s.num_bays+1)];
end
s.l=[s.l,s.bay_width*ones(1,s.nel-s.numcol)];
s.lx=[zeros(1,s.numcol),ones(1,s.nel-s.numcol)];
s.ly=[ones(1,s.numcol),zeros(1,s.nel-s.numcol)];
```

```
%-------------------------------------------------------------------------------------------
%                 degree of freedom number assignment for each node (row).
%                 access using dof#=s.dof(node#,1-3), where 1-3 selects u,v,theta.
%                 (numbered from left to right by rows, TOP to bottom
%                 bounded DOF all come last, but this may change in future versions.)
%-------------------------------------------------------------------------------------------

s.dof=zeros(s.nn,3);
currentdof=3*s.nn;
for i=1:s.nb
   s.dof(s.b(i,1),s.b(i,2))=currentdof;
   currentdof=currentdof-1;
end
for i=1:s.nn
   for j=1:3
      if s.dof(i,j) == 0
         s.dof(i,j)=currentdof;
         currentdof=currentdof-1;
      end
   end
end
s.ndof=3*s.nn-s.nb;   % total number of unbounded (unrestrained) DOF

%-------------------------------------------------------------------------------------------
%                 destination array for each element (row).
%                 access using dof#=s.da(elem#,1-6), where 1-6 selects
%                 from ui,vi,thetai,uj,vj,thetaj on the element i/j ends.
%-------------------------------------------------------------------------------------------

s.da=[s.dof(s.i,:),s.dof(s.j,:)];

%-------------------------------------------------------------------------------------------
%                 element numbers of columns incident to each node
%                 for use with effective length calculations.
%                 access using elem#=s.colinc(node#,1-2). If only one
%                 column is attached, the second entry is zero.
%-------------------------------------------------------------------------------------------

s.colinc=zeros(s.nn,2);
for i=1:s.nel
   if s.sptype(s.sps(i))==1
      iend=s.i(i);
      if s.colinc(iend,1)==0
         s.colinc(iend,1)=i;
      else
         s.colinc(iend,2)=i;
      end
      jend=s.j(i);
      if s.colinc(jend,1)==0
         s.colinc(jend,1)=i;
      else
         s.colinc(jend,2)=i;
      end
   end
end

%-------------------------------------------------------------------------------------------
%                 element numbers of beams incident to each node
%                 for use with effective length calculations.
%                 access using elem#=s.colinc(node#,1-2). If fewer than
%                 two beams are attached, the remaining entries are zero.
%-------------------------------------------------------------------------------------------

s.beaminc=zeros(s.nn,2);
for i=1:s.nel
   if s.sptype(s.sps(i))==2
      iend=s.i(i);
      if s.beaminc(iend,1)==0
         s.beaminc(iend,1)=i;
      else
         s.beaminc(iend,2)=i;
      end
      jend=s.j(i);
      if s.beaminc(jend,1)==0
         s.beaminc(jend,1)=i;
      else
         s.beaminc(jend,2)=i;
      end
   end
end
```

```
%-------------------------------------------------------------------------------------------
%                 build pre-stiffness matrix for each element, so that
%                 element stiffness is s.kt*diag(A,I,I,A,I,I).
%-------------------------------------------------------------------------------------------

for i=1:s.nel
   e=s.mod(s.mps(i));
   l=s.l(i);
   s.kt(:,:,i)=zeros(6);
   s.kt(1,1,i)=e/l;
   s.kt(1,4,i)=-e/l;
   s.kt(2,2,i)=12*e/l^3;
   s.kt(2,3,i)=6*e/l^2;
   s.kt(2,5,i)=-12*e/l^3;
   s.kt(2,6,i)=6*e/l^2;
   s.kt(3,2,i)=6*e/l^2;
   s.kt(3,3,i)=4*e/l;
   s.kt(3,5,i)=-6*e/l^2;
   s.kt(3,6,i)=2*e/l;
   s.kt(4,1,i)=-e/l;
   s.kt(4,4,i)=e/l;
   s.kt(5,2,i)=-12*e/l^3;
   s.kt(5,3,i)=-6*e/l^2;
   s.kt(5,5,i)=12*e/l^3;
   s.kt(5,6,i)=-6*e/l^2;
   s.kt(6,2,i)=6*e/l^2;
   s.kt(6,3,i)=2*e/l;
   s.kt(6,5,i)=-6*e/l^2;
   s.kt(6,6,i)=4*e/l;

   lx=s.lx(i); %build transformation matrix for element i.
   ly=s.ly(i);
   s.t(:,:,i)=zeros(6);
   s.t(1,1,i)=lx;
   s.t(1,2,i)=ly;
   s.t(2,1,i)=-ly;
   s.t(2,2,i)=lx;
   s.t(3,3,i)=1;
   s.t(4,4,i)=lx;
   s.t(4,5,i)=ly;
   s.t(5,4,i)=-ly;
   s.t(5,5,i)=lx;
   s.t(6,6,i)=1;
end
```

function s=s_modify_2_5s4b_splice

```
s.sps = [    1     2     3     2     1  ...
             1     2     3     2     1  ...
             4     5     6     5     4  ...
             4     5     6     5     4  ...
             4     5     6     5     4  ...
             7     7     7     7 ...
             8     8     8     8 ...
             9     9     9     9 ...
            10    10    10    10 ...
            11    11    11    11];

s.num_col_dv = 6;
s.col_dv_per_floor = 3;
s.numdv = 11;
s.sptype = [ 1     1     1     1     1     1     2     2     2     2     2];
s.spdv=[70*ones(1,s.num_col_dv),120*ones(1,s.numdv-s.num_col_dv)];
s.spsml = [  4     4     2     6     6     3     4     4     4     4     4];
```

function s=compute_general_data(s,bw,cw)

```
% calculate general-purpose data for all future designs.
% also define 's.L' (the live load condition)


%-------------------------------------------------------------------------------------------
%                 basic data
%-------------------------------------------------------------------------------------------

s.dead_intensity=[76/144*ones(s.num_stories-1,1);67/144]; %UNIT: lbs/in/in
```

```
s.W_extWall_facade_intensity=[30/144*ones(s.num_stories-1,1);30/144];

s.live_intensity=[45/144*ones(s.num_stories-1,1);16/144];

% from top to bottom, '39'=W18X35, '46'=W18X40
Gracity_beam_dv=[39  46  46  46  46  46  46  46  46];

% from top to bottom, '21'=W14X48, '39'=W14X82, '47'=W14X109
Gracity_col_dv =[21  21  21  21  21  39  39  47  47];

lengthX=s.bay_width*s.num_bays;       % building dimension along SMRF direction
lengthY=s.bay_width*s.num_baysY;      % building dimension orthogonal to SMRF direction

plan_area=lengthX*lengthY;            % [in^2]
plan_perimeter=2*(lengthX+lengthY);   % [in]

%--------------------------------------------------------------------------------------------
%                 initialize arrays
%--------------------------------------------------------------------------------------------

s.DEADweight = zeros(s.num_stories,1);
s.LIVEweight = zeros(s.num_stories,1);
s.GravityFrame_weight=zeros(s.num_stories,1);

%--------------------------------------------------------------------------------------------
%                 calculate external dead load
%--------------------------------------------------------------------------------------------

s.DEADweight= plan_area*s.dead_intensity...
            +plan_perimeter*s.story_height.*s.W_extWall_facade_intensity;

%--------------------------------------------------------------------------------------------
%                 calculate live load
%--------------------------------------------------------------------------------------------

s.L= -s.trib*s.live_intensity;  % 'negative = downward'

LLfactor=ones(s.num_stories,1); % calculate live load reduction factor, based on ASCE 7-98
Kii = 4; % for exterior columns w/o cantilever slabs; NOTE: Kii=2 for edge beams w/o c-slabs
At = (s.trib/12)*(s.num_bays*s.bay_width/12);
if Kii*At >= 400
    reduction_factor = 0.25+4.57/sqrt(Kii*At);
    LLfactor(1:s.num_stories-1)=ones(1,s.num_stories-1)*max(reduction_factor,0.4);
    LLfactor(s.num_stories)=max(reduction_factor,0.5);
end

s.L = s.L.*LLfactor;            % modify live load

s.LIVEweight=plan_area*s.live_intensity.*LLfactor;

%--------------------------------------------------------------------------------------------
%                 calculate dead load due to gravity beam/column
%--------------------------------------------------------------------------------------------

s.GravityFrame_weight=zeros(s.num_stories,1);

TotalLength_grav_beam_per_story =  (s.num_baysYframe+1)*s.num_bays*s.bay_width ...
                                 + (s.num_bays-1)*s.num_baysYframe*s.bay_widthY ...
                                 + (s.num_baysY-s.num_baysYframe)*(s.num_bays+1)*s.bay_widthY;
TotalLength_grav_col_per_story  =  (s.num_baysYframe-1)*(s.num_bays-1)*s.story_height;

tmpid=0;
for istory=s.num_stories:-1:1 % count down coz gravity members are list from top to bottom
    tmpid=tmpid+1;
    if istory==s.num_stories
        s.GravityFrame_weight(istory)=  TotalLength_grav_beam_per_story...
                                       *bw(Gracity_beam_dv(tmpid),2)/12 ...
                                       +TotalLength_grav_col_per_story(istory)...
                                       *cw(Gracity_col_dv(tmpid),2)/12/2;
    else
        s.GravityFrame_weight(istory)=  TotalLength_grav_beam_per_story...
                                       *bw(Gracity_beam_dv(tmpid),2)/12 ...
                                       +TotalLength_grav_col_per_story(istory)...
                                       *(cw(Gracity_col_dv(tmpid),2)...
                                        +cw(Gracity_col_dv(tmpid-1),2))/12/2;
    end
end

%--------------------------------------------------------------------------------------------
%                 calculate fireproofing areas of gravity beams/column
%--------------------------------------------------------------------------------------------
```

```
s.gravityFireProofArea_col=0;
s.gravityFireProofArea_beam=0;

tmpid=0;
for istory=s.num_stories:-1:1 % count down coz gravity members are list from top to bottom

    tmpid=tmpid+1;

    dv_gbeam=Gracity_beam_dv(tmpid);
    d =bw(dv_gbeam,4);       % depth [in]
    bf=bw(dv_gbeam,6);       % flange width [in]
    k1=bw(dv_gbeam,8);       % distance [in]
    s.gravityFireProofArea_beam=   s.gravityFireProofArea_beam ...
                                 + TotalLength_grav_beam_per_story*(2*d+3*bf-4*k1);

    dv_gcol=Gracity_col_dv(tmpid);
    d =cw(dv_gcol,4);
    bf=cw(dv_gcol,6);
    k1=cw(dv_gcol,8);
    s.gravityFireProofArea_col=   s.gravityFireProofArea_col ...
                                + TotalLength_grav_col_per_story(istory)*(2*d+4*bf-4*k1);

end

%-------------------------------------------------------------------------------------------
%                 prepare cost data for 7 damage states
%-------------------------------------------------------------------------------------------

central_damage_index = [0  0.5/100  5/100  20/100  45/100  80/100  100/100]';
mean_days_function_loss = [0  3.4  12.08  44.72  125.66  235.76  346.93]';
minor_injury_rate = [ 0  0.00003  0.003  0.003  0.03  0.3  0.4]';
serious_injury_rate = [ 0  0.000004  0.00004  0.0004  0.004  0.04  0.4]';
fraction_death = [0.000001  0.00001  0.0001  0.001  0.001  0.01  0.2]';

s.total_floor_area = (s.num_stories-1)*(s.num_bays*s.bay_width/12)...
                   *(s.num_baysY*s.bay_width/12); %foot_sq
gross_leasable_area = 0.7*s.total_floor_area;

cost_damage_repair = 85*s.total_floor_area*central_damage_index;

cost_lost_contents = 28.9*s.total_floor_area*central_damage_index;

cost_relocation = 1.5*gross_leasable_area*mean_days_function_loss/30; % [month]

cost_rental = 0.58*gross_leasable_area*mean_days_function_loss/30;    % [month]

cost_income = 100*gross_leasable_area*mean_days_function_loss/365;    % [year]

cost_injury = (1000*minor_injury_rate+10000*serious_injury_rate)...
            *(s.total_floor_area/1000*2);

cost_death = (1740000*fraction_death)*(s.total_floor_area/1000*2);

s.LScost = cost_damage_repair+cost_lost_contents...
         +cost_relocation+cost_rental+cost_income+cost_injury;

%-------------------------------------------------------------------------------------------
%                 define some seismic data
%-------------------------------------------------------------------------------------------

Ss = 1.61; S1 = 1.19;  % [g],

Fa=1.0;Fv=1.0;

s.Sds = 2/3*Fa*Ss;
s.Sd1 = 2/3*Fv*S1;

s.Reduction = 8.0;           % for SRMF
s.Importance = 1.0;          % seismic use group I

Cr = 0.028;                  % FEMA-368
hn = sum(s.story_height)/12; % frame total height [ft]
s.Ta = Cr*hn^(0.8);          % empirical period, FEMA-368
s.Cu=1.4;                    % coeff. for upper limit on calculated fundamental period


function s = compute_bldg_weight(s,bw,cw)

% calculate story weight of the entire building (3-D),
```

```
% and define 's.D'(the dead load condition)
% and mass for natural period calculation in DRAIN-2DX
% also for seismic lateral load calculation of CsW (i.e., reactive seismic weight)

% input arguments--------------------------------------------------------------------------------
%                 's':  data structure of the present steel SMRF
%                 'bw': database of steel section types for beamms
%                 'cw': database of steel section types for columns
%-------------------------------------------------------------------------------------------

s.FrameFactor_Y = 1.0;   % assume that weight of each story of the EXTERIOR FRAMES along
                         % the orthogonal direction is 1.0 times that of present SMRF

s.totalWeightPerStory = zeros(s.num_stories,1); %initialize totalstory weight for one SMRF

s.SMRFweight = zeros(s.num_stories,1); % steel weight of one SMRF

%-------------------------------------------------------------------------------------------
%                 treat the first story/floor separately
%-------------------------------------------------------------------------------------------
node_start_id=(s.num_bays+1)*1+1;
for k=1:s.num_bays+1
   cw_lower_id=s.spdv(s.sps(s.colinc(node_start_id+k-1,1)));
   s.SMRFweight(1)=s.SMRFweight(1)+cw(cw_lower_id,2)/12*s.story_height(1);
   col_upper_id=s.colinc(node_start_id+k-1,2);
   cw_upper_id=s.spdv(s.sps(col_upper_id));
   s.SMRFweight(1)=s.SMRFweight(1)+cw(cw_upper_id,2)/12*s.story_height(1+1)/2;
end
beam_start_id=s.numcol+s.num_bays*(1-1)+1;
for k=1:s.num_bays
   bw_id=s.spdv(s.sps(beam_start_id+k-1));
   s.SMRFweight(1)=s.SMRFweight(1)+bw(bw_id,2)/12*s.bay_width;
end

%-------------------------------------------------------------------------------------------
%                 treat the remaining stories
%-------------------------------------------------------------------------------------------

for j=2:s.num_stories
   beam_start_id=s.numcol+s.num_bays*(j-1)+1;
   %add beam self-weight
   for k=1:s.num_bays
      bw_id=s.spdv(s.sps(beam_start_id+k-1));
      s.SMRFweight(j)=s.SMRFweight(j)+bw(bw_id,2)/12*s.bay_width;
   end
   node_start_id=(s.num_bays+1)*j+1;
   for k=1:s.num_bays+1
      cw_lower_id=s.spdv(s.sps(s.colinc(node_start_id+k-1,1)));
      s.SMRFweight(j)=s.SMRFweight(j)+cw(cw_lower_id,2)/12*s.story_height(j)/2;
      col_upper_id=s.colinc(node_start_id+k-1,2);
      if col_upper_id ~=0   %upper column exists
         cw_upper_id=s.spdv(s.sps(col_upper_id));
         s.SMRFweight(j)=s.SMRFweight(j)+cw(cw_upper_id,2)/12*s.story_height(j+1)/2;
      end
   end
end

%-------------------------------------------------------------------------------------------
%                 total weight for one SMRF
%-------------------------------------------------------------------------------------------

s.totalWeightPerStory =(2*(1+s.FrameFactor_Y)*s.SMRFweight+s.DEADweight...
                        +s.GravityFrame_weight+0.25*s.LIVEweight)/2;

%-------------------------------------------------------------------------------------------
%                 steel weight per story for the entire building
%-------------------------------------------------------------------------------------------

s.totalSteelWeight = 2*(1+s.FrameFactor_Y)*s.SMRFweight + s.GravityFrame_weight;

%-------------------------------------------------------------------------------------------
%                 define total dead load intensity for one SMRF
%                 'negative = downward'
%-------------------------------------------------------------------------------------------

s.D= -s.trib*s.dead_intensity -s.SMRFweight/s.num_bays/s.bay_width...
     -s.W_extWall_facade_intensity.*s.story_height;


function s=modify_s_Live(s)
```

```
% modify 's' generated by 'create_SMRF.m'
% compute invariant floor/roof live load intensity 's.L' [lbs/in], respectively.

% currently, the follwoing 9 BASIC loads are defined:
% (1)  D  only
% (2)  L  only (full)
% (3)  L  only (stagger I)
% (4)  L  only (stagger II)
% (5)  Lr only (full)
% (6)  Lr only (stagger I)
% (7)  Lr only (stagger II)
% (8)  W  only
% (9)  E  only

s.numBasicLoad=9;

% uniformly distributed BASIC element loads
% from D, L(full), L(stagger I), L(stagger II), Lr(full), Lr(stagger I),
%      Lr(stagger II), respectively.
% access using distr_load=s.eload(elem#,BasicLoad#).
% sign convention is positive up

s.numELoad=7;

L_staggerIarray=zeros(s.nel,1);
for istory=1:s.num_stories-1
   if mod(istory,2)==1        % odd story number
      L_staggerIarray(s.numcol+s.num_bays*(istory-1)+(1:2:s.num_bays))=1;
   else
      L_staggerIarray(s.numcol+s.num_bays*(istory-1)+(2:2:s.num_bays))=1;
   end
end

Lr_staggerIarray=zeros(s.nel,1);
if mod(s.num_stories,2)==1    % odd story number
   Lr_staggerIarray(s.numcol+s.num_bays*(s.num_stories-1)+(1:2:s.num_bays))=1;
else
   Lr_staggerIarray(s.numcol+s.num_bays*(s.num_stories-1)+(2:2:s.num_bays))=1;
end

L_staggerIIarray=zeros(s.nel,1);
for istory=1:s.num_stories-1
   if mod(istory,2)==2        % even story number
      L_staggerIIarray(s.numcol+s.num_bays*(istory-1)+(1:2:s.num_bays))=1;
   else
      L_staggerIIarray(s.numcol+s.num_bays*(istory-1)+(2:2:s.num_bays))=1;
   end
end

Lr_staggerIIarray=zeros(s.nel,1);
if mod(s.num_stories,2)==2    % even story number
   Lr_staggerIIarray(s.numcol+s.num_bays*(s.num_stories-1)+(1:2:s.num_bays))=1;
else
   Lr_staggerIIarray(s.numcol+s.num_bays*(s.num_stories-1)+(2:2:s.num_bays))=1;
end

% treat live loads first. Dead, wind, and earthquake loads will be done later
% because they are related with steel weight of currently designed frame

s.eload=zeros(s.nel,s.numELoad);

%assign floor live load
for istory=1:s.num_stories-1
    s.eload(s.numcol+s.num_bays*(istory-1)+1:s.numcol+s.num_bays*istory,2)...
        = ones(s.num_bays,1)*s.L(istory);                % L only (full)
    s.eload(s.numcol+s.num_bays*(istory-1)+1:s.numcol+s.num_bays*istory,3)...
        = L_staggerIarray(s.numcol+s.num_bays*(istory-1)+1:s.numcol...
                          +s.num_bays*istory)*s.L(istory); % L only (stagger I)
    s.eload(s.numcol+s.num_bays*(istory-1)+1:s.numcol+s.num_bays*istory,4)...
        = L_staggerIIarray(s.numcol+s.num_bays*(istory-1)+1:s.numcol...
                          +s.num_bays*istory)*s.L(istory); % L only (stagger II)
end

%assign roof live load
s.eload(s.numcol+s.num_bays*(s.num_stories-1)+1:s.nel,5)...
    = ones(s.num_bays,1)*s.L(s.num_stories);             % Lr only (full)
s.eload(s.numcol+s.num_bays*(s.num_stories-1)+1:s.nel,6)...
    = Lr_staggerIarray(s.numcol+s.num_bays*(s.num_stories-1)...
              +1:s.nel)*s.L(s.num_stories);              % Lr only (stagger I)
s.eload(s.numcol+s.num_bays*(s.num_stories-1)+1:s.nel,7)...
```

```
    = Lr_staggerIIarray(s.numcol+s.num_bays*(s.num_stories-1)....
                +1:s.nel)*s.L(s.num_stories);            % Lr only (stagger II)

%-------------------------------------------------------------------------------------------
%                 compute fixed end forces for use in stiffness method.
%                 access using force=s.fef(1-6,elem#,lc#), where
%                 1-6 selects Ni,Vi,Mi,Nj,Vj,Mj on the i/j ends of the element.
%-------------------------------------------------------------------------------------------

s.fef=zeros(6,s.nel,s.numBasicLoad);

for i=2:7        % for live floor/roof load only now
   for j=1:s.nel
      if s.sptype(s.sps(j)) == 2  % beam type
         v=-s.eload(j,i)*s.l(j)/2;
         m=-s.eload(j,i)*s.l(j)^2/12;
         s.fef(2,j,i)=v;
         s.fef(3,j,i)=m;
         s.fef(5,j,i)=v;
         s.fef(6,j,i)=-m;
      end
   end
end

%-------------------------------------------------------------------------------------------
%                 nodal load vector for stiffness equations (in order of DOF),
%                 padded with zeros at end for bounded DOF.
%                 access using load=s.nl(dof#,lc#).
%                 's.nl' contains all 's.numBasicLoad' basic loads
%-------------------------------------------------------------------------------------------

s.nl=zeros(3*s.nn,s.numBasicLoad); % 's.numBasicLoad' is defined in 's_modify_BasicLive02.m'

for i=2:7   % for live floor/roof load only now, do later for dead, wind, EQ loads
   for j=1:s.nel
      for k=1:6
         s.nl(s.da(j,k),i)=s.nl(s.da(j,k),i)-s.fef(k,j,i);
      end
   end
end



function [s,pushoverProfile]=modify_s_DWE(s,bw,cw,T1)

% modify 's' generated by 'create_SMRF.m'
% compute frame-depedent basic loads of D, W, and E

% "T1': calculated fundamental period

%-------------------------------------------------------------------------------------------
%                 earthquake load
%-------------------------------------------------------------------------------------------

T=min(T1,s.Cu*s.Ta);  % maximum allowable fundamental period

% determine 'k' factor for vertical load distributions
if T <= 0.5
   k = 1;
elseif T >= 2.5
   k = 2;
else
   k = 1+(T-0.5)/2;   % linear interpolation
end

% calculate seimic response coefficient 'Cs'
Cs = max(min(s.Sds/(s.Reduction/s.Importance),s.Sd1/T/(s.Reduction/s.Importance)),...
         0.044*s.Sds*s.Importance);
CsT1 = max(min(s.Sds/(s.Reduction/s.Importance),s.Sd1/T1/(s.Reduction/s.Importance)),...
         0.044*s.Sds*s.Importance);
Cs = Cs*1.05;         % consider effect of accidental torsion
s.drift_Cs_factor = CsT1/Cs; %use 'T1' in calculating drift ratio in 'LRFDcheck'
s.Cs = Cs;

Vb = Cs*sum(s.totalWeightPerStory); % design base shear [lbs]

%determine nodal seismic load
s.E = zeros(s.num_stories,1);
tmp = zeros(s.num_stories,1);
for istory = 1:s.num_stories
   tmp(istory) = s.totalWeightPerStory(istory)*sum(s.story_height(1:istory))^k;
```

```
end
s.E = tmp/sum(tmp)*Vb;

for jstory=1:s.num_stories
   dofnum=s.dof(jstory*(s.num_bays+1)+1,1);
   s.nl(dofnum,9)=s.E(jstory);
end

%normalized lateral force profile (sum=1) used in the DRAIN pushover analysis
pushoverProfile=tmp/sum(tmp);

%-------------------------------------------------------------------------------------------
%                dead load
%-------------------------------------------------------------------------------------------

%assign floor & roof dead load
for istory=1:s.num_stories
    s.eload(s.numcol+s.num_bays*(istory-1)+1:s.numcol+s.num_bays*istory,1)...
        = ones(s.num_bays,1)*s.D(istory);  %D only (full)
end

for j=1:s.nel
   if s.sptype(s.sps(j)) == 2     % beam type
      v=-s.eload(j,1)*s.l(j)/2;
      m=-s.eload(j,1)*s.l(j)^2/12;
      s.fef(2,j,1)=v;
      s.fef(3,j,1)=m;
      s.fef(5,j,1)=v;
      s.fef(6,j,1)=-m;
   end
end

s.nl(:,1)=zeros(3*s.nn,1);  %re-zerolize nodal force for dead load
for j=1:s.nel
   for k=1:6
      s.nl(s.da(j,k),1)=s.nl(s.da(j,k),1)-s.fef(k,j,1);
   end
end

%-------------------------------------------------------------------------------------------
%                WIND load
%-------------------------------------------------------------------------------------------

% based on ASCE-7 98 (design procedure 6.5.3)
% which uses building vibrational frequency as a factor,
% so it is frame-dependent, i.e., not a constant

% STEP 1: basic wind spped V and wind directionality factor Kd
V = 85;         % [mph], for Los Angeles
Kd = 0.85;      % for 'Buildings' type

% STEP 2: determine an 'importance factor I'
I = 1.0;        % building category II, and non-hurricane prone regions

% STEP 3: exposure category, and velocity pressure exposure coefficient Kz
alpha = 5.0;    % 'Exposure A'
zg = 1500;      % [ft]
Kz = zeros(s.num_stories,1);
for istory=1:s.num_stories
    if istory==1
        mid_height = s.story_height(istory)/2/12;
    else
        mid_height = (s.story_height(istory)+s.story_height(istory-1))/2/12;
    end
    if mid_height < 15
        Kz(istory) = 2.01*(15/zg)^(2/alpha);
    elseif mid_height < zg
        Kz(istory) = 2.01*(mid_height/zg)^(2/alpha);
    end
end
Kh = 2.01*(sum(s.story_height)/12/zg)^(2/alpha);

% STEP 4: topographic factor Kzt
Kzt = 1;        % not located over hills, ridges, or escarpments

%STEP 5: gust effect factor G
gQ = 3.4;       % for flexible or dynamically sensitive structures,
gv = 3.4;       % defined as those w/ T > 1 sec
n1 = 1/T1;      % building natural frequency;
h = sum(s.story_height)/12; % mean roof height of a building in ft
zmin = 60;      % minimal length, for Exposure A
```

```
zbar = max(0.6*h, zmin);
c = 0.45;       %turbulence intensity factor
epsilonbar = 1/2.0; % integral length scale power law exponent
l = 180;            % integral length scale factor in ft
bbar = 0.30;        % mean hourly wind speed factor
alphabar = 1/3.0;   % mean hourly wind speed power law exponent
Lzbar = l*(zbar/33)^epsilonbar; % integral length scale of turbulence
                                % at the eqivalent height
L = s.bay_width*s.num_bays/12;  % horizontal dimension of building
                                % measured parallel to wind direction [ft]

B = s.bay_widthY*s.num_baysY/12;% horizontal dimension of building
                                % measured normal to wind direction [ft]
Q = sqrt(1/(1+0.63*((B+h)/Lzbar)^0.63)); % background response
Izbar = c*(33/zbar)^(1/6); % intensity of turbulence at height zbar
Vzbar = bbar*(zbar/33)^alphabar*V*(88/60); % mean hourly wind speed at
                                           % height zbar [ft/s]
yita = 4.6*n1*h/Vzbar;
Rh = 1/yita - 1/2/yita/yita*(1-exp(-2*yita));
yita = 4.6*n1*B/Vzbar;
RB = 1/yita - 1/2/yita/yita*(1-exp(-2*yita));
yita = 4.6*n1*L/Vzbar;
RL = 1/yita - 1/2/yita/yita*(1-exp(-2*yita));
gR = sqrt(2*log(3600*n1))+0.577/sqrt(2*log(3600*n1));
N1 = n1*Lzbar/Vzbar;            % reduced frequency
Rn = 7.47*N1/(1+10.3*N1)^(5/3);
beta = 0.05;                    % damping ratio
R = sqrt(1/beta*Rn*Rh*RB*(0.53+0.47*RL));  % resonant response factor
Gf = 0.925*(1+1.7*Izbar*sqrt(gQ*gQ*Q*Q+gR*gR*R*R))/(1+1.7*gv*Izbar);

% STEP 6: enclosure classification
% classified as 'enclosed' building

%STEP 7: internal pressure coefficient GCpi, from Table 6-7
%        'GCpi' shall be used with 'qz' or 'qh'
%        Two cases shall be considered to determine the critical load requirements.
%        (1) a positive value of GCpi applied to all internal surfaces
%        (1) a negative value of GCpi applied to all internal surfaces
GCpi = 0.18;

%STEP 8: external pressure coefficient Cp
%        all use absolute vlaues. signs will be determined by inspection.
Cp_windward = 0.8;              % for windward wall, use with qz
Cp_leeward = 0.5;               % for leeward wall, use with qh
Cp_roof = 1.0*0.8;              % for roof pressures, use with qh

% STEP 9: velocity pressure qz or qh  [lb/ft^2]
qz = 0.00256*Kz*Kzt*Kd*V*V*I;
qh = 0.00256*Kh*Kzt*Kd*V*V*I;

%STEP 10: design wind load P
% wind loads acting on nodes.
% each row lists node #, direction (x,y,moment), magnitude
area_for_each_lateral_force = B/2*s.story_height/12; % [ft^2]
for istory = 1:s.num_stories    % deal with windward load
   dofnum = s.dof(istory*(s.num_bays+1)+1,1);
   s.nl(dofnum,8) = area_for_each_lateral_force(istory)...
                    *max(10,(qz(istory)*Gf*Cp_windward - qh*GCpi));
end
for istory = 1:s.num_stories    % deal with leeward load
   dofnum = s.dof((istory+1)*(s.num_bays+1),1);
   s.nl(dofnum,8) = area_for_each_lateral_force(istory)...
                    *max(10,(qh*Gf*Cp_leeward - qh*GCpi));
end
area_for_each_vertical_force = s.bay_widthY/2/12*s.bay_width/12 ...
                               *[1/2,ones(1,s.num_bays-1),1/2]; % [ft^2]
for iRoofNode = 1:s.num_bays+1  % deal with upward roof load
   dofnum = s.dof(s.num_stories*(s.num_bays+1)+iRoofNode,2);
   s.nl(dofnum,8) = area_for_each_vertical_force(iRoofNode)...
                    *max(10,(qh*Gf*Cp_roof - qh*GCpi));
end
```

F.2 Checking code-compliance of an SMRF design based on elastic analysis

```
function [r,flagR] =LRFD_analysis(s,bw,cw)

% Perform elastic structural analysis using structure defined in 's'

% 'r' contains displacements and member forces
% 'flagR' = 1 if stiffness matrix is bad-conditioned

%The basic loads are also defined in 's', as follows
%       -- s.D (1:s.num_stories) contains self-weight of the current SMRF, facade-wall,
%                                and dead load from tributary areas only
%       -- s.L (1:s.num_stories) contains live load (floor or roof) from tributary areas only
%       -- s.W (1:s.num_stories) contains lateral nodal wind load
%       -- s.E (1:s.num_stories) contains lateral nodal earthquake load

%-----------------------------------------------------------------------------------------
%                 section properties
%-----------------------------------------------------------------------------------------

dv=s.spdv(s.sps);               % design variables index for each member dv(1 x s.nel)
area=zeros(1,s.nel);
Ix=zeros(1,s.nel);
for j=1:s.nel
    if(s.sptype(s.sps(j)) == 1)  % column type
        area(j)=cw(dv(j),3);
        Ix(j)=cw(dv(j),22);
    else                         % beam type
        area(j)=bw(dv(j),3);
        Ix(j)=bw(dv(j),22);
    end
end
secprop=[area;Ix;Ix;area;Ix;Ix];

%-----------------------------------------------------------------------------------------
%                 assemble global stiffness matrix
%-----------------------------------------------------------------------------------------

r.k=zeros(3*s.nn);
for j=1:s.nel
    r.kt(:,:,j)=s.kt(:,:,j)*diag(secprop(:,j))*s.t(:,:,j);
    r.k(s.da(j,:),s.da(j,:))=r.k(s.da(j,:),s.da(j,:))+s.t(:,:,j)'*r.kt(:,:,j);
end

%-----------------------------------------------------------------------------------------
%                 solve stiffness equations
%-----------------------------------------------------------------------------------------

lastwarn('');                               % clear warning message
r.d=r.k(1:s.ndof,1:s.ndof)\s.nl(1:s.ndof,:); % solve stiffness for all BASIC LOADs
flagR=~isempty(lastwarn);                   % set flag if there was a warning

%-----------------------------------------------------------------------------------------
%                 compute member forces r.f with access using r.f(dir,elem,lc),
%                 dir = [axial(i),shear(i),moment(i),axial(j),shear(j),moment(j)]
%-----------------------------------------------------------------------------------------

r.dl=[r.d;zeros(s.nb,s.numBasicLoad)];       % pad displacements with zeros
for j=1:s.nel
    r.f(:,j,:)=r.kt(:,:,j)*r.dl(s.da(j,:),:);% member forces
end
r.f=r.f+s.fef;                              % correct forces for element fixed end forces


function [Kcol,flagG] = Kfactor(s,bw,cw)

% calculate effective length factor, K, for all columns.
% the output Kcol is of size [s.numcol by 2] for no sidesway and sidesway cases, respectively.
% flagG = 1 if error occurs when calculating G factors

Kcol=zeros(s.numcol,2);

%-----------------------------------------------------------------------------------------
%                 section properties
%-----------------------------------------------------------------------------------------

dv=s.spdv(s.sps);
```

```
Ix=zeros(1,s.nel);
for j=1:s.nel
    if(s.sptype(s.sps(j)) == 1)
        Ix(j)=cw(dv(j),22);
    else %beam type
        Ix(j)=bw(dv(j),22);
    end
end

%-------------------------------------------------------------------------------------------
%                 compute G value at each node
%-------------------------------------------------------------------------------------------

Ix_L=Ix./s.l;
G=zeros(1,s.nn);
                                % assign values of 10 where there are no beams
noBeamList=(sum(s.beaminc')'==0);
singleColumnList=s.colinc(:,2)==0;
singleBeamList=xor(s.beaminc(:,2)==0,noBeamList);
G(find(noBeamList))=10;
Sigma_Col_Ix_L=zeros(1,s.nn);
Sigma_Beam_Ix_L=zeros(1,s.nn);
Sigma_Col_Ix_L(find(singleColumnList))=Ix_L(s.colinc(find(singleColumnList),1));
Sigma_Col_Ix_L(find(~singleColumnList))=Ix_L(s.colinc(find(~singleColumnList),1))+...
    Ix_L(s.colinc(find(~singleColumnList),2));
Sigma_Beam_Ix_L(find(singleBeamList))=Ix_L(s.beaminc(find(singleBeamList),1));
Sigma_Beam_Ix_L(find(~singleBeamList & ~noBeamList))=Ix_L(s.beaminc(find(~singleBeamList & ...
    ~noBeamList),1))+Ix_L(s.beaminc(find(~singleBeamList & ~noBeamList),2));
G(find(~noBeamList))=Sigma_Col_Ix_L(find(~noBeamList))./Sigma_Beam_Ix_L(find(~noBeamList));

for j=1:s.numcol % assumes columns come first in element list
    Ga=G(s.i(j));
    Gb=G(s.j(j));
    flagG=1;
    if (isnan(Ga)|isinf(Ga)|isnan(Gb)|isinf(Gb))
        disp(sprintf('G out of bounds: col #%d, Ga= %f, Gb=%f',j,Ga,Gb))
        return
    end

    %non-sidesway case

    f1=alignchart2(0.5,Ga,Gb);
    f2=alignchart2(0.9999,Ga,Gb);
    if sign(f1)==sign(f2)
        disp(sprintf('G''s same sign: col #%d, Ga= %f, Gb=%f',j,Ga,Gb))
        disp(sprintf('K=0.5 gives f=%f, K=0.9999 gives f=%f',f1,f2))
        return
    end
    [Kcol(j,1),fv,res]=fzero('alignchart2',[.5,0.9999],optimset('Display','none'),Ga,Gb);
    if res<0
        disp('error return from fzero finding K');
        return;
    end

    %sidesway case

    f1=alignchart(0.9999,Ga,Gb);
    f2=alignchart(20,Ga,Gb);
    if sign(f1)==sign(f2)
        disp(sprintf('G''s same sign: col #%d, Ga= %f, Gb=%f',j,Ga,Gb))
        disp(sprintf('K=0.9999 gives f=%f, K=20 gives f=%f',f1,f2))
        return
    end
    [Kcol(j,2),fv,res]=fzero('alignchart',[0.9999,20],optimset('Display','none'),Ga,Gb);
    if res<0
        disp('error return from fzero finding K');
        return;
    end
    flagG=0;
end
```

function f=alignchart(K,Ga,Gb)

```
% alignment chart equation for calculating K in sidesway case
% f is returned as zero if K, Ga, and Gb solve the alignment equation.
% K is the effective length factor for the sidesway case.

f=(Ga*Gb*(pi/K)^2-36)*sin(pi/K)-(6*pi/K)*(Ga+Gb)*cos(pi/K);
```

```
function f2=alignchart2(K,Ga,Gb)

% alignment chart equation for calculating K
% f2 is returned as zero if K, Ga, and Gb solve the alignment equation
% K is the effective length factor for the restrained case (i.e sidesway inhibited).

f2=((Ga*Gb)/4)*(pi/K)^2+((Ga+Gb)/2)*(1-((pi/K)/tan(pi/K)))+2*tan(pi/(2*K))/(pi/K)-1;



function [flagBehavior,constr_violation]=LRFDcheck(s,r,bw,cw,Kcol,relaxed_constr_limits)

% check the current design according to AISC-LRFD/NEHRP-368 Seismic Provisions
% the structural description, s, and the analysis, r.
% 'relaxed_constr_limits' contains acceptable upper limits for each normalized 'interqn ratio'
% of an infeasible designs.

% --inteqnSCWB of size (1:#joints except for roof, 1:s.nlcNoSigma)
%              'strong column weak beam check'
% --inteqnELMT of size (1:#elem, 1:s.nlc).
%              'member-wise strength check'
% --inteqnLambdaBEAM of size (1:#beams,s.nlc+1)
%              'seismic beam width-thickness ratio check'
%              first s.nlc columns for 'h/tw', the last column for 'b/t'
% --inteqnLambdaCOL of size (1:#columns,1:2*s.nlcNoSigma)
%              'seismic column width-thickness ratio check'
%              first s.nlcNoSigma columns for 'h/tw', next s.nlcNoSigma columns for 'b/t'
% --inteqnDR of size (1:#stories, 1:s.nlcHor)
%              'inter-story drift ratio check', only when lateral forces are present
% --inteqnStability (a scalar quantity)
%              'story stability factor ratio check'

% The satisfactory values for all ratios are <= 1.
% quit if any violation is detected, and set flagBehavior = 1.

constr_violation = zeros(1,6); %contains actual NORMALIZED contraint-violating values for all 6 inteqn's.

flagBehavior=0;
% flagBehavior=1 means "infeasible design, and constraint violation exceeds pre-defined 'relaxed_constr_limits'
% flagBehavior=2 means "infeasible design, but constraint violation DOES NOT exceed
% pre-defined 'relaxed_constr_limits'


%-------------------------------------------------------------------------------------------
%                generate load combination cases: both primary and secondary
%-------------------------------------------------------------------------------------------

[ComboPrimary,ComboB1,ComboB2,linkB1B2,eloadPrimary,s] = LRFD_combo(s,r);

%-------------------------------------------------------------------------------------------
%                initialize matrices
%-------------------------------------------------------------------------------------------

njoints = (s.num_stories-1)*(s.num_bays+1);  % #joints except for roof
inteqnSCWB = zeros(njoints,s.nlcNoSigma);

inteqnELMT = zeros(s.nel,s.nlc);

inteqnLambdaBEAM = zeros(s.nel-s.numcol,s.nlcNoSigma+1);

inteqnLambdaCOL  = zeros(s.numcol,2*s.nlcNoSigma);

inteqnDR = zeros(s.num_stories,1);

inteqnStability = 0;

%-------------------------------------------------------------------------------------------
%                section properties
%-------------------------------------------------------------------------------------------

dv=s.spdv(s.sps);
area=zeros(1,s.nel);
Ix=zeros(1,s.nel);
for j=1:s.nel
    if(s.sptype(s.sps(j)) == 1)
        area(j)=cw(dv(j),3);          % area
        Ix(j)=cw(dv(j),22);           % moment of inertia
        Zx(j)=cw(dv(j),28);           % plastic section modulus
        rx(j)=cw(dv(j),24);           % radius of gyration
```

```
    else      % beam type
        area(j)=bw(dv(j),3);
        Ix(j)=bw(dv(j),22);
        Zx(j)=bw(dv(j),28);
    end
end

%-------------------------------------------------------------------------------------------
%                calculate design compression strength
%-------------------------------------------------------------------------------------------

PnColComp = column_comp_strength(s,cw,Kcol);

currentCombo=zeros(6,s.nel);
currentEload=zeros(s.nel);

%-------------------------------------------------------------------------------------------
%                calculate the ratio of 'strong-column-weak-beam' requirement
%-------------------------------------------------------------------------------------------

% results saved in 'inteqnSCWB' of size (1:#joints except for roof, 1:s.nlcNoSigma).

E=s.mod;
Fyc=s.yieldC;
Fyb=s.yieldB;

for ijoint = 1:njoints

      %compute sum of plastic moment demands in beams at joint i

      nodeID=s.num_bays+1+ijoint;

      firstBeamID = s.beaminc(nodeID,1);
      firstBeamDV = s.spdv(s.sps(firstBeamID));
      Fyb = s.yieldB;
      Zbm = bw(firstBeamDV,28); %plastic section modulus
      Ry = 1.5; %for A36
      sumMpb = 1.1*Ry*Zbm*Fyb;

      secondBeamID = s.beaminc(nodeID,2);
      if secondBeamID ~= 0  %if exists
         secondBeamDV = s.spdv(s.sps(secondBeamID));
         Fyb = s.yieldB;
         Zbm = bw(secondBeamDV,28);
         Ry = 1.5;
         sumMpb = sumMpb + 1.1*Ry*Zbm*Fyb;
      end

    for ilc = 1:s.nlcNoSigma

      %compute sum of plastic moment capacities in columns above and below joint i

      lowerColID = s.colinc(nodeID,1);
      lowerColDV = s.spdv(s.sps(lowerColID));
      Zc = cw(lowerColDV,28);
      Fyc = s.yieldC;
      Puc = ComboPrimary(1,lowerColID,ilc); %required column compressive strength
      Ag = cw(lowerColDV,3);
      sumMpc = Zc*(Fyc-Puc/Ag);

      upperColID = s.colinc(nodeID,2);
      upperColDV = s.spdv(s.sps(upperColID));
      Zc = cw(upperColDV,28);
      Fyc = s.yieldC;
      Puc = ComboPrimary(1,upperColID,ilc);
      Ag = cw(upperColDV,3);
      sumMpc = sumMpc + Zc*(Fyc-Puc/Ag);

      inteqnSCWB(ijoint,ilc) = sumMpb/sumMpc;

   end
end

maxSCWB = max(max(inteqnSCWB));
if maxSCWB>relaxed_constr_limits(1)
    flagBehavior=1;
    return;
elseif maxSCWB>1.0
    constr_violation(1) = maxSCWB-1;
end
```

```
%-------------------------------------------------------------------------------------------
%                checking strength for each column and beam using LRFD criteria
%-------------------------------------------------------------------------------------------

%Deal with BEAMS first, coz the same beam formula is for ALL load combos and need no B1/B2 factors

for iLoadCase=1:s.nlcNoSigma
    currentCombo=ComboPrimary(:,:,iLoadCase);
    currentEload=eloadPrimary(:,iLoadCase);
    for ibeam=s.numcol+1:s.nel % deal with beams only
        phiMn=0.90*Zx(ibeam)*Fyb;
        Mu=max([abs(currentCombo(3,ibeam)), abs(currentCombo(6,ibeam))]);
        load=currentEload(ibeam);                                          % get distr load magn
        if load~=0
           loc_x=-currentCombo(2,ibeam)/load;                      % where zero shear force occurs
           if loc_x > 0 & loc_x < s.l(ibeam)                    % is it inside of beam?
              Mint=-currentCombo(3,ibeam)+currentCombo(2,ibeam)*loc_x/2;
              Mu=max([Mu,abs(Mint)]);
           end
        end
        Pu=currentCombo(1,ibeam);
        if Pu>0 % current beam in compression
             phiPn=0.85*area(ibeam)*Fyb;               % for beams, no buckling is considered
             Py = area(ibeam)*Fyb;                     % axial yield strength of beam
             if Pu/(0.9*Py)<=0.125
                inteqnLambdaBEAM(ibeam-s.numcol,ilc)...
                    = bw(dv(ibeam),11)/(520/sqrt(Fyb/1000)*(1-1.54*Pu/0.9/Py));
             else
                inteqnLambdaBEAM(ibeam-s.numcol,ilc)...
                    = bw(dv(ibeam),11)/max(191/sqrt(Fyb/1000)*(2.33-Pu/0.9/Py), 253/sqrt(Fyb/1000));
             end
        else    % current beam in tension
             phiPn=0.90*area(ibeam)*Fyb;
             inteqnLambdaBEAM(ibeam-s.numcol,ilc) = 0; %no need to check for tension
        end
        AxialRatio=abs(Pu/phiPn);
        BendingRatio=abs(Mu/phiMn);
        if AxialRatio>=0.2
            inteqnELMT(ibeam,iLoadCase)=AxialRatio+BendingRatio*8/9;
        else
            inteqnELMT(ibeam,iLoadCase)=AxialRatio/2+BendingRatio;
        end
        inteqnLambdaBEAM(ibeam-s.numcol,s.nlcNoSigma+1)...
            = bw(dv(ibeam),9)/(52/sqrt(Fyb/1000)); % ratio related to b/t of beam
                                                   % b = bf/2 LRFD 6-36
    end  % end of loop over all beams
end      % end of loop over all primary load combinations (for beams only)
maxBEAM = max(max(inteqnLambdaBEAM));
if maxBEAM>relaxed_constr_limits(3)
    flagBehavior=1;
    return;
elseif maxBEAM>1.0
    constr_violation(3) = maxBEAM-1;
end
if max(max(inteqnELMT))>relaxed_constr_limits(2)
    flagBehavior=1;
    return;
end

%Now deal with COLUMNS, distinguish load combinations w/ or w/o lateral loads

for iLoadCase=1:s.nlcVert    % primary load combinations w/ vertical loads only
    currentCombo=ComboPrimary(:,:,iLoadCase);
    for icolumn=1:s.numcol
        Pu=currentCombo(1,icolumn);
        if Pu>0 % current column in compression
             phiPn=0.85*PnColComp(icolumn,1);  % no sidesway case
        else    % current column in tension
             phiPn=0.90*area(icolumn)*Fyc;
        end
        phiMn=0.90*Zx(icolumn)*Fyc;
        Mu=max([abs(currentCombo(3,icolumn)), abs(currentCombo(6,icolumn))]);
        AxialRatio=abs(Pu/phiPn);
        BendingRatio=abs(Mu/phiMn);
        if AxialRatio>=0.2
            inteqnELMT(icolumn,iLoadCase)=AxialRatio+BendingRatio*8/9;
        else
            inteqnELMT(icolumn,iLoadCase)=AxialRatio/2+BendingRatio;
        end
    end  % end of loop over all columns
end  % end of loop over all primary load combinations w/ vertical loads (for columns only)
```

```
if max(max(inteqnELMT))>relaxed_constr_limits(2)
    flagBehavior=1;
    return;
end

for iLoadCase=s.nlcVert+1:s.nlc     % for primary load cases 8~18 which contain BOTH
                                    % vertical and horizontal basic loads i.e., B1/B2 are needed
    currentCombo=ComboPrimary(:,:,iLoadCase);
    currentVerticalSecondary=ComboB1(:,:,iLoadCase-s.nlcVert); % vertical load combos for B1 factor
    currentHorID=linkB1B2(iLoadCase-s.nlcVert);
    currentHorizontalSecondary=ComboB2(:,:,currentHorID);      % horizontal load combos for B2 factor
    B2 = B2_factor(s,cw,Kcol,currentCombo); % B2 is a vector of s.num_stories by 1, for CURRENT combo
    for istory=1:s.num_stories
       for j=1:s.num_bays+1                % loop over all columns within current story
          colID=(istory-1)*(s.num_bays+1)+j;
          dv_pointer=s.spdv(s.sps(colID));
          area=cw(dv_pointer,3);
          rx=cw(dv_pointer,24);
          E=s.mod(s.mps(colID));
          Fyc=s.yieldC(s.mps(colID));
          Ix=cw(dv_pointer,22);
          Zx=cw(dv_pointer,28);
          phiMn=0.90*Zx*Fyc;
          tmp1=currentVerticalSecondary(3,colID);
          tmp2=currentVerticalSecondary(6,colID);
          M2=max(abs(tmp1),abs(tmp2)); % larger end bendng moment
          M1=min(abs(tmp1),abs(tmp2)); % smaller end bendng moment
          Mnt=M2;
          if M2==0.0
              Cm=0.6;
          elseif sign(tmp1)==sign(tmp2)
              Cm=0.6-0.4*(M1/M2);
          else
              Cm=0.6+0.4*(M1/M2);
          end
          Pu=abs(currentVerticalSecondary(1,colID));  % these vertical loads produce ONLY compression.
          Lambdac=Kcol(colID,1)*s.l(colID)/rx/pi*sqrt(Fyc/E);
          Pe1=area*Fyc/Lambdac/Lambdac; % LRFD pg 6-41
          B1=max(Cm/(1-Pu/Pe1),1);      % C1-2, must be greater than 1
          Mlt=max(abs(currentHorizontalSecondary(3,colID)),abs(currentHorizontalSecondary(6,colID)));
          Mu=B1*Mnt+B2(istory)*Mlt;
          Pu=currentCombo(1,colID);
          if Pu>0 % current column in compression
               phiPn=0.85*PnColComp(colID,2);  % 6-47 LRFD, section E2, using unbraced K factor
          else    % current column in tension
               phiPn=0.90*area*Fyc;
          end
          AxialRatio=abs(Pu/phiPn);
          BendingRatio=abs(Mu/phiMn);
          if AxialRatio>=0.2
              inteqnELMT(colID,iLoadCase)=AxialRatio+BendingRatio*8/9;
          else
              inteqnELMT(colID,iLoadCase)=AxialRatio/2+BendingRatio;
          end
       end  % end of loop over all columns in current story
    end  % end of loop over all stories for current load combination
end  % end of loop over all primary load combinations w/ horizontal loads (for columns only)

maxELMT = max(max(inteqnELMT));
if maxELMT>relaxed_constr_limits(2)
    flagBehavior=1;
    return;
elseif maxELMT>1.0
    constr_violation(2) = maxELMT-1;
end

%-------------------------------------------------------------------------------------------
%                calculate 'seismic width-thickness ratio' for columns
%-------------------------------------------------------------------------------------------

% inteqnLambdaCOL of size (1:#columns,1:2*s.nlcNoSigma)
% first 's.nlcNoSigma' data columns for 'h/tw', next 's.nlcNoSigma' data columns for 'b/t'

for ilc = 1:s.nlcNoSigma
   currentCombo=ComboPrimary(:,:,ilc);
   for i = 1:njoints
      nodeID = (s.num_bays+1)+i;
      lowerColID = s.colinc(nodeID,1);
      lowerColDV = s.spdv(s.sps(lowerColID));
      if inteqnSCWB(i,ilc)<1/2.0
         inteqnLambdaCOL(lowerColID,s.nlcNoSigma+ilc) = ...
```

```
            max(inteqnLambdaCOL(lowerColID,s.nlcNoSigma+ilc), cw(lowerColDV,9)/(65/sqrt(Fyc/1000)));
         Pu = currentCombo(1,lowerColID);        % required axial strength
         if Pu<0 % column in tension
            inteqnLambdaCOL(lowerColID,ilc) = 0; % no need to check for tension
         else    % column in compression
            Py = cw(lowerColDV,3)*Fyc;           % axial yield strength of column
            if Pu/(0.9*Py)<=0.125
               inteqnLambdaCOL(lowerColID,ilc) = ...
                  max(inteqnLambdaCOL(lowerColID,ilc), ...
                      cw(lowerColDV,11)/(640/sqrt(Fyc/1000)*(1-2.75*Pu/0.9/Py)));
            else
               inteqnLambdaCOL(lowerColID,ilc) = ...
                  max(inteqnLambdaCOL(lowerColID,ilc), cw(lowerColDV,11)...
                                /max(191/sqrt(Fyc/1000)*(2.33-Pu/0.9/Py), 253/sqrt(Fyc/1000)));
            end
         end
      else % >1/2
         inteqnLambdaCOL(lowerColID,s.nlcNoSigma+ilc) = ...
            max(inteqnLambdaCOL(lowerColID,s.nlcNoSigma+ilc), cw(lowerColDV,9)/(52/sqrt(Fyc/1000)));
         Pu = currentCombo(1,lowerColID);
         if Pu<0
            inteqnLambdaCOL(lowerColID,ilc) = 0;
         else
            Py = cw(lowerColDV,3)*Fyc;
            if Pu/(0.9*Py)<=0.125
               inteqnLambdaCOL(lowerColID,ilc) = ...
                  max(inteqnLambdaCOL(lowerColID,ilc), ...
                      cw(lowerColDV,11)/(520/sqrt(Fyc/1000)*(1-1.54*Pu/0.9/Py)));
            else
               inteqnLambdaCOL(lowerColID,ilc) = ...
                  max(inteqnLambdaCOL(lowerColID,ilc), cw(lowerColDV,11)...
                                /min(191/sqrt(Fyc/1000)*(2.33-Pu/0.9/Py), 253/sqrt(Fyc/1000)));
            end
         end
      end

      upperColID = s.colinc(nodeID,2);
      if upperColID ~= 0
         upperColDV = s.spdv(s.sps(upperColID));
         if inteqnSCWB(i,ilc)<1/2.0
            inteqnLambdaCOL(upperColID,s.nlcNoSigma+ilc) = ...
               max(inteqnLambdaCOL(upperColID,s.nlcNoSigma+ilc), cw(upperColDV,9)/(65/sqrt(Fyc/1000)));
            Pu = currentCombo(1,upperColID);
            if Pu<0
               inteqnLambdaCOL(upperColID,ilc) = 0;
            else
               Py = cw(upperColDV,3)*Fyc;
               if Pu/(0.9*Py)<=0.125
                  inteqnLambdaCOL(upperColID,ilc) = ...
                     max(inteqnLambdaCOL(upperColID,ilc), ...
                         cw(upperColDV,11)/(640/sqrt(Fyc/1000)*(1-2.75*Pu/0.9/Py)));
               else
                  inteqnLambdaCOL(upperColID,ilc) = ...
                     max(inteqnLambdaCOL(upperColID,ilc), cw(upperColDV,11)...
                                   /max(191/sqrt(Fyc/1000)*(2.33-Pu/0.9/Py), 253/sqrt(Fyc/1000)));
               end
            end
         else % >1/2.0
            inteqnLambdaCOL(upperColID,s.nlcNoSigma+ilc) = ...
               max(inteqnLambdaCOL(upperColID,s.nlcNoSigma+ilc), cw(upperColDV,9)/(52/sqrt(Fyc/1000)));
            Pu = currentCombo(1,upperColID);
            if Pu<0
               inteqnLambdaCOL(upperColID,ilc) = 0;
            else
               Py = cw(upperColDV,3)*Fyc;
               if Pu/(0.9*Py)<=0.125
                  inteqnLambdaCOL(upperColID,ilc) = ...
                     max(inteqnLambdaCOL(upperColID,ilc), ...
                         cw(upperColDV,11)/(520/sqrt(Fyc/1000)*(1-1.54*Pu/0.9/Py)));
               else
                  inteqnLambdaCOL(upperColID,ilc) = ...
                     max(inteqnLambdaCOL(upperColID,ilc), cw(upperColDV,11)...
                                   /max(191/sqrt(Fyc/1000)*(2.33-Pu/0.9/Py), 253/sqrt(Fyc/1000)));
               end
            end
         end
      end
   end
end

maxCOL = max(max(inteqnLambdaCOL));
```

```
if maxCOL>relaxed_constr_limits(4)
    flagBehavior=1;
    return;
elseif maxCOL>1.0
    constr_violation(4) = maxCOL-1;
end

%-------------------------------------------------------------------------------------------
%                calculate the ratio of 'inter-story drift ratio' requirement
%-------------------------------------------------------------------------------------------

percentPanelZone=0.0; % 'no deformation contribution from panel zone'.
Cd = 5.5;             % ductility factor for SMRF
Importance = 1;       % seismic user group I
calculatedDrift = zeros(s.num_stories,1);
for istory = 1:s.num_stories
   representColID = (s.num_bays+1)*(istory-1)+1;
   lowerDisp = r.dl(s.da(representColID,1),9);
   upperDisp = r.dl(s.da(representColID,4),9);
   calculatedDrift(istory) = abs(upperDisp-lowerDisp)*Cd/Importance/(1-percentPanelZone)*s.drift_Cs_factor;
end

%-------------------------------------------------------------------------------------------
%                calculate the ratio of 'story stability factor' requirement
%-------------------------------------------------------------------------------------------

theta_per_story = zeros(s.num_stories,1);  % stability factors for each floor
for istory = 1:s.num_stories
   Px=0;
   for jstory=istory:s.num_stories
      Px = Px + abs(s.D(jstory)+0.25*s.L(jstory))*s.num_bays*s.bay_width;
   end
   delta = abs(calculatedDrift(istory));
   Vx = sum(s.E(istory:s.num_stories));    % lateral seismic forces at current story
   hsx = s.story_height(istory);
   Cd = 5.5;
   theta_per_story(istory) = Px*delta/Vx/hsx/Cd;
end
theta_max=max(theta_per_story);
beta=1;                                    % conservative value as the ratio of shear demand v.s. capacity
theta_allowed=min(0.5/beta/Cd,0.25);
inteqnStability = theta_max/theta_allowed;
if inteqnStability>relaxed_constr_limits(6)
    flagBehavior=1;
    return;
elseif inteqnStability>1.0
    constr_violation(6) = inteqnStability-1;
end

allowableDrift = 0.02*s.story_height;      % allowable story drift ratio for SMRF, seismicuse group I
inteqnDR=calculatedDrift./(1-theta_per_story)./allowableDrift;
maxDR = max(inteqnDR);
if maxDR>relaxed_constr_limits(5)
    flagBehavior=1;
    return;
elseif maxDR>1.0
    constr_violation(5) = maxDR-1;
end
if sum(constr_violation)>0  % constraints are violated within pre-defined limits as in 'relaxed_constr_limits'
    flagBehavior=2;
end



function [ComboPrimary,ComboB1,ComboB2,linkB1B2,eloadPrimary,s] = LRFD_combo(s,r)

% assemble member forces according to LRFD load combinations.

% output arguments------------------------------------------------------------------------
%                'ComboPrimary': load combinations per AISC-LRFD
%                'ComboB1':      non-sidesway load combinations for B1 calculation
%                'ComboB2':      sidesway load combinations for B2 calculation
%                'linkB1B2':     relation between 'ComboB1' and 'ComboB2'
%                'eloadPrimary': combination of element loads
%-------------------------------------------------------------------------------------------

rho=1.0; % redundancy factor
SIGMA=3; % system overstrength factor, load combinations w/ this factor are for
         % COLUMNS checking only.
```

```
% 'primary' load combinations are:

              % only vertical loads

%  1) 1.4D
%  2) 1.2D + 1.6L + 0.5Lr        (live load: full)
%  3) 1.2D + 1.6L + 0.5Lr        (live load: stagger I)
%  4) 1.2D + 1.6L + 0.5Lr        (live load: stagger II)
%  5) 1.2D + 0.5L + 1.6Lr        (live load: full)
%  6) 1.2D + 0.5L + 1.6Lr        (live load: stagger I)
%  7) 1.2D + 0.5L + 1.6Lr        (live load: stagger II)

              % vertical and lateral loads

%  8) 1.2D + 0.5L   + 0.5Lr + 1.3W (live load: full)
%  9) 1.2D + 0.5L   + 0.5Lr + 1.3W (live load: stagger I)
% 10) 1.2D + 0.5L   + 0.5Lr + 1.3W (live load: stagger II)
% 11) 0.9D                  + 1.3W
% 12) 1.2D          + 1.6Lr + 0.8W (live load: full)
% 13) 1.2D          + 1.6Lr + 0.8W (live load: stagger I)
% 14) 1.2D          + 1.6Lr + 0.8W (live load: stagger II)
% 15) (1.2+0.2Sds)D + 0.5L         + rho*E     (live load: full)
% 16) (1.2+0.2Sds)D + 0.5L         + rho*E     (live load: stagger I)
% 17) (1.2+0.2Sds)D + 0.5L         + rho*E     (live load: stagger II)
% 18) (1.2-0.2Sds)D + 0.5L         - rho*E     (live load: full)
% 19) (1.2-0.2Sds)D + 0.5L         - rho*E     (live load: stagger I)
% 20) (1.2-0.2Sds)D + 0.5L         - rho*E     (live load: stagger II)
% 21) (0.9+0.2Sds)D                + rho*E
% 22) (0.9-0.2Sds)D                - rho*E

              % vertical and lateral loads for COLUMNS checking ONLY

% 23) (1.2+0.2*SIGMA*Sds)D + 0.5L  + SIGMA*rho*E     (live load: full)
% 24) (1.2+0.2*SIGMA*Sds)D + 0.5L  + SIGMA*rho*E     (live load: stagger I)
% 25) (1.2+0.2*SIGMA*Sds)D + 0.5L  + SIGMA*rho*E     (live load: stagger II)
% 26) (0.9-0.2*SIGMA*Sds)D         - SIGMA*rho*E


% 'secondary' load combinations (w/o sidesway moments for B1 calculation):
%  1) 1.2D + 0.5L + 0.5Lr (live load: full)               :related to primary combo 8
%  2) 1.2D + 0.5L + 0.5Lr (live load: stagger I)          :related to primary combo 9
%  3) 1.2D + 0.5L + 0.5Lr (live load: stagger II)         :related to primary combo 10
%  4) 0.9D                                                :related to primary combo 11
%  5) 1.2D        + 1.6Lr (live load: full)               :related to primary combo 12
%  6) 1.2D        + 1.6Lr (live load: stagger I)          :related to primary combo 13
%  7) 1.2D        + 1.6Lr (live load: stagger II)         :related to primary combo 14
%  8) (1.2+0.2Sds)D + 0.5L(live load: full)               :related to primary combo 15
%  9) (1.2+0.2Sds)D + 0.5L(live load: stagger I)          :related to primary combo 16
% 10) (1.2+0.2Sds)D + 0.5L(live load: stagger II)         :related to primary combo 17
% 11) (1.2-0.2Sds)D + 0.5L(live load: full)               :related to primary combo 18
% 12) (1.2-0.2Sds)D + 0.5L(live load: stagger I)          :related to primary combo 19
% 13) (1.2-0.2Sds)D + 0.5L(live load: stagger II)         :related to primary combo 20
% 14) (0.9+0.2Sds)D                                      :related to primary combo 21
% 15) (0.9-0.2Sds)D                                      :related to primary combo 22
% 16) (1.2+0.2*SIGMA*Sds)D + 0.5L (live load: full)       :related to primary combo 23
% 17) (1.2+0.2*SIGMA*Sds)D + 0.5L (live load: stagger I)  :related to primary combo 24
% 18) (1.2+0.2*SIGMA*Sds)D + 0.5L (live load: stagger II) :related to primary combo 25
% 19) (0.9-0.2*SIGMA*Sds)D                                :related to primary combo 26


% load combinations w/ sidesway moments when lateral loads are present:
%  1)  1.3W                                          :related to primary combo 8,9,10,11
%  2)  0.8W                                          :related to primary combo 12,13,14
%  3)  rho*E                                         :related to primary combo 15,16,17,21
%  4) -rho*E                                         :related to primary combo 18,19,20,22
%  5)  SIGMA*rho*E                                   :related to primary combo 23,24,25
%  6) -SIGMA*rho*E                                   :related to primary combo 26

%r.f(:,:,1) contains forces due to D  only
%r.f(:,:,2) contains forces due to L  only (full)
%r.f(:,:,3) contains forces due to L  only (stagger I)
%r.f(:,:,4) contains forces due to L  only (stagger II)
%r.f(:,:,5) contains forces due to Lr only (full)
%r.f(:,:,6) contains forces due to Lr only (stagger I)
%r.f(:,:,7) contains forces due to Lr only (stagger II)
%r.f(:,:,8) contains forces due to W  only
%r.f(:,:,9) contains forces due to E  only

s.nlcNoSigma=22;              % # of primary load combinations excluding those with 'SIGMA'
s.nlc=26;                     % # of primary load combinations
s.nlcVert=7;                  % # of primary load combinations which contain ONLY vertical loads
```

```
s.nlcHor=s.nlc-s.nlcVert;     % # of primary load combinations which contain BOTH vertical and horizontal loads
s.nlcHorNoSigma=s.nlcHor-4;   % # of primary load combinations which contain BOTH vertical and horizontal loads
                              %   excluding those with 'SIGMA'

CombPrimary=zeros(6,s.nel,s.nlc);

ComboPrimary(:,:, 1)= 1.4*r.f(:,:,1);
ComboPrimary(:,:, 2)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,2) + 0.5*r.f(:,:,5);
ComboPrimary(:,:, 3)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,3) + 0.5*r.f(:,:,6);
ComboPrimary(:,:, 4)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,4) + 0.5*r.f(:,:,7);
ComboPrimary(:,:, 5)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,2) + 1.6*r.f(:,:,5);
ComboPrimary(:,:, 6)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,3) + 1.6*r.f(:,:,6);
ComboPrimary(:,:, 7)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,4) + 1.6*r.f(:,:,7);
ComboPrimary(:,:, 8)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,2) + 0.5*r.f(:,:,5) + 1.3*r.f(:,:,8);
ComboPrimary(:,:, 9)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,3) + 0.5*r.f(:,:,6) + 1.3*r.f(:,:,8);
ComboPrimary(:,:,10)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,4) + 0.5*r.f(:,:,7) + 1.3*r.f(:,:,8);
ComboPrimary(:,:,11)= 0.9*r.f(:,:,1)                                    + 1.3*r.f(:,:,8);
ComboPrimary(:,:,12)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,5) + 0.8*r.f(:,:,8);
ComboPrimary(:,:,13)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,6) + 0.8*r.f(:,:,8);
ComboPrimary(:,:,14)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,7) + 0.8*r.f(:,:,8);

ComboPrimary(:,:,15)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2) + rho*r.f(:,:,9);
ComboPrimary(:,:,16)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3) + rho*r.f(:,:,9);
ComboPrimary(:,:,17)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4) + rho*r.f(:,:,9);

ComboPrimary(:,:,18)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2) - rho*r.f(:,:,9);
ComboPrimary(:,:,19)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3) - rho*r.f(:,:,9);
ComboPrimary(:,:,20)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4) - rho*r.f(:,:,9);

ComboPrimary(:,:,21)= (0.9+0.2*s.Sds)*r.f(:,:,1)                  + rho*r.f(:,:,9);
ComboPrimary(:,:,22)= (0.9-0.2*s.Sds)*r.f(:,:,1)                  - rho*r.f(:,:,9);

ComboPrimary(:,:,23)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2) + SIGMA*rho*r.f(:,:,9);
ComboPrimary(:,:,24)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3) + SIGMA*rho*r.f(:,:,9);
ComboPrimary(:,:,25)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4) + SIGMA*rho*r.f(:,:,9);

ComboPrimary(:,:,26)= (0.9-0.2*SIGMA*s.Sds)*r.f(:,:,1)                  - SIGMA*rho*r.f(:,:,9);

ComboB1=zeros(6,s.nel,s.nlcHor);

ComboB1(:,:, 1)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,2) + 0.5*r.f(:,:,5);
ComboB1(:,:, 2)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,3) + 0.5*r.f(:,:,6);
ComboB1(:,:, 3)= 1.2*r.f(:,:,1) + 0.5*r.f(:,:,4) + 0.5*r.f(:,:,7);
ComboB1(:,:, 4)= 0.9*r.f(:,:,1);
ComboB1(:,:, 5)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,5);
ComboB1(:,:, 6)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,6);
ComboB1(:,:, 7)= 1.2*r.f(:,:,1) + 1.6*r.f(:,:,7);

ComboB1(:,:, 8)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2);
ComboB1(:,:, 9)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3);
ComboB1(:,:,10)= (1.2+0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4);

ComboB1(:,:,11)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2);
ComboB1(:,:,12)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3);
ComboB1(:,:,13)= (1.2-0.2*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4);

ComboB1(:,:,14)= (0.9+0.2*s.Sds)*r.f(:,:,1);
ComboB1(:,:,15)= (0.9-0.2*s.Sds)*r.f(:,:,1);

ComboB1(:,:,16)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,2);
ComboB1(:,:,17)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,3);
ComboB1(:,:,18)= (1.2+0.2*SIGMA*s.Sds)*r.f(:,:,1) + 0.5*r.f(:,:,4);

ComboB1(:,:,19)= (0.9-0.2*SIGMA*s.Sds)*r.f(:,:,1);

ComboB2=zeros(6,s.nel,6);

ComboB2(:,:,1)=  1.3*r.f(:,:,8);
ComboB2(:,:,2)=  0.8*r.f(:,:,8);
ComboB2(:,:,3)=  rho*r.f(:,:,9);
ComboB2(:,:,4)= -rho*r.f(:,:,9);
ComboB2(:,:,5)=  SIGMA*rho*r.f(:,:,9);
ComboB2(:,:,6)= -SIGMA*rho*r.f(:,:,9);

linkB1B2=[1 1 1 1 2 2 2 3 3 3 4 4 4 3 4 5 5 5 6];

eloadPrimary=zeros(s.nel,s.nlc);

eloadPrimary(:, 1)= 1.4*s.eload(:,1);
eloadPrimary(:, 2)= 1.2*s.eload(:,1) + 1.6*s.eload(:,2) + 0.5*s.eload(:,5);
eloadPrimary(:, 3)= 1.2*s.eload(:,1) + 1.6*s.eload(:,3) + 0.5*s.eload(:,6);
```

```
eloadPrimary(:, 4)= 1.2*s.eload(:,1) + 1.6*s.eload(:,4) + 0.5*s.eload(:,7);
eloadPrimary(:, 5)= 1.2*s.eload(:,1) + 0.5*s.eload(:,2) + 1.6*s.eload(:,5);
eloadPrimary(:, 6)= 1.2*s.eload(:,1) + 0.5*s.eload(:,3) + 1.6*s.eload(:,6);
eloadPrimary(:, 7)= 1.2*s.eload(:,1) + 0.5*s.eload(:,4) + 1.6*s.eload(:,7);
eloadPrimary(:, 8)= 1.2*s.eload(:,1) + 0.5*s.eload(:,2) + 0.5*s.eload(:,5);
eloadPrimary(:, 9)= 1.2*s.eload(:,1) + 0.5*s.eload(:,3) + 0.5*s.eload(:,6);
eloadPrimary(:,10)= 1.2*s.eload(:,1) + 0.5*s.eload(:,4) + 0.5*s.eload(:,7);
eloadPrimary(:,11)= 0.9*s.eload(:,1);
eloadPrimary(:,12)= 1.2*s.eload(:,1) + 1.6*s.eload(:,5);
eloadPrimary(:,13)= 1.2*s.eload(:,1) + 1.6*s.eload(:,6);
eloadPrimary(:,14)= 1.2*s.eload(:,1) + 1.6*s.eload(:,7);

eloadPrimary(:,15)= (1.2+0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,2);
eloadPrimary(:,16)= (1.2+0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,3);
eloadPrimary(:,17)= (1.2+0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,4);

eloadPrimary(:,18)= (1.2-0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,2);
eloadPrimary(:,19)= (1.2-0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,3);
eloadPrimary(:,20)= (1.2-0.2*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,4);

eloadPrimary(:,21)= (0.9+0.2*s.Sds)*s.eload(:,1);
eloadPrimary(:,22)= (0.9-0.2*s.Sds)*s.eload(:,1);

eloadPrimary(:,23)= (1.2+0.2*SIGMA*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,2);
eloadPrimary(:,24)= (1.2+0.2*SIGMA*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,3);
eloadPrimary(:,25)= (1.2+0.2*SIGMA*s.Sds)*s.eload(:,1) + 0.5*s.eload(:,4);

eloadPrimary(:,26)= (0.9-0.2*SIGMA*s.Sds)*s.eload(:,1);
```

```
function PnColComp = column_comp_strength(s,cw,Kcol)

% calculate design compression strength for flexural buckling of columns
% according to E2, AISC-LRFD pg 6-47
% the output PnColComp is of size [s.numcol by 2] for non-sidesway and sidesway casea, respectively

PnColComp=zeros(s.numcol,2);

for icolumn=1:s.numcol
    dv_current=s.spdv(s.sps(icolumn));
    area=cw(dv_current,3);
    rx=cw(dv_current,24);               % radius of gyration
    E=s.mod(s.mps(icolumn));            % modulus of elasticity
    Fy=s.yieldC(s.mps(icolumn));        % yield strength
    len=s.l(icolumn);
    for j=1:2
        Lambdac=Kcol(icolumn,j)*len*sqrt(Fy/E)/rx/pi;
        if Lambdac<=1.5
            Fcr=0.658^(Lambdac*Lambdac)*Fy;
        else
            Fcr=0.877*Fy/Lambdac/Lambdac;
        end
        PnColComp(icolumn,j)=area*Fcr;
    end
end
```

```
function B2 = B2_factor(s,cw,Kcol,currentCombo)

% calculate amplification factor B2 (sidesway part of P-delta effects) for each story
% with the current load combination 'currentCombo'
% B2 is [s.num_stories by 1].
% NOTE: only primary load cases 8~18 (totally 11) need B2 factors.
% 'Kcol': created by 'Kfactor'

B2=zeros(s.num_stories,1);

for istory=1:s.num_stories
    SumPu=0;
    SumPe2=0;
    for j=1:s.num_bays+1                % loop over all columns within current story
       colID=(istory-1)*(s.num_bays+1)+j;
       dv_pointer=s.spdv(s.sps(colID));
       area=cw(dv_pointer,3);
       rx=cw(dv_pointer,24);
       E=s.mod(s.mps(colID));
       Fy=s.yieldC(s.mps(colID));
       SumPu=SumPu + currentCombo(1,colID,1);
       lambdac=Kcol(colID,2)*s.l(colID)/(rx*pi)*sqrt(Fy/E);
```

```
       SumPe2=SumPe2 + area*Fy/lambdac^2;
    end
    B2(istory)=max(1/(1-SumPu/SumPe2),1); % must be greater than 1
end
```

F.3 Static pushover analysis of a valid SMRF design using DRAIN-2DX

```
function [] = create_DRAIN_INP_mode(s,bw,cw)

% create "DRAIN.INP" of the current SMRF 's' for DRAIN-2DX
% obtain the realistic vibration periods only.


%-----------------------------------------------------------------------------------------
%                control informatin for SMRF
%-----------------------------------------------------------------------------------------

% UNIT: kips, inch

X_code=1;
Y_code=1;
R_code=1;
Pdelta_column=1;
Pdelta_beam=0;
StrainHardRatioCol=0.03;
StrainHardRatiobeam=0.03;
Fyc=57.6; %[ksi] for A50 steel
Fyb=49.2; %[ksi] for A36 steel

%-----------------------------------------------------------------------------------------
%                control informatin for DRAIN-2DX analysis
%-----------------------------------------------------------------------------------------

KEXE=0;
ECHO=0;
KPDEL=1;
ENERGY=1;

PROBLEM_NAME='MULTIOBJ';
SESSION_TITLE=strcat(['GENERATED AT ',datestr(now)]);

%-----------------------------------------------------------------------------------------
%                create DRAIN.INP
%-----------------------------------------------------------------------------------------

fid = fopen('DRAIN.INP','w');
fprintf(fid,'*STARTXX\n');
fprintf(fid,'%2s%8s%9s%1d%1s%1d%1s%1d%1s%1d%16s%40s\n',...
          '',PROBLEM_NAME,'',KEXE,'',ECHO,'',KPDEL,'',ENERGY,'',SESSION_TITLE);
fprintf(fid,'*NODECOORDS\n');
for i=1:s.num_bays+1
    fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',i,s.bay_width*(i-1),0);
end
for i=1:s.num_bays+1
    fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',s.num_bays+1+i,s.bay_width*(i-1),s.story_height(1));
end
for i=1:s.num_bays+1
    fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',s.nn-s.num_bays-1+i,s.bay_width*(i-1),sum(s.story_height));
end
for i=1:s.num_bays+1
    fprintf(fid,'%1s%9d%10d%10d\n','L',s.num_bays+1+i,s.nn-s.num_bays-1+i,s.num_bays+1);
end

% define nodes for the dummy gravity column to consider P-delta
fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',s.nn+1,s.bay_width*(s.num_bays+1),0);
fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',s.nn+2,s.bay_width*(s.num_bays+1),s.story_height(1));
fprintf(fid,'%1s%9d%10.2f%10.2f\n','C',s.nn+1+s.num_stories,s.bay_width*(s.num_bays+1),sum(s.story_height));
fprintf(fid,'%1s%9d%10d%10d\n','L',s.nn+2,s.nn+1+s.num_stories,1);

fprintf(fid,'*RESTRAINTS\n');
fprintf(fid,'%1s%1s%1d%1d%1d%10d%10d\n','S','',X_code,Y_code,R_code,1,s.num_bays+1);

% define the pinned condition for the dummy gravity column
fprintf(fid,'%1s%1s%1d%1d%1d%10d\n','S','',1,1,0,s.nn+1);
```

```
fprintf(fid,'*SLAVING\n');
for istory=1:s.num_stories
    fprintf(fid,'%1s%1s%1d%1d%1d%10d%10d%10d\n','S','',1,0,0,istory*(s.num_bays+1)+1,...
                istory*(s.num_bays+1)+2,(istory+1)*(s.num_bays+1));
end

% define the slaving condition for the dummy gravity column
for istory=1:s.num_stories
    fprintf(fid,'%1s%1s%1d%1d%1d%10d%10d\n','S','',1,0,0,istory*(s.num_bays+1)+1,s.nn+1+istory);
end

fprintf(fid,'*MASSES\n');
for istory=1:s.num_stories
    nodeID=istory*(s.num_bays+1)+1;
    nodeMASS=s.totalWeightPerStory(istory)/s.num_bays/1000;  % change from lb to kip
    fprintf(fid,'%1s%1s%1d%1d%1d%10.2f%10d%10d%30s%5.1f%10.2f\n','S','',...
                1,0,0,nodeMASS/2,nodeID,nodeID,'',386.4,1.0);
    fprintf(fid,'%1s%1s%1d%1d%1d%10.2f%10d%10d%30s%5.1f%10.2f\n','S','',...
                1,0,0,nodeMASS,nodeID+1,nodeID+s.num_bays-1,'',386.4,1.0);
    fprintf(fid,'%1s%1s%1d%1d%1d%10.2f%10d%10d%30s%5.1f%10.2f\n','S','',...
                1,0,0,nodeMASS/2,nodeID+s.num_bays,nodeID+s.num_bays,'',386.4,1.0);
end
fprintf(fid,'*ELEMENTGROUP\n');
fprintf(fid,'%5d%4s%1d%4s%1d%5s%10.2f%40s\n',2,'',1,'',Pdelta_column,'',1,'COLUMNS');
fprintf(fid,'%5d%5d%5d\n',s.num_col_dv,0,s.num_col_dv);
fprintf(fid,'!STIFFNESS TYPES\n');
for icoldv=1:s.num_col_dv
    colDVid=s.spdv(icoldv);
    area=cw(colDVid,3);
    Ix=cw(colDVid,22);
    fprintf(fid,'%5d%10.1f%10.2f%10.1f%10.2f%5.1f%5.1f%5.1f\n',...
                icoldv,s.mod/1000,StrainHardRatioCol,area,Ix,4,4,2);
end
fprintf(fid,'!YIELD SURFACE TYPES\n');
for icoldv=1:s.num_col_dv
    colDVid=s.spdv(icoldv);
    area=cw(colDVid,3);
    Zx=cw(colDVid,28); % plastic section modulus
    Myplus=Zx*Fyc;
    Myminus=Zx*Fyc;
    Pyc=area*Fyc;
    Pyt=area*Fyc;
    fprintf(fid,'%5d%5d%10.1f%10.1f%10.1f%10.1f%5.1f%5.3f%5.1f%5.3f\n',...
                icoldv,2,Myplus,Myminus,Pyc,Pyt,1,.125,1,.125);
end
fprintf(fid,'!ELEMENT GENERATION COMMANDS\n');
for icol=1:s.numcol
    colTypeID=s.sps(icol);
    fprintf(fid,'%5d%10d%10d%10s%5d%5s%5d%5d\n',...
                icol,s.i(icol),s.j(icol),'',colTypeID,'',colTypeID,colTypeID);
end
fprintf(fid,'*ELEMENTGROUP\n');
fprintf(fid,'%5d%4s%1d%4s%1d%5s%10.2f%40s\n',2,'',1,'',Pdelta_beam,'',1,'BEAMS/GIRDERS');
fprintf(fid,'%5d%5d%5d\n',s.numdv-s.num_col_dv,0,s.numdv-s.num_col_dv);
fprintf(fid,'!STIFFNESS TYPES\n');
for ibeamdv=1:s.numdv-s.num_col_dv
    beamDVid=s.spdv(s.num_col_dv+ibeamdv);
    area=bw(beamDVid,3);
    Ix=bw(beamDVid,22);
    fprintf(fid,'%5d%10.1f%10.2f%10.1f%10.1f%5.1f%5.1f%5.1f\n',...
                ibeamdv,s.mod/1000,StrainHardRatiobeam,area,Ix,4,4,2);
end
fprintf(fid,'!YIELD SURFACE TYPES\n');
for ibeamdv=1:s.numdv-s.num_col_dv
    beamDVid=s.spdv(s.num_col_dv+ibeamdv);
    Zx=bw(beamDVid,28); %plastic section modulus
    Myplus=Zx*Fyb;
    Myminus=Zx*Fyb;
    fprintf(fid,'%5d%5d%10.1f%10.1f\n',ibeamdv,1,Myplus,Myminus);
end
fprintf(fid,'!ELEMENT GENERATION COMMANDS\n');
for ibeam=1:s.nel-s.numcol
    beamID=s.numcol+ibeam;
    beamTypeID=s.sps(beamID)-s.num_col_dv;
    fprintf(fid,'%5d%10d%10d%10s%5d%5s%5d%5d\n',...
                ibeam,s.i(beamID),s.j(beamID),'',beamTypeID,'',beamTypeID,beamTypeID);
end

% define rigid links bwtween SMRF and the dummy column
largeA = 1000000;
largeI = 1000000;
```

```
smallI = 0.1;
fprintf(fid,'*ELEMENTGROUP\n');
fprintf(fid,'%5d%4s%1d%4s%1d%5s%10.2f%40s\n',2,'',1,'',Pdelta_column,'',1,'RIGID LINKS');
fprintf(fid,'%5d%5d%5d\n',1,0,1);  % only one rigid link type is necessary
fprintf(fid,'!STIFFNESS TYPES\n'); % provide infinite axial strength and infinite flexural resistance
fprintf(fid,'%5d%10.1f%10.2f%10.1f%10.2f%5.1f%5.1f%5.1f\n',...
           1,s.mod/1000,StrainHardRatioCol,largeA,largeI,0,0,0); % assgin huge numbers, also pinned ends
fprintf(fid,'!YIELD SURFACE TYPES\n');
fprintf(fid,'%5d%5d%10.1f%10.1f%10.1f%10.1f%5.1f%5.3f%5.1f%5.3f\n',...
           1,2,1000000,1000000,1000000,1000000,1,.125,1,.125);
fprintf(fid,'!ELEMENT GENERATION COMMANDS\n');
for istory=1:s.num_stories
    fprintf(fid,'%5d%10d%10d%10s%5d%5s%5d%5d\n',istory,(s.num_bays+1)*(1+istory),s.nn+1+istory,'',1,'',1,1);
end
%define the dummy column
fprintf(fid,'*ELEMENTGROUP\n');
fprintf(fid,'%5d%4s%1d%4s%1d%5s%10.2f%40s\n',2,'',1,'',Pdelta_column,'',1,'DUMMY COLUMN');
fprintf(fid,'%5d%5d%5d\n',1,0,1);  % only one dummy column type is necessary
fprintf(fid,'!STIFFNESS TYPES\n'); % provide infinite axial strength and no flexural resistance
fprintf(fid,'%5d%10.1f%10.2f%10.1f%10.2f%5.1f%5.1f%5.1f\n',...
           1,s.mod/1000,StrainHardRatioCol,largeA,smallI,4,4,2); %assgin huge numbers
fprintf(fid,'!YIELD SURFACE TYPES\n');
fprintf(fid,'%5d%5d%10.1f%10.1f%10.1f%10.1f%5.1f%5.3f%5.1f%5.3f\n',...
           1,2,1,1,1000000,1000000,1,.125,1,.125);
fprintf(fid,'!ELEMENT GENERATION COMMANDS\n');
for istory=1:s.num_stories
    fprintf(fid,'%5d%10d%10d%10s%5d%5s%5d%5d\n',istory,s.nn+istory,s.nn+1+istory,'',1,'',1,1);
end

fprintf(fid,'*ELEMLOAD\n');
fprintf(fid,'%1s%4s%35s%40s\n','','VERT','','UNFACTORED DL&LL FROM TRIBUTARY AREAS');
fprintf(fid,'%1s%4d%5d\n','G',2,s.num_stories);
for istory=1:s.num_stories
    load=abs(s.D(istory)+0.25*s.L(istory))/1000;  % change from lb/in to kip/in
    V=load*s.bay_width/2;
    M=1/12*load*s.bay_width*s.bay_width;
    fprintf(fid,'%5d%5d%10.1f%10.1f%10.1f%10.1f%10.1f%10.1f%10.1f%10.1f\n',istory,0,1,0.0,V,M,0.0,V,-M);
end
for istory=1:s.num_stories
    fprintf(fid,'%5d%5d%5s%5d%10.2f\n',(istory-1)*s.num_bays+1,istory*s.num_bays,'',istory,1.0);
end
fprintf(fid,'\n'); % an intentious blank line required by DRAIN-2DX

% apply interior gravity loads to the dummy column at each story level
interior_area = (s.num_bays-1)*s.bay_width*(s.num_baysY-1)*s.bay_width; % in^2
Paxial=interior_area*(s.dead_intensity+0.25*s.live_intensity)/1000;     % kips
fprintf(fid,'*NODALOAD\n');
fprintf(fid,'%1s%2s%35s%40s\n','','INTR','','UNFACTORED DL&LL FROM INTERIOR AREAS');
for istory=1:s.num_stories
    fprintf(fid,'%1s%9.4f%10.4f%10.4f%10d\n','S',0,-Paxial(istory),0,s.nn+1+istory);
end

fprintf(fid,'*PARAMETERS\n');
fprintf(fid,'%2s%3s%5d%5d%5d%5d\n','OS','',0,0,1,0,0);
fprintf(fid,'*GRAV\n');
fprintf(fid,'%1s%5s%4s%10.1f\n','E','','VERT',1.0);
fprintf(fid,'%1s%5s%4s%10.1f\n','N','','INTR',1.0);
fprintf(fid,'*MODE\n');
fprintf(fid,'%5d%10s%5d%5d%5d\n',1,'',1,1,1);
fprintf(fid,'*STOP\n');
fclose(fid);
```

function [T1,flagMODE]=get_DRAIN_mode()

```
% extract the fundamental (i.e., the first) natural period of the SMRF
% from the .OUT file generated by DRAIN-2DX.

fid = fopen('MULTIOBJ.OUT','r');
for i=1:12              % skip the first 12 lines
    tmp=fgetl(fid);
end
if feof(fid)==1         % reach the end of file, which indicates that
    flagMODE=1;         % calcualtion of period is not successful.
    T1=0;
    fclose(fid);
    return;
end
for i=1:8               % skip the next 8 lines
    tmp=fgetl(fid);
```

```
end
tmp=fgetl(fid);
T1=str2num(tmp(8:22)); % first period
fclose(fid);
flagMODE=0;            % mission successful
```

```
function [] = create_DRAIN_INP_pushover(s,bw,cw,pushover_profile,pushLimit)

% create "DRAIN.INP" of the current SMRF 's' for DRAIN-2DX
% for static pushover analysis.

%  'PushLimit' is the maximum roof deflection (UNIT: inch) used in DRAIN.INP.

< same coding as in 'create_DRAIN_INP_mode' not shown >

fprintf(fid,'*RESULTS\n');
fprintf(fid,'%3s%4s%1d%1d%1d%10d%10d%10d\n','NSD','',...
            0,0,1,(s.num_bays+1)+1,s.num_stories*(s.num_bays+1)+1,s.num_bays+1);

fprintf(fid,'*NODALOAD\n');
fprintf(fid,'%1s%2s%35s%40s\n','','NHRP','','PUSHOVWE ANALYSIS UNIT HORIZ LOAD');
for istory=1:s.num_stories
    fprintf(fid,'%1s%9.4f%10.4f%10.4f%10d\n','S',...
                pushover_profile(istory),0,0,(s.num_bays+1)*istory+1);
end

fprintf(fid,'*PARAMETERS\n');
fprintf(fid,'%2s%3s%5d%5d%5d%5d%5d\n','OS','',0,0,1,0,0);
fprintf(fid,'*GRAV\n');
fprintf(fid,'%1s%5s%4s%10.1f\n','E','','VERT',1.0);
fprintf(fid,'%1s%5s%4s%10.1f\n','N','','INTR',1.0);
fprintf(fid,'*STAT\n');
fprintf(fid,'%1s%5s%4s\n','N','','NHRP');
fprintf(fid,'%1s%9d%14s%1d%10.1f%10.1f%5d\n','D',...
            (s.num_bays+1)*s.num_stories+1,'',1,1.0,pushLimit,MAXEV);
fprintf(fid,'*STOP\n');
fclose(fid);
```

```
function [PHI_yield,Kpre,Kpost,alpha,Dy,flagPUSH,flagALPHA,flagBILINEAR,Sy,DeflectProfile]=...
                            get_DRAIN_pushover(s,PushLimit,PushNumber,PushStep)

% extract the lateral deflection profiles at yield point of bilinear curve
% from the .OUT file generated by DRAIN-2DX.

PHI_yield=zeros(s.num_stories,1);
Kpre=0;
Kpost=0;
alpha=0;
Dy=0;
Sy=0;
flagPUSH=0;
flagALPHA=0;
flagBILINEAR=0;

fid = fopen('MULTIOBJ.OUT','r');
for i=1:12                          % skip the first 12 lines
    tmp=fgetl(fid);
end
for istory=1:s.num_stories
    for i=1:6                       % skip 4 lines for each story info
        tmp=fgetl(fid);
    end
end
for i=1:6                           % skip the 6 lines
    tmp=fgetl(fid);
end
num_steps = PushLimit/PushStep;
DeflectProfile=zeros(num_steps,s.num_stories);
for istory=1:s.num_stories
    for i=1:5                       % skip the 5 lines
        tmp=fgetl(fid);
    end
    A=fscanf(fid,'%d%e%e%e%e',[5,num_steps]);
    A=A';
    [rownum,colnum]=size(A);
    if rownum~=num_steps
        flagPUSH=1;
```

```
        fclose(fid);
        return;
    end
    DeflectProfile(:,istory)=A(:,3);
end
fclose(fid);

%-------------------------------------------------------------------------------------------
%                get the global (roof) push-over curve
%-------------------------------------------------------------------------------------------

RoofPushCurve=[A(:,3) A(:,2)];    % global (roof) push-over curve

%-------------------------------------------------------------------------------------------
%                approximate the roof pushover curve by binear curve
%-------------------------------------------------------------------------------------------

N = 3;
Npre=N;
ptx = RoofPushCurve(1:Npre,1);
mx = mean(ptx);
pty = RoofPushCurve(1:Npre,2);
my = mean(pty);
SXX = ptx'*ptx-Npre*mx*mx;
SXY = ptx'*pty-Npre*mx*my;
Kpre = SXY/SXX;
intcptPre = my - Kpre*mx;

Npost=N*PushNumber;
start = num_steps-2*Npost-1;
ptx = RoofPushCurve(start+1:start+Npost,1);
mx = mean(ptx);
pty = RoofPushCurve(start+1:start+Npost,2);
my = mean(pty);
SXX = ptx'*ptx-Npost*mx*mx;
SXY = ptx'*pty-Npost*mx*my;
Kpost = SXY/SXX;
intcptPost = my - Kpost*mx;

alpha = Kpost/Kpre;               % strain-hardening ratio
if alpha<0
    flagALPHA=1;
    return;
end

RHS = [intcptPre;intcptPost];
LHS = [-Kpre,1;-Kpost,1];
intersection = LHS\RHS;
Dy=intersection(1);               % yield displacement [in]
Sy=intersection(2);               % yield strength Sy  [kips]

[Kpre,Kpost,Dy,Sy,flagBILINEAR]=fit_pushover_BILINEAR(RoofPushCurve,Kpre,Kpost,Dy);

if flagBILINEAR==1                % unable to fit the pushover curve by a bilinear model
    return;
end

Sy=Sy/sum(s.totalWeightPerStory/1000);  % system yield coefficient

RoofDrift=Dy;
lowerBound=floor(RoofDrift/PushStep);
upperBound=ceil(RoofDrift/PushStep);
RoofDrift_LB = DeflectProfile(lowerBound,end);
RoofDrift_UB = DeflectProfile(upperBound,end);
PHI_yield=DeflectProfile(lowerBound,:)+(DeflectProfile(upperBound,:)-DeflectProfile(lowerBound,:))...
                                    *(RoofDrift-RoofDrift_LB)/(RoofDrift_UB-RoofDrift_LB);
PHI_yield=PHI_yield/PHI_yield(end);     % determine deflection profile at yield displacement 'Dy'



function [Kpre,Kpost,Dy,Sy,flagESDOF]=fit_pushover_BILINEAR(curve,Kpre,Kpost,Dy)

Sy=0;
flagESDOF=0;

RoofDeflectionVector = curve(:,1);
LateralForceVector = curve(:,2);

NN = length(RoofDeflectionVector);

Re=Kpre;
```

```
Rp=Kpost;
F0=Kpre*Dy;
initial_vector=[Re,Rp,F0]';
options = optimset('Display','off','TolFun',1e-3,'MaxFunEvals',2000);
[p,tmp1,tmp2,exitflagP]=lsqnonlin('BILINEAR',initial_vector,[],[],options,...
                                  RoofDeflectionVector,LateralForceVector);
if exitflagP<=0
    p=[];
    flagESDOF=1;
    return;
end
Re=p(1);
Rp=p(2);
F0=p(3);
Kpre = Re;
Kpost = Rp;
Dy = F0/(Re-Rp);
Sy = Re*Dy;


function F = BILINEAR(p,RoofDeflectionVector,LateralForceVector)

Re=p(1);
Rp=p(2);
M0=p(3);

F=min(Re*RoofDeflectionVector,M0+Rp*RoofDeflectionVector)- LateralForceVector;


function [p, flagLN,maxDRvector,mu_vector] = equivalent_SDOF(s,PHI_yield,Kpre,Kpost,Dy,...
                                             alpha,pushoverProfile,DeflectProfile,PushStep)

PHI=PHI_yield/PHI_yield(end); % use deflection profile at yield point to develop the ESDOF system
PHI=PHI';

%-------------------------------------------------------------------------------------------
%                 stablish ESDOF system
%                 based on Krawinkler and Seneviratna (1998) in "Pros and cons of ..."
%-------------------------------------------------------------------------------------------

Vy = Kpre*Dy;                                           % MDOF yield force
Qy = Vy*pushoverProfile;                                % lateral load distibution
PHIprime = PHI';
Qy_star = PHIprime*Qy;                                  % ESDOF yield force
M_matrix = diag(s.totalWeightPerStory/1000/386.4);
M_star = abs(PHIprime*M_matrix*ones(s.num_stories,1));
xy_star = PHIprime*M_matrix*PHI/M_star*Dy;              % ESDOF yield displacement
Teq = 2*pi*sqrt(xy_star*M_star/Qy_star);                % ESDOF period
weq = 2*pi/Teq;

%-------------------------------------------------------------------------------------------
%                 calculate roof drift using ESDOF
%-------------------------------------------------------------------------------------------

PE50level = 5;
PE50vector=[0.75 0.5 0.10 0.05 0.02]';                  % probability of exccedance in 50 years
PEannualvector=-log(1-PE50vector)/50;                   % assumes a Poisson process

maxDRvector = zeros(1,PE50level);                       % max interstory drift ratios.
mu_vector = zeros(1,PE50level);                               % system ductility

for iPE=1:PE50level
    Sa = get_Sa_SAC(Teq,iPE);
    Sd = Sa/weq/weq;
    if Sd < xy_star   % within linear range
        RoofDispl=Sd*M_star/(PHIprime*M_matrix*PHI);
        driftRatioProfile = find_DriftRatioProfile(RoofDispl,DeflectProfile,...
                                                   PushStep,s.num_stories,s.story_height);
        maxDRvector(iPE) = max(abs(driftRatioProfile));
        mu_vector(iPE) = 1;
    else              % into non-linear range
        R = max(Sa*M_star/Qy_star,1);
        if alpha <= 0.02
            a=1;
            b=0.37+(0.42-0.37)*(0.02-alpha)/0.02;
        elseif alpha <= 0.1
            a=0.80+(1.00-0.80)*(0.10-alpha)/0.08;
            b=0.29+(0.37-0.29)*(0.10-alpha)/0.08;
        end
```

```
        c = Teq^a/(Teq^a+1) + b/Teq;
        mu = 1+1/c*(R^c-1);
        mu_vector(iPE) = mu;
        RoofDispl=Dy*mu;
        driftRatioProfile = find_DriftRatioProfile(RoofDispl,DeflectProfile,...
                                                   PushStep,s.num_stories,s.story_height);
        maxDRvector(iPE) = max(abs(driftRatioProfile));
    end
end

%-------------------------------------------------------------------------------------------
%                fit a lognormal distribution
%-------------------------------------------------------------------------------------------

p = LNfit(maxDRvector',PEannualvector,[-7 1]');

if max(imag(p))>1e-5
    flagLN=1;
    p=[0 0];
    return;
elseif isempty(p)
    flagLN=1;
    p=[0 0];
    return;
else
    p=real(p);
    flagLN=0;
end



function Sa = get_Sa_SAC(T,PElevel)

% get spectral acceleraton at the fundamental period
% in uniform hazard spectra of a particular PE level.

% calculate Sa at 75%PE in 50 yr and 5%PE in 50 yr
% based on FEMA 273 Section 2.6.1.3

%PE50 = [0.75  0.05];
%because Ss of 2%PE is >=1.5g
%PR = 1./(1-exp(0.02*log(1-PE50)));%mean return period
%Si_s = 107.0*(PR/475).^0.44; %0.44 is for California, Table 2-12 USE 20/50 value
%Si_1 =  68.0*(PR/475).^0.44;

SsData =  [ 133.7978  198.6096  413.4480  567.4059  622.1040];
S1Data =  [  85.0304  111.2832  262.7520  360.5944  459.8160];

Ss = SsData(PElevel);
S1 = S1Data(PElevel);

Sa = min(Ss,S1/T);



function DR_profile = find_DriftRatioProfile(RoofDispl,DeflectProfile,PushStep,num_stories,...
                                             single_story_height)

% obtain maximumm intertory drift ratio at the current target displacement level 'RoofDispl'

lowerBound = floor(RoofDispl/PushStep);
upperBound = ceil(RoofDispl/PushStep);
if lowerBound==0
    current_profile = DeflectProfile(upperBound,:)*RoofDispl/upperBound/PushStep;
else
    current_profile = DeflectProfile(lowerBound,:)+(DeflectProfile(upperBound,:)...
                      -DeflectProfile(lowerBound,:))*(RoofDispl-lowerBound*PushStep)/(upperBound-lowerBound);
end
current_profile = [0;current_profile']; % augment for ground level displ condition (fixed)
current_modeshpe=(current_profile/current_profile(num_stories+1))';
DR_profile = (current_profile(2:num_stories+1)-current_profile(1:num_stories))./single_story_height;
DR_profile = DR_profile';



function p = LNfit(DRvector,PEvector,startingPoint)

DRvector=[0.0001;DRvector];
PEvector=[1;PEvector];

options = optimset('Display','off','TolFun',1e-5,'MaxFunEvals',2000);
```

```
[p,tmp1,tmp2,exitflagP]=lsqnonlin('LN',startingPoint,[],[],options,DRvector,PEvector);

if exitflagP<=0
    p=[];
end


function F = LN(p,Sa_vector,PEvector)

% lognormal distribution fitting

F=(1-0.5*(1+erf((log(Sa_vector)-p(1))./(p(2)*sqrt(2))))-PEvector)./PEvector;
```

F.4 Nonlinear time history analysis using DRAIN-2DX

```
% main_NTHA

load beam_W_sec                   % database of beam sections
load col_W_sec                    % database of column sections
s=create_SMRF;
s_modify_2_5s4b_splice;

s.spdv=[ 71   71   71   55   71   71  119  119  119   72   72];  % an example SMRF

s=compute_general_data(s,bw,cw);
s=modify_s_Live09(s);
s=compute_bldg_weight(s,bw,cw);
create_DRAIN_INP_mode(s,bw,cw);
!DRAIN_2DX
[T1,flagMODE]=get_DRAIN_mode;

%-------------------------------------------------------------------------------------------
%                 compute alpha and beta values for damping info
%-------------------------------------------------------------------------------------------
xi1=0.04; %  use 4% viscous damping ratio at T1 and 0.2 sec
xi2=0.04;
w1=2*pi/T1;
w2=2*pi/0.2;
tmp=2*[w2,-w1;-1/w2,1/w1]*w1*w2/(w2*w2-w1*w1)*[xi1;xi2];
alpha_damp=tmp(1);
beta_damp=tmp(2);
record_array=[
    'la41','la42','la43','la44','la45','la46','la47','la48','la49','la50',...
        'la51','la52','la53','la54','la55','la56','la57','la58','la59','la60' % SAC 50/50 hazard level
    'la21','la22','la23','la24','la25','la26','la27','la28','la29','la30',...
        'la31','la32','la33','la34','la35','la36','la37','la38','la39','la40' % SAC 2/50 hazard level
];

accel_scale_factor = [1 1];

num_records_per_set = 20;
DR_matrix_maxABS = zeros(num_records_per_set,s.num_stories);
for ilevel=1:2
    if ilevel==1
        file_prefix='EQrecords\la50in50\';
    else
        file_prefix='EQrecords\la2in50\';
    end
    for irecord=1:20
        disp(['level = ',num2str(ilevel),', irecord = ',num2str(irecord)]);
        file_name=strcat(file_prefix,record_array(ilevel,(irecord-1)*4+1:irecord*4));
        [EQ_g,dt,num_accel]=read_EQdata(file_name);
        EQ_inss = EQ_g*386.4;          %convert accel from 'g' to 'in/s/s'
        save_seismic(EQ_inss);
        create_DRAIN_INP_NTHA_pdelta(s,bw,cw,num_accel,dt,alpha_damp,beta_damp,...
                                     accel_scale_factor(ilevel));
        !DRAIN_2DX
        DR_history = get_response_history(s,num_accel);
        DR_matrix_maxABS(irecord,:) = max(abs(DR_history));
    end
    DRmedian = exp(mean(log(DR_matrix_maxABS)))
    dispersion = sqrt(sum((log(DR_matrix_maxABS)-log(ones(20,1)*DRmedian)).^2)/(20-1));
    DR84th = DRmedian.*exp(dispersion)
    DR95th = DRmedian.*exp(2*dispersion)
```

```
end


function [EQ_g,dt,num_accel]=read_EQdata(file_name)

fid = fopen(file_name,'r');
tmp = fgetl(fid);              % skip the title line
tmp = fgetl(fid);
num_accel = str2num(tmp(4:7)); % total number of discrete accel data
dt = str2num(tmp(12:22));      % time increment

data_per_line = 6;
%total number of data line, including the last incomplete line
num_complete_line = floor(num_accel/data_per_line);
num_incomplete_data = num_accel - num_complete_line*data_per_line;

EQ_g = zeros(num_accel,1);
for iline = 1:num_complete_line
    tmp = str2num(fgetl(fid));
    EQ_g(data_per_line*(iline-1)+1:data_per_line*iline)=tmp';
end

if num_incomplete_data>0
    tmp = str2num(fgetl(fid));
    EQ_g(num_accel-num_incomplete_data+1:num_accel)=tmp';
end
EQ_g=EQ_g/1000;                % now in terms of 'g'

fclose(fid);


function [] = create_DRAIN_INP_NTHA(s,bw,cw,num_accel,dt,alpha_damp,beta_damp,accel_scale_factor)

% create "DRAIN.INP" of the current SMRF 's' for DRAIN-2DX
% for Nonlinear Time History Analysis

< same coding as in 'create_DRAIN_INP_mode' not shown >

for i=1:s.num_stories
    fprintf(fid,'*GENDISP\n');
    fprintf(fid,'%10d%5d%10.5f\n',(s.num_bays+1)*i,    1,-1/s.story_height(i));
    fprintf(fid,'%10d%5d%10.5f\n',(s.num_bays+1)*(i+1),1, 1/s.story_height(i));
end

fprintf(fid,'*RESULTS\n');
fprintf(fid,'%1s%1s%7s%1d\n','G','D','',1);

fprintf(fid,'*ACCNREC\n');
fprintf(fid,'%1s%4s%15s%20s\n','','EAQK','seismic.txt','(f10.3)');
fprintf(fid,'%5d%5d%5d%5d%20s%10.4f%10.4f\n',num_accel,1,0,2,'',dt,0);
fprintf(fid,'*PARAMETERS\n');
fprintf(fid,'%2s%8s%10.5f%10.5f\n','VS','',alpha_damp,beta_damp);
fprintf(fid,'%2s%33s%5d%25s%5d%10.3f\n','OD','',1,'',0,0);
fprintf(fid,'%2s%5d%5d%5d%5d\n','DC',1,1,1,MAXEV);
fprintf(fid,'%2s%8.4f\n','DT',dt);
fprintf(fid,'*GRAV\n');
fprintf(fid,'%1s%5s%4s%10.1f\n','E','','VERT',1.0);
fprintf(fid,'*ACCN\n');
fprintf(fid,'%10.4f%5d%5d\n',dt*(num_accel-1),90000,1);
fprintf(fid,'%1d%5s%4s%10.3f\n',1,'','EAQK',accel_scale_factor);
fprintf(fid,'*STOP\n');
fclose(fid);


function DR_history = get_response_history(s,num_accel)

fid = fopen('NTHA.OUT','r');
DR_history = zeros(num_accel-1,s.num_stories);
num_all_nodes = s.nn+(s.num_stories+1);
total_lines_to_neglect = 19 + num_all_nodes + 8 + s.numcol*4 + 8 + (s.nel-s.numcol)*4 ...
                         + 2*(8+s.num_stories*4) + 32;
neglect_these_lines(fid,total_lines_to_neglect);
tmp=fscanf(fid,'%d%e%e%e%e%e%e',[2+s.num_stories,num_accel-1]);
tmp=tmp';
DR_history = tmp(:,3:7);
fclose(fid);
```

F.5 Quantification of merit objective functions of a valid SMRF design

```
function EQcost = compute_ SeismicDamageCost(p,s)

% calculate total lifetime seismic damage cost

LimitStateProb = zeros(7,1);

LSDRvector = [0.5  1.5  2.0  3.5  5.5  10.0]'/100;
LSPEvector = 1-0.5*(1+erf((log(LSDRvector)-p(1))/(p(2)*sqrt(2))));
designLife = 50;            % [year]
lambda = 0.05;              % monetary discount rate over years
Gt = [log(1-LSPEvector);0];
for iLS=2:7
    LimitStateProb(iLS) = Gt(iLS)-Gt(iLS-1);
end
LimitStateProb(1)=1-sum(LimitStateProb(2:7));
EQcost = sum(s.LScost.*LimitStateProb)*(1-exp(-lambda*designLife))/lambda;



function Icost = compute_InitialCost(s,bw,cw)

% include frame steel, metal decking, lightweight concrete, beam/column fireproofing.

total_construction_area = s.num_stories*(s.num_bays*s.bay_width/12)*(s.num_baysY*s.bay_width/12); % [ft^2]

lightConcreteCost = 1.63*total_construction_area; %$1.63/ft^2
metalDeckingCost  = 1.86*total_construction_area; %$1.86/ft^2
totalSteelCost = sum(s.totalSteelWeight)/2240*2375; % $2375/ton (1ton = 2240 lbs)

fireproofingArea_col=0;
fireproofingArea_beam=0;
dv=s.spdv(s.sps);
for imember = 1:s.numcol
    storyID=ceil(imember/(s.num_bays+1));
    d =cw(dv(imember),4);
    bf=cw(dv(imember),6);
    k1=cw(dv(imember),8);
    fireproofingArea_col=fireproofingArea_col+(2*d+4*bf-4*k1)*s.story_height(storyID);
end
for imember = s.numcol+1:s.nel
    d =bw(dv(imember),4);
    bf=bw(dv(imember),6);
    k1=bw(dv(imember),8);
    fireproofingArea_beam=fireproofingArea_beam+(2*d+3*bf-4*k1)*s.bay_width;
end

fireproofingCost=  1.46*(2*(1+s.FrameFactor_Y)*fireproofingArea_col +s.gravityFireProofArea_col )/12/12 ...
                 + 1.14*(2*(1+s.FrameFactor_Y)*fireproofingArea_beam+s.gravityFireProofArea_beam)/12/12;

Icost = lightConcreteCost + metalDeckingCost + fireproofingCost + totalSteelCost;
```

F.6 Implementation of multiobjective genetic algorithm

```
function [newgen,validobj,obj]=generate_initial_population(s,bw,cw,numobj)

% generate all combinations of structures with uniform beams and columns
% selected from the tables of standard W beams and columns.

% newgen(npop x ndv) w-section pointers for the design variables
% validobj(npop x 1) =1 if obj and objscale contain valid data, =0 otherwise
% obj(npop x numobj) objective values for members in newgen
% numobj(scalar) number of objective functions

numbw=size(bw,1);               % # of beam w-sections
numcw=size(cw,1);               % # column w-sections
numcoldv=sum(s.sptype==1);      % # of col dv's
newgen=zeros(numbw*numcw,s.numdv);

k=1;
for i=1:numbw
    for j=1:numcw
```

```
        newgen(k,:)=[j*ones(1,numcoldv),i*ones(1,s.numdv-numcoldv)];
        k=k+1;
    end
end
validobj=zeros(numbw*numcw,1);
obj=zeros(numbw*numcw,numobj);
```

```
function repro=reproduce(incpop,ndindex,tournsize,ranking,crowdist,valid,constrVIOLATE,infeasible_designs)

% Perform tournament selection over all members incpop times and
% appends all of ND set to end of reproduced population.
% Returns new population, repro, in the form of a vector
% containing pointers into newgen.
%
% The fitness measure is based on Deb's Nondominated Sorting with Crowded Distance
%
% incpop(1 x 1) number of members in newgen not in ndset
% ndindex(ndsize x 1) pointers into newgen of  nondominated valid members
% ranking(npop x 1) NSGAII ranking information (for valid designs only)
% crowdist(npop x 1) NSGAII ranking information (for valid designs only)
% constrVIOLATE(npop x 6) contraint violation values
% valid(number of valid memberss x 1) indices of members with validobj==1 and umember==1
% repro(incpop+nnd x 1) pointers into newgen of reproduced population

tic
nnd=size(ndindex,1);                    % number in ND set
repro=zeros(incpop+nnd,1);              % pointers into newgen of reproduced population
nvalid=size(valid,1);                   % number of valid members of population
ninfea=size(infeasible_designs,1);      % number of designs with moderate constraint violations

num_total = nvalid + ninfea;

%-------------------------------------------------------------------------------------------
%                 perform binary tournament selection, considering constraint violation
%-------------------------------------------------------------------------------------------

for i=1:incpop
    [ignore,p] = sort(rand(1,num_total));
    contestants=p(1:tournsize);
    if contestants(1)<=nvalid          % first guy is valid
        if contestants(2)<=nvalid      % second guy is valid
            current_rank_list = ranking(valid(contestants));
            min_rank_list = find(current_rank_list==min(current_rank_list));
            [maxval,maxindex] = max(crowdist(valid(contestants(min_rank_list))));
            repro(i)=valid(contestants(min_rank_list(maxindex)));
        else  % second guy is invalid
            repro(i)=valid(contestants(1)); % first guy wins
        end
    else      % first guy is invalid
        if contestants(2)<=nvalid           % second guy is valid
            repro(i)=valid(contestants(2)); % second guy wins
        else  % both are invalid
            contestants_infea=contestants-nvalid; % locations of infeasible designs
            [minval,minindex]=min(sum((constrVIOLATE(infeasible_designs(contestants_infea),:))'));
            repro(i)=infeasible_designs(contestants_infea(minindex));
        end
    end
end

%-------------------------------------------------------------------------------------------
%                 put whole ND set in repro (elitism)
%-------------------------------------------------------------------------------------------

repro(incpop+1:incpop+nnd)=ndindex;
```

```
function [offspring,newvalidobj,newobj,newSy,newDRvector,newmu]=...
    cross_mutate(s,bw,cw,newgen,repro,incpop,ndindex,probc,probm,numobj,obj,Sy,DRvector,mu,validobj)

% perform crossover and mutation and puts offspring
% in array offspring. Maintains objective values for reuse.
%
% newgen(npop x ndv) w-section pointers for the design variables
% repro(incpop+nnd x 1) pointers into newgen of reproduced population
% incpop(scalar) number of members in newgen not in ndset
% ndindex(ndsize x 1) pointers into newgen of  nondominated valid members
% probc(scalar) probablity of crossover
% probm(scalar) probability of mutation
```

```
% numobj(scalar) number of objective functions
% obj(npop x numobj) objective values for valid members in newgen
% Sy(npop x 1) system yield coefficient
% DRvector(npop x 5) drift ratio vectors at different hazard levels
% mu(npop x 5) system ductility values at different hazard levels
% newvalidobj(nrepro x 1) =1 if obj and objscale contain valid data, =0 otherwise
% newobj(nrepro x numobj) objective values for members in newgen

numbw=size(bw,1);
numcw=size(cw,1);
ncol=sum(s.sptype==1);
nbeam=s.numdv-ncol;
nrepro=size(repro,1);
offspring=zeros(nrepro,s.numdv);
newvalidobj=zeros(nrepro,1);
newobj=zeros(nrepro,numobj);

newSy=zeros(nrepro,1);

newDRvector=zeros(nrepro,5);
newmu=zeros(nrepro,5);

%-------------------------------------------------------------------------------------------
%                 perform crossover and mutation on initial incpop members
%-------------------------------------------------------------------------------------------

for i=1:incpop
    if rand(1)<probc/100
        mate=ceil(nrepro*rand(1));
        cloc=ceil((ncol-1)*rand(1));
        bloc=ceil((nbeam-1)*rand(1));
        % cross over once in column set and once in beam set
        offspring(i,:)=[newgen(repro(i),1:cloc),newgen(repro(mate),cloc+1:ncol),...
                newgen(repro(i),ncol+1:ncol+bloc),newgen(repro(mate),ncol+bloc+1:s.numdv)];
    else
        offspring(i,:)=newgen(repro(i),:); % no crossover
        feasibleINFO=validobj(repro(i));
        newvalidobj(i)=feasibleINFO;       % preserve whether a design from 'repro' is valid or not.
        if feasibleINFO==1 % a feasibel design
            newobj(i,:)=obj(repro(i),:); %recover objective values for valid designs only
            newSy(i)=Sy(repro(i));
            newDRvector(i,:)=DRvector(repro(i),:);
            newmu(i,:)=mu(repro(i),:);
        end
    end
    if rand(1)<probm/100
        if rand(1)<0.5 % mutate a column member
            offspring(i,ceil(ncol*rand(1)))=ceil(numcw*rand(1));
        else % mutate a beam member
            offspring(i,ncol+ceil(nbeam*rand(1)))=ceil(numbw*rand(1));
        end
        newvalidobj(i)=0; % new evaluation is needed.
    end
end

%-------------------------------------------------------------------------------------------
%                 append nondominated (ND) set in offspring
%-------------------------------------------------------------------------------------------

offspring(incpop+1:nrepro,:)=newgen(repro(incpop+1:nrepro),:);
newvalidobj(incpop+1:nrepro)=1; % preserve objective values
newobj(incpop+1:nrepro,:)=obj(repro(incpop+1:nrepro),:);
newvalidobj(incpop+1:nrepro)=1;

newSy(incpop+1:nrepro)=Sy(repro(incpop+1:nrepro));
newDRvector(incpop+1:nrepro,:)=DRvector(repro(incpop+1:nrepro),:);
newmu(incpop+1:nrepro,:)=mu(repro(incpop+1:nrepro),:);



function [ranking, crowdist, ndindex]=NSGAII(objscale,valid,obj)

% nondominated sorting with crowded distance (Deb 2001)

% given array objscale of multiple objectives (columns),
% which have been scaled 0 -> 1 for each member of population (rows),
% return ndindex, pointers into newgen, identifying the
% nondominated set (i.e., in the first rank).
%
% objscale(npop x numobj) each valid objective scaled 0 to 1
% valid(number of valid memberss x 1) indices of members with validobj==1 and umember==1
```

```
% obj(npop x numobj) objective values for valid members in newgen
% fitness(npop x 1) minimax fitness value
% ndindex(ndsize x 1) pointers into newgen of  nondominated valid members
% ranking(npop x 1) rank information for all (valid) designs
% crowdist(npop x 1) crowded distance measures for all (valid) designs

vscale=objscale(valid,:);       % focus on valid members only
[npop,numobj]=size(objscale);   % population size and number of objectives
nvalid=size(valid,1);           % size of valid population

ranking  = zeros(npop,1);
crowdist = zeros(npop,1);

%-------------------------------------------------------------------------------------------
%                 nondominated sorting
%-------------------------------------------------------------------------------------------

rank_valid = zeros(nvalid,1);   % initialize the ranking vector
dominance = repmat(struct('Sp',[],'np',0),nvalid,1); % initialization for all valid designs.

for p=1:nvalid
    for q=1:nvalid
        if p==q % one should not compete with itself
            continue;
        end
        if max(vscale(p,:)-vscale(q,:)) <= 0 % if p dominates q
            dominance(p).Sp = [dominance(p).Sp,q];
        elseif max(vscale(q,:)-vscale(p,:)) <= 0  % if q dominates p
            dominance(p).np = dominance(p).np +1; % increment the dominance counter of p
        end
    end
    if dominance(p).np == 0 % no other designs dominate p
        rank_valid(p)=1;    % so p belongs to the (first) nondominated set
    end
end

irank=1;  % initialize the front counter
current_front = find(rank_valid==1);
ndindex = valid(current_front); % the first ND set used as elitists for the next generation

while ~isempty(current_front)
    irank = irank + 1; % looking for the next front
    Q = []; % used to store the members of the next front
    num_in_front = length(current_front);
    for p = 1:num_in_front
        each_p = current_front(p);
        for q = 1:length(dominance(each_p).Sp)
            each_q = dominance(each_p).Sp(q);
            dominance(each_q).np = dominance(each_q).np - 1;
            if dominance(each_q).np == 0 % q belongs to the next front
               rank_valid (each_q) = irank;
               Q = [Q,each_q];
            end
        end
    end
    current_front = Q;
end

%-------------------------------------------------------------------------------------------
%                 crowded distance assignment
%-------------------------------------------------------------------------------------------

crowd_valid = zeros(nvalid,1); % initialize the crowded distance vector
for irank = 1:max(rank_valid)  % calculate crowded distances for all solutions in the same rank
    current_front = find(rank_valid==irank);
    num_in_front = length(current_front); % # of designs in the current rank
    if num_in_front <= 2 % i.e., only one/two extreme designs available
        crowd_valid(current_front) = 1000; % assign an arbitrarily large value
    else
        current_obj = vscale(current_front,:);
        [tmp,index] = sort(current_obj); % sort each objective values from min to max
        for iobj = 1:numobj
            maxiobj = tmp(end,iobj);
            miniobj = tmp(  1,iobj);
            crowd_valid(current_front(index(end,iobj))) = 1000; %assign large value to boundary designs
            crowd_valid(current_front(index(  1,iobj))) = 1000;
            if maxiobj ~= miniobj % if designs have different objective values
                for idesign = 2:num_in_front-1 % for all designs except boundary designs
                    inter_design  = current_front(index(  idesign,iobj));
                    before_design = current_front(index(idesign-1,iobj)); % two neighboring designs
                    after_design  = current_front(index(idesign+1,iobj));
```

```
                    crowd_valid(inter_design) = crowd_valid(inter_design) + (vscale(after_design,iobj)...
                                                - vscale(before_design,iobj))/(maxiobj - miniobj);
                end
            end
        end
    end
end

%-------------------------------------------------------------------------------------------
%                 expand from nvalid to npop designs
%-------------------------------------------------------------------------------------------
ranking(valid)  = rank_valid;
crowdist(valid) = crowd_valid;
```

F.7 Main programs for multiobjective design optimization of SMRF

```
% main_multiobj

% This is the main program to perform the multibjective design optimziation of steel SMRF structures

load beam_W_sec
load col_W_sec

s=create_SMRF;
s_modify_2_5s4b_splice;
s=compute_general_data(s,bw,cw);
s=modify_s_Live(s);

rand('seed',0);

incpop=1000; % population size in addition to ND set
tournsize=2; % binary tournament selection
probc=50;    % crossover probability in percentage
probm=30;    % mutation probability in percentage
numobj=3;    % number of objectives

relaxed_constr_limits = [1.5,1.5,1.5,1.5,1.5,1.5];
numconstr = length(relaxed_constr_limits);

filename_NDnum=strcat(['NDset\ND042703\ND_num.m']);
fid = fopen(filename_NDnum,'a');
fprintf(fid,'\n%6s  %6s  %% started at %20s\n','','',datestr(now));
fclose(fid);

gen=0;

[newgen,validobj,obj]=generate_initial_population(s,bw,cw,numobj);

num_gen0 = size(newgen,1);
Sy = zeros(num_gen0,1);
maxDRvector = zeros(num_gen0,5);
mu = zeros(num_gen0,5);

ndindex = [];

tic;
[validobj,obj,objscale,valid,Sy,maxDRvector,mu,constrVIOLATE,infeasible_designs]=...
    objective_evaluation(s,bw,cw,newgen,validobj,obj,numobj,gen,Sy,maxDRvector,mu,...
                         ndindex,relaxed_constr_limits,numconstr);
[ranking, crowdist, ndindex]=NSGAII(objscale,valid,obj);
time_gen0=toc;

save NDset\ND042703\gen0\newgen.m newgen -ascii
save NDset\ND042703\gen0\validobj.m validobj -ascii
save NDset\ND042703\gen0\obj.m obj -ascii
save NDset\ND042703\gen0\objscale.m objscale -ascii
save NDset\ND042703\gen0\valid.m valid -ascii
save NDset\ND042703\gen0\Sy.m Sy -ascii
save NDset\ND042703\gen0\maxDRvector.m maxDRvector -ascii
save NDset\ND042703\gen0\mu.m mu -ascii
save NDset\ND042703\gen0\ranking.m ranking -ascii
save NDset\ND042703\gen0\crowdist.m crowdist -ascii
save NDset\ND042703\gen0\ndindex.m ndindex -ascii
save NDset\ND042703\gen0\gen.m gen -ascii
save NDset\ND042703\gen0\constrVIOLATE.m constrVIOLATE -ascii
```

```
save NDset\ND042703\gen0\infeasible_designs.m infeasible_designs -ascii

save NDset\ND042703\last_run\newgen.m newgen -ascii
save NDset\ND042703\last_run\validobj.m validobj -ascii
save NDset\ND042703\last_run\obj.m obj -ascii
save NDset\ND042703\last_run\objscale.m objscale -ascii
save NDset\ND042703\last_run\valid.m valid -ascii
save NDset\ND042703\last_run\Sy.m Sy -ascii
save NDset\ND042703\last_run\maxDRvector.m maxDRvector -ascii
save NDset\ND042703\last_run\mu.m mu250 -ascii
save NDset\ND042703\last_run\ranking.m ranking -ascii
save NDset\ND042703\last_run\crowdist.m crowdist -ascii
save NDset\ND042703\last_run\ndindex.m ndindex -ascii
save NDset\ND042703\last_run\gen.m gen -ascii
save NDset\ND042703\last_run\constrVIOLATE.m constrVIOLATE -ascii
save NDset\ND042703\last_run\infeasible_designs.m infeasible_designs -ascii

oldndset=newgen(ndindex,:); % save ND set to compare to next one
oldndobj=obj(ndindex,:);
oldndSy=Sy(ndindex,:);
oldndmaxDRvector=maxDRvector(ndindex,:);
oldndmu=mu(ndindex,:);

disp(['nondominated set size = ',num2str(size(ndindex,1)),' gen=0']);

NDset=[oldndset,oldndobj,oldndSy,oldndmu,oldndmaxDRvector*100];
filename_NDset=strcat(['NDset\ND042703\ND_set_gen_',num2str(0)',' .m']);
fid = fopen(filename_NDset,'w');
lengthNDset = length(NDset(:,1));
for i=1:lengthNDset
    fprintf(fid,'%4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d
    %4d %4d %15.2f  %15.2f  %4d  %10.5f\n',NDset(i,:));
end
fclose(fid);

fid = fopen(filename_NDnum,'a');
fprintf(fid,'\n%6d  %6d  %% generation #%3d, finished at %20s\n',lengthNDset,lengthNDset,gen,datestr(now));
fclose(fid);

while 1==1

    gen=gen+1;

    rand('seed',gen);

    repro=reproduce(incpop,ndindex,tournsize,ranking,crowdist,valid,constrVIOLATE,infeasible_designs);

    [offspring,newvalidobj,newobj,newSy,newmaxDRvector,newmu]=...
        cross_mutate(s,bw,cw,newgen,repro,incpop,ndindex,probc,probm,numobj,obj,Sy,maxDRvector,mu,validobj);

    newgen=offspring;
    validobj=newvalidobj;
    obj=newobj;
    Sy=newSy;
    maxDRvector=newmaxDRvector;
    mu=newmu;

    clear offspring newvalidobj newobj newSy newOSfactor newDDR newmaxDRvector newmu250;
    pack;

    tic;

    [validobj,obj,objscale,valid,Sy,maxDRvector,mu,constrVIOLATE,infeasible_designs]=...
        objective_evaluation(s,bw,cw,newgen,validobj,obj,numobj,gen,Sy,maxDRvector,mu,...
                             ndindex,relaxed_constr_limits,numconstr);

    [ranking, crowdist, ndindex]=NSGAII(objscale,valid,obj);

    time_current_gen=toc;

    save NDset\ND042703\last_run\newgen.m newgen -ascii
    save NDset\ND042703\last_run\validobj.m validobj -ascii
    save NDset\ND042703\last_run\obj.m obj -ascii
    save NDset\ND042703\last_run\objscale.m objscale -ascii
    save NDset\ND042703\last_run\valid.m valid -ascii
    save NDset\ND042703\last_run\Sy.m Sy -ascii
    save NDset\ND042703\last_run\maxDRvector.m maxDRvector -ascii
    save NDset\ND042703\last_run\mu.m mu -ascii
    save NDset\ND042703\last_run\ranking.m ranking -ascii
    save NDset\ND042703\last_run\crowdist.m crowdist -ascii
    save NDset\ND042703\last_run\ndindex.m ndindex -ascii
```

```
    save NDset\ND042703\last_run\gen.m gen -ascii %added Mar-26-03
    save NDset\ND042703\last_run\constrVIOLATE.m constrVIOLATE -ascii
    save NDset\ND042703\last_run\infeasible_designs.m infeasible_designs -ascii

    tic;
    [c,ia,ib]=intersect(oldndset,newgen(ndindex,:),'rows'); % intersect old and new ND set
    clear c ib;
    numnewnd=size(ndindex,1)-size(ia,1);                    % number of new ND members

    oldndset=newgen(ndindex,:); % save ND set to compare to next one
    oldndobj=obj(ndindex,:);
    oldndSy=Sy(ndindex,:);
    oldndmaxDRvector=maxDRvector(ndindex,:);
    oldndmu=mu(ndindex,:);

    disp(sprintf('%5d number of new ND members',numnewnd));

    NDset=[oldndset,oldndobj,oldndSy,oldndmu,oldndmaxDRvector*100];

    lengthNDset = length(NDset(:,1));

    if mod(gen,10)==0
        filename_NDset=strcat(['NDset\ND042703\ND_set_gen_',num2str(gen),'.m']);
        fid = fopen(filename_NDset,'w');
        for i=1:lengthNDset
             fprintf(fid,'%4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d
             %4d %4d  %4d %4d %15.2f  %15.2f  %4d  %10.5f\n',NDset(i,:));
        end
        fclose(fid);
    end

    fid = fopen(filename_NDnum,'a');
    fprintf(fid,'\n%6d  %6d  %% generation #%3d, finished at %20s\n',lengthNDset,numnewnd,gen,datestr(now));
    fclose(fid);

end
```

```
function [validobj,obj,objscale,valid,Sy,maxDRvector,mu,constrVIOLATE,infeasible_designs]=...
                 objective_evaluation(s,bw,cw,newgen,validobj,obj,numobj,gen,Sy,...
                 maxDRvector,mu,ndindex,relaxed_constr_limits,numconstr)

% Compute merit objectives for each member in newgen (by rows) that does not have validobj=1.

% Return obj with three objectives:
% 1) initial cost ($), 2) lifetime seismic damage cost ($), 3) number of different steel section types

% for each member, by rows, with failed analyses identified in validobj=0.
% Checks for duplicates and does not evaluate them (umembers =0 for duplicates).

numbw=size(bw,1); % number of beam w-sections
numcw=size(cw,1); % number of column w-sections

[tmp,IBEAM,ICOLUMN]=intersect(bw(:,1:2),cw(:,1:2),'rows');

npop=size(newgen,1); % size of current population
numnd=length(ndindex);

numdvb=s.col_dv_per_floor;
numcoldv=sum(s.sptype==1);

constrVIOLATE = zeros(npop,numconstr);

[b,ugen,n]=unique(newgen,'rows'); % identify duplicates, ugen points to unique ones in newgen
umember=zeros(npop,1);
umember(ugen)=1;                  % set flag for unique members

for i=1:npop

    if validobj(i)==0 & umember(i)==1

        s.spdv=newgen(i,:);

%-----------------------------------------------------------------------------------------
%                 check if the lower column is weaker than adjacent upper column
%-----------------------------------------------------------------------------------------

        flag_ColSize=0;
        node_per_half_floor = floor(s.num_bays/2+1);
```

```
for istory = 2:s.num_stories
    if flag_ColSize==0
        for inode = 1:node_per_half_floor
            nodeID = (istory-1)*(s.num_bays+1)+inode;
            lowerColID = s.colinc(nodeID,1);
            lowerColDV = s.spdv(s.sps(lowerColID));
            Ix_lowerCol = cw(lowerColDV,23);
            upperColID = s.colinc(nodeID,2);
            upperColDV = s.spdv(s.sps(upperColID));
            Ix_upperCol = cw(upperColDV,23);
            if Ix_lowerCol < Ix_upperCol
                flag_ColSize=1;
                break;
            end
        end
    else
        break;
    end
end
if flag_ColSize==1
    continue;
end

s = compute_bldg_weight(s,bw,cw);
create_DRAIN_INP_mode(s,bw,cw);
!DRAIN_2DX
[T1,flagMODE]=get_DRAIN_mode;
if flagMODE==1
    continue;
end
[s,pushoverProfile]=modify_s_DWE(s,bw,cw,T1);
[r,flagR] =LRFD_analysis(s,bw,cw);
if flagR==1
    continue;
end
[Kcol,flagG] = Kfactor05(s,bw,cw);
if flagG==1
    continue;
end

[flagBehavior,constrVIOLATE(i,:)]=LRFDcheck(s,r,bw,cw,Kcol,relaxed_constr_limits);
if flagBehavior==1
    continue;
elseif flagBehavior==2
    validobj(i)=2;
    continue;
end
PushLimit=30.;
PushNumber=1;
PushStep=1;
PushIncrement=15.;
flagPushNumber=0;
flagPush=1;
while flagPush
    create_DRAIN_INP_Ppushover(s,bw,cw,pushoverProfile,PushLimit);
    !DRAIN_2DX
    [PHI_yield,Kpre,Kpost,alpha,Dy,flagPUSH,flagALPHA,flagBILINEAR,Sy(i),DeflectProfile]...
          = get_DRAIN_pushover(s,PushLimit,PushNumber,PushStep);
    if flagPUSH==1
        break;
    end
    if flagALPHA==1
        break;
    end
    if alpha>0.1
        PushLimit=PushLimit+PushIncrement;
        PushNumber=PushNumber+1;
        if PushNumber>5
            flagPushLimit=1;
            break;
        end
    else
        flagPush=0;
    end
end

if flagPush==0
    [p,flagLN,maxDRvector(i,:),mu(i,:)] = equivalent_SDOF(s,PHI_yield,Kpre,Kpost,Dy,alpha,p...
                                                          ushoverProfile,DeflectProfile,PushStep);
    if flagLN==1
        continue;
```

```
            end
        else
            continue;
        end

        obj(i,1)=compute_InitialCost(s,bw,cw) + 500*5/2240*2375;

        obj(i,2)=compute_ SeismicDamageCost(p,s);

        tmp_newgen = newgen(i,:);
        for idv=numcoldv+1:s.numdv
            if ismember(tmp_newgen(idv),IBEAM) & ismember(ICOLUMN(find(tmp_newgen(idv)==IBEAM)),...
                                                                    tmp_newgen(1:numcoldv))
                tmp_newgen(idv)=0;
            end
        end
        obj(i,3)=sum(unique(tmp_newgen(1:numcoldv))~=0)+sum(unique(tmp_newgen(numcoldv+1:s.numdv))~=0);

        validobj(i)=1;

    end
end

%scale the fitnesses between 0 and 1 in each column
valid=find(validobj==1 & umember==1);
nvalid=size(valid,1);
mins=min(obj(valid,:));
objscale=zeros(npop,numobj);
objscale(valid,:)=obj(valid,:)-mins(ones(nvalid,1),:);
maxs=max(objscale(valid,:));

warning off;
objscale(valid,:)=objscale(valid,:)./maxs(ones(nvalid,1),:);

objscale(find(isnan(objscale)))=1; %if all of one objective are same, removes NaNs

infeasible_designs=find(validobj==2 & umember==1);
```

Example DRAIN-2DX input file

The following DRAIN.INP is generated using the SMRF of eight different section types provided in Table 7.8.

```
*STARTXX
  MULTIOBJ         0 0 1 1                     GENERATED AT 26-Jul-2003 19:34:19
*NODECOORDS
C         1      0.00      0.00
C         2    300.00      0.00
C         3    600.00      0.00
C         4    900.00      0.00
C         5   1200.00      0.00
C         6      0.00    180.00
C         7    300.00    180.00
C         8    600.00    180.00
C         9    900.00    180.00
C        10   1200.00    180.00
C        26      0.00    804.00
C        27    300.00    804.00
C        28    600.00    804.00
C        29    900.00    804.00
C        30   1200.00    804.00
L         6        26         5
L         7        27         5
L         8        28         5
L         9        29         5
L        10        30         5
C        31   1500.00      0.00
C        32   1500.00    180.00
C        36   1500.00    804.00
L        32        36         1
*RESTRAINTS
S 111          1         5
S 110         31
*SLAVING
S 100          6         7        10
S 100         11        12        15
```

```
S 100        16        17        20
S 100        21        22        25
S 100        26        27        30
S 100         6        32
S 100        11        33
S 100        16        34
S 100        21        35
S 100        26        36
*MASSES
S 100    104.41         6         6                              386.4      1.00
S 100    208.81         7         9                              386.4      1.00
S 100    104.41        10        10                              386.4      1.00
S 100     98.75        11        11                              386.4      1.00
S 100    197.50        12        14                              386.4      1.00
S 100     98.75        15        15                              386.4      1.00
S 100     97.95        16        16                              386.4      1.00
S 100    195.90        17        19                              386.4      1.00
S 100     97.95        20        20                              386.4      1.00
S 100     97.35        21        21                              386.4      1.00
S 100    194.70        22        24                              386.4      1.00
S 100     97.35        25        25                              386.4      1.00
S 100     82.63        26        26                              386.4      1.00
S 100    165.26        27        29                              386.4      1.00
S 100     82.63        30        30                              386.4      1.00
*ELEMENTGROUP
    2    1    1           1.00                                  COLUMNS
    6    0    6
!STIFFNESS TYPES
    1   29000.0      0.03      68.5   3010.00  4.0  4.0  2.0
    2   29000.0      0.03     109.0   5440.00  4.0  4.0  2.0
    3   29000.0      0.03     109.0   5440.00  4.0  4.0  2.0
    4   29000.0      0.03      42.7   1710.00  4.0  4.0  2.0
    5   29000.0      0.03     101.0   4900.00  4.0  4.0  2.0
    6   29000.0      0.03     109.0   5440.00  4.0  4.0  2.0
!YIELD SURFACE TYPES
    1    2   25113.6   25113.6    3945.6    3945.6  1.00.125  1.00.125
    2    2   42393.6   42393.6    6278.4    6278.4  1.00.125  1.00.125
    3    2   42393.6   42393.6    6278.4    6278.4  1.00.125  1.00.125
    4    2   14976.0   14976.0    2459.5    2459.5  1.00.125  1.00.125
    5    2   38707.2   38707.2    5817.6    5817.6  1.00.125  1.00.125
    6    2   42393.6   42393.6    6278.4    6278.4  1.00.125  1.00.125
!ELEMENT GENERATION COMMANDS
    1         1         6              1         1    1
    2         2         7              2         2    2
    3         3         8              3         3    3
    4         4         9              2         2    2
    5         5        10              1         1    1
    6         6        11              1         1    1
    7         7        12              2         2    2
    8         8        13              3         3    3
    9         9        14              2         2    2
   10        10        15              1         1    1
   11        11        16              4         4    4
   12        12        17              5         5    5
   13        13        18              6         6    6
   14        14        19              5         5    5
   15        15        20              4         4    4
   16        16        21              4         4    4
   17        17        22              5         5    5
   18        18        23              6         6    6
   19        19        24              5         5    5
   20        20        25              4         4    4
   21        21        26              4         4    4
   22        22        27              5         5    5
   23        23        28              6         6    6
   24        24        29              5         5    5
   25        25        30              4         4    4
*ELEMENTGROUP
    2    1    0           1.00                              BEAMS/GIRDERS
    5    0    5
!STIFFNESS TYPES
    1   29000.0      0.03      39.7    7800.0  4.0  4.0  2.0
    2   29000.0      0.03      39.7    7800.0  4.0  4.0  2.0
    3   29000.0      0.03      34.7    5900.0  4.0  4.0  2.0
    4   29000.0      0.03      27.7    3270.0  4.0  4.0  2.0
    5   29000.0      0.03      13.0     843.0  4.0  4.0  2.0
!YIELD SURFACE TYPES
    1    1   25042.8   25042.8
    2    1   25042.8   25042.8
    3    1   20418.0   20418.0
    4    1   13677.6   13677.6
```

```
    5    1    4693.7    4693.7
!ELEMENT GENERATION COMMANDS
    1         6         7              1         1    1
    2         7         8              1         1    1
    3         8         9              1         1    1
    4         9        10              1         1    1
    5        11        12              2         2    2
    6        12        13              2         2    2
    7        13        14              2         2    2
    8        14        15              2         2    2
    9        16        17              3         3    3
   10        17        18              3         3    3
   11        18        19              3         3    3
   12        19        20              3         3    3
   13        21        22              4         4    4
   14        22        23              4         4    4
   15        23        24              4         4    4
   16        24        25              4         4    4
   17        26        27              5         5    5
   18        27        28              5         5    5
   19        28        29              5         5    5
   20        29        30              5         5    5
*ELEMENTGROUP
    2    1    1           1.00                            RIGID LINKS
    1    0    1
!STIFFNESS TYPES
    1   29000.0      0.03 1000000.01000000.00  0.0  0.0  0.0
!YIELD SURFACE TYPES
    1    2 1000000.0 1000000.0 1000000.0 1000000.0  1.00.125  1.00.125
!ELEMENT GENERATION COMMANDS
    1        10        32              1         1    1
    2        15        33              1         1    1
    3        20        34              1         1    1
    4        25        35              1         1    1
    5        30        36              1         1    1
*ELEMENTGROUP
    2    1    1           1.00                            DUMMY COLUMN
    1    0    1
!STIFFNESS TYPES
    1   29000.0      0.03 1000000.0      0.10  4.0  4.0  2.0
!YIELD SURFACE TYPES
    1    2       1.0       1.0 1000000.0 1000000.0  1.00.125  1.00.125
!ELEMENT GENERATION COMMANDS
    1        31        32              1         1    1
    2        32        33              1         1    1
    3        33        34              1         1    1
    4        34        35              1         1    1
    5        35        36              1         1    1
*GENDISP
         5    1  -0.00556
        10    1   0.00556
*GENDISP
        10    1  -0.00641
        15    1   0.00641
*GENDISP
        15    1  -0.00641
        20    1   0.00641
*GENDISP
        20    1  -0.00641
        25    1   0.00641
*GENDISP
        25    1  -0.00641
        30    1   0.00641
*RESULTS
GD       1
*ELEMLOAD
 VERT                                       UNFACTORED DL&LL FROM TRIBUTARY AREAS
G   2    5
    1    0       1.0       0.0      24.1    1206.3       0.0      24.1   -1206.3
    2    0       1.0       0.0      21.5    1075.7       0.0      21.5   -1075.7
    3    0       1.0       0.0      21.1    1055.6       0.0      21.1   -1055.6
    4    0       1.0       0.0      20.8    1040.6       0.0      20.8   -1040.6
    5    0       1.0       0.0      17.3     864.9       0.0      17.3    -864.9
    1    4         1      1.00
    5    8         2      1.00
    9   12         3      1.00
   13   16         4      1.00
   17   20         5      1.00

*NODALOAD
 INTR                                      UNFACTORED DL&LL FROM INTERIOR AREAS
```

```
S   0.0000 -817.9688    0.0000        32
S   0.0000 -817.9688    0.0000        33
S   0.0000 -817.9688    0.0000        34
S   0.0000 -817.9688    0.0000        35
S   0.0000 -665.6250    0.0000        36
*ACCNREC
 EAQK    seismic.txt             (f10.3)
 2686    1    0    2                        0.0100    0.0000
*PARAMETERS
VS           0.38086   0.00216
OD                                    1                              0     0.000
DC    1    1    1  100
DT  0.0100
*GRAV
E     VERT       1.0
*ACCN
   26.850090000    1
1     EAQK     1.000
*STOP
```