Technical University of Denmark

41525 Finite Element Methods

Matlab Codes Assignment 1

Team 26 - Lion

Fabio Rasera, s216375

Niccolo Andreetta, s221920

September 27, 2022

Contents

1 Exercise 1

1.1 Input file

This is the file used to load the structure

```
% Created with:       FlExtract v1.13
% Element type:       truss
% Number of nodes:    19
% Number of elements: 34

clear all;

% Node coordinates: x, y
X = [
0    0
0    0.5
0.5  0
0.5  0.5
1    0
1    0.5
1.5  0
1.5  0.5
1.5  1
2    0
2    0.5
2.5  0
2.5  0.5
3    0
3    0.5
3.5  0
3.5  0.5
4    0
4    0.5
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2    1    1
3    1    1
4    2    1
3    2    1
4    3    1
5    3    1
6    4    1
5    4    1
6    5    1
7    5    1
8    6    1
7    6    1
8    7    1
10   7    1
11   8    1
10   8    1
11   9    1
11   10   1
12   10   1
13   11   1
12   11   1
13   12   1
14   12   1
15   13   1
14   13   1
```

```
15   14   1
16   14   1
17   15   1
16   15   1
17   16   1
18   16   1
19   17   1
18   17   1
19   18   1
];
% Element properties: Young's modulus, area
mprop = [
7e+010    0.0002
];
% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
2    1    0
2    2    0
9    1    0
9    2    0
];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [
18   2    -15000
];
% Control parameters
plotdof = 2;
```

1.2 File performing the thrust analysis

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Basis truss program                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
%example1                % Input file
TrussExercise1

neqn = size(X,1)*size(X,2);           % Number of equations
ne = size(IX,1);                      % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%
Kmatr=sparse(neqn,neqn);                  % Stiffness matrix
P=zeros(neqn,1);                          % Force vector
D=zeros(neqn,1);                          % Displacement vector
R=zeros(neqn,1);                          % Residual vector
strain=zeros(ne,1);                       % Element strain vector
stress=zeros(ne,1);                       % Element stress vector

%--- Calculate displacements ---------------------------------------------%
[P]=buildload(X,IX,ne,P,loads,mprop);         % Build global load vector

[Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr);      % Build global stiffness matrix

[Kmatr,Pmatr]=enforce(Kmatr,P,bound);         % Enforce boundary conditions

D = Kmatr \ Pmatr;                            % Solve system of equations

[strain,stress]=recover(mprop,X,IX,D,ne,strain,stress); % Calculate element
                                                        % stress and strain

% Calculate element axial forces and the support reactions
[strain,stress,N,R]=recover2(mprop,X,IX,D,ne,strain,stress,P);

%--- Print the results on the command window -----------------------------%
% External matrix
disp('External forces applied (N)')
P'

% Stress
disp('Stress on the bars (MPa)')
stress'

% Displacement
disp('Displacement (m)')
D'

% Strain
disp('Strain of the bars')
strain'

% Forces on the bars
```

```
disp('Internal forces on the bar (N)')
N

% Support reaction
disp('Support reactions forces (N)')
R'

%--- Plot results --------------------------------------------------------%
PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)         % Plot structure

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_e = E * A * L0 * B0 * B0'; % 4x4 matrix

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build K by summing k_e
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_e(ii, jj);
```

```
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress]=recover(mprop,X,IX,D,ne,strain,stress);

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces

% allocate memory for stress and strain vectors
strain = zeros(ne, 1);
stress = zeros(ne, 1);

for e=1:ne
  d = zeros(4, 1); % allocate memory for element stiffness matrix

 [edof] = build_edof(IX, e); % index for buildg K

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  strain(e) = B0' * d;
  stress(e) = strain(e) * E;
```

```matlab
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress, N, R]=recover2(mprop,X,IX,D,ne,strain,stress,P);

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
strain = zeros(ne, 1);
stress = zeros(ne, 1);
B0_sum = zeros(2*size(X,1), 1);
for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  strain(e) = B0' * d;
  stress(e) = strain(e) * E;
  N(e) = stress(e) * A;

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      B0_sum(edof(jj)) = B0_sum(edof(jj)) + B0(jj)*N(e)*L0;
  end

end

% compute the support reactions (N)
R = B0_sum - P; % 2nnx1 (nn is node number)



return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure
```

```
h1=0;h2=0;

% Plotting Un-Deformed and Deformed Structure
figure_truss_structure= figure('Position', get(0, 'Screensize'));
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
fake_zero = max(abs(stress)) / 1e5; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > fake_zero  % tension
      col = colors(1);
    elseif stress(e) < - fake_zero  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    if e == 34 % unloaded
      xx1=xx;
      yy1 = yy;
      col1=col;
    elseif e == 4 % compression
      xx4=xx;
      yy4 = yy;
      col4=col;
    elseif e == 5 % tensile
      xx3=xx;
      yy3 = yy;
      col3=col;
    end

    if e ~= 34
      h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
    else
      h2=plot(xx,yy, 'k','LineWidth',3.5); % Deformed structure
    end
end
plotsupports
plotloads

h_u=plot(xx1,yy1,col1,'LineWidth',3.5);
h_c=plot(xx4,yy4,col4,'LineWidth',3.5);
h_t=plot(xx3,yy3,col3,'LineWidth',3.5);

legend([h1 h2 h_u h_c h_t],{'Undeformed state','Deformed ...
state','Unloaded','Compressed','Tensioned'})

axis equal;
hold off
text_size = 20;
```

```
title('Truss structure in deformed and undeformed state')
xlabel('x [m]')
ylabel('y [m]')
text(0,0.6,'A','FontSize', text_size)
text(0.5,0.6,'B','FontSize', text_size)
text(0.5,-0.1,'E','FontSize', text_size)
text(1,-0.1,'F','FontSize', text_size)
text(2.05,0.6,'C','FontSize', text_size)
text(1.6,1,'D','FontSize', text_size)
text(3.85,-0.15,'G','FontSize', text_size)
text(0,-0.1,'H','FontSize', text_size)
text(4.05,0.5,'I','FontSize', text_size)
text(1,0.6,'J','FontSize', text_size)

set(gca,'FontAngle','oblique','FontSize', text_size)
saveas(figure_truss_structure, 'C:\Users\Niccolo\Documents\UNIVERSITA\ ...
5ANNO\FEM\assignment1\truss.png','png');

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e);

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e);
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return
```

2 Exercise 2

2.1 Input file

```
% Created with:       FlExtract v1.12
% Element type:       truss
% Number of nodes:    55
% Number of elements: 124

clear all
close all
clc

%nincr=100;
incr_vector=[100];
%incr_vector = [5:5:100];
eSTOP=10^(-8);
i_max=100;
Pfinal=80;


% Node coordinates: x, y
X = [
0     0
0     4
5.55556    0
5.55556    4
5.55556    8
11.1111    4
11.1111    8
11.1111    12
16.6667    8
16.6667    12
16.6667    16
22.2222    12
22.2222    16
22.2222    20
27.7778    16
27.7778    20
27.7778    24
33.3333    20
33.3333    24
33.3333    28
38.8889    24
38.8889    28
38.8889    32
44.4444    28
44.4444    32
44.4444    36
50    32
50    36
50    40
55.5556    28
55.5556    32
55.5556    36
61.1111    24
61.1111    28
61.1111    32
66.6667    20
66.6667    24
66.6667    28
```

```
72.2222   16
72.2222   20
72.2222   24
77.7778   12
77.7778   16
77.7778   20
83.3333   8
83.3333   12
83.3333   16
88.8889   4
88.8889   8
88.8889   12
94.4444   0
94.4444   4
94.4444   8
100  0
100  4
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2    1    1
3    1    1
5    2    1
4    2    1
3    2    1
4    3    1
6    3    1
5    4    1
6    4    1
8    5    1
7    5    1
6    5    1
7    6    1
9    6    1
8    7    1
9    7    1
11   8    1
10   8    1
9    8    1
10   9    1
12   9    1
11   10   1
12   10   1
14   11   1
13   11   1
12   11   1
13   12   1
15   12   1
14   13   1
15   13   1
17   14   1
16   14   1
15   14   1
16   15   1
18   15   1
17   16   1
18   16   1
20   17   1
19   17   1
18   17   1
19   18   1
```

21	18	1
20	19	1
21	19	1
23	20	1
22	20	1
21	20	1
22	21	1
24	21	1
23	22	1
24	22	1
26	23	1
25	23	1
24	23	1
25	24	1
27	24	1
26	25	1
27	25	1
29	26	1
28	26	1
27	26	1
28	27	1
32	27	1
31	27	1
30	27	1
29	28	1
32	28	1
32	29	1
31	30	1
35	30	1
34	30	1
33	30	1
32	31	1
35	31	1
35	32	1
34	33	1
38	33	1
37	33	1
36	33	1
35	34	1
38	34	1
38	35	1
37	36	1
41	36	1
40	36	1
39	36	1
38	37	1
41	37	1
41	38	1
40	39	1
44	39	1
43	39	1
42	39	1
41	40	1
44	40	1
44	41	1
43	42	1
47	42	1
46	42	1
45	42	1
44	43	1
47	43	1

```
47   44   1
46   45   1
50   45   1
49   45   1
48   45   1
47   46   1
50   46   1
50   47   1
49   48   1
53   48   1
52   48   1
51   48   1
50   49   1
53   49   1
53   50   1
52   51   1
55   51   1
54   51   1
53   52   1
55   52   1
55   53   1
55   54   1 ];

% Element properties: [ E A c1 c2 c3 c4],
mprop = [ 1.0 1.0 1.2 5 0.2 50 ]; % parameters for ex1 to 3
%mprop = [ 1.0 1.0 0 1 0.2 200 ]; % parameters for ex 4

% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
1    1    0
3    1    0
1    2    0
3    2    0
51   2    0
54   2    0];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [ 24   2   Pfinal
          30   2  -Pfinal ];

% Control parameters
plotdof = 48;
```

2.2 Euler method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Basic Euler method                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
TrussExercise2_2022              % Input file

neqn = size(X,1)*size(X,2);          % Number of equations
ne = size(IX,1);                     % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%
K=sparse(neqn,neqn);                 % Stiffness matrix
P_final=zeros(neqn,1);                   % Force vector
R=zeros(neqn,1);                         % Residual vector
strain=zeros(ne,1);                      % Element strain vector
stress=zeros(ne,1);                      % Element stress vector
P_plot=zeros(max(incr_vector), size(incr_vector, 2));
D_plot=zeros(max(incr_vector), size(incr_vector, 2));

%--- Calculate displacements ---------------------------------------------%

[P_final] = buildload(X,IX,ne,P_final,loads,mprop); % vector of the external loads

rubber_param = [mprop(3) mprop(4) mprop(5) mprop(6)]; % coefficients for the nonlinear material
residual_norm = zeros(1, size(incr_vector, 2));

for j = 1:size(incr_vector, 2)  % cycle over the different # of load incr

  %  number of increments
  nincr = incr_vector(j);

  % load increment
  delta_P = P_final / nincr;

  % Initialize arrays
  P=zeros(neqn,1);                          % Force vector
  D=zeros(neqn,1);                          % Displacement vector
  K = zeros(neqn, neqn);

  for n = 1:nincr

    P = P + delta_P;  % increment the load

    K = zeros(neqn, neqn);
    [K, epsilon]=buildstiff(X,IX,ne,mprop,K,D,rubber_param);    % Build global tangent stiffness
    [K, delta_P]=enforce(K,delta_P,bound);        % Enforce boundary conditions
    delta_D = K \ delta_P;                         % Solve system of equations
    D = D + delta_D;

    P_plot(n, j) = P(48);
    D_plot(n, j) = D(48);
```

```
    [~, stress]=recover(mprop,X,IX,D,ne,rubber_param);

  end
  [R] = residual(stress,ne,IX, X, P, D, mprop);
  [~, R]=enforce(K,R,bound);       % Enforce boundary conditions
  residual_norm(j) = norm(R);
  disp(residual_norm(end))

  % [strain, stress, ~, ~]=recover(mprop,X,IX,D,ne,strain,stress,P,rubber_param);
end

%--- Plot results ---------------------------------------------------------%
save('euler_method.mat', 'P_plot', 'D_plot', 'residual_norm');
%save('euler_method_mi_r2.mat', 'P_plot', 'D_plot', 'residual_norm');


PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

figure(2)
legend_name = strings(1, size(incr_vector,2));
for j=1:size(incr_vector,2)
  plot(D_plot(:,j), P_plot(:,j), 'o', 'LineWidth', 2.5)
  % build a vector with the name
  legend_name(j) = strcat("Number of increment n = ", num2str(incr_vector(j)));
  hold on
end
xlabel("Displacement (m)")
ylabel("Force (N)")
legend(legend_name,'Location','southeast')
hold off

figure(3)
plot(incr_vector, residual_norm, 'o');
xlabel('Number of increments')
ylabel('Norm of the final residual')
title('Norm of the final residual')

% plot of the final displacement
figure(4)
for j=1:size(incr_vector,2)
  plot(incr_vector(j), D_plot(incr_vector(j),j), 'o');
  hold on
end
hold off
xlabel('Number of load increments')
ylabel('Final displacement')

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
```

```
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K, epsilon]=buildstiff(X,IX,ne,mprop,K, D,rubber_param);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build d vector
  [d] = build_d(D, edof);

  % compute the displacement
  epsilon = B0' * d';
  Et = Etfunction(epsilon, rubber_param);

  % element stiffness matrix
  k_e = Et * A * L0 * B0 * B0'; % 4x4 matrix

  % build K by summing k_e
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_e(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
```

```matlab
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain, stress]=recover(mprop,X,IX,D,ne,rubber_param)

% This subroutine recovers the element stress, element strain, force on each element
% and nodal reaction forces
stress = zeros(ne, 1);
strain = zeros(ne,1);

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  strain(e) = B0' * d';
  stress(e) = stress_function(strain(e), rubber_param);

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [strain,stress, N, R]=recover(mprop,X,IX,D,ne,strain,stress,P,rubber_param);
%
% % This subroutine recovers the element stress, element strain,
% % and nodal reaction forces
% strain = zeros(ne, 1);
% stress = zeros(ne, 1);
% B0_sum = zeros(2*size(X,1), 1);
% for e=1:ne
%   d = zeros(4, 1);
%   [edof] = build_edof(IX, e);
%
%   % build the matrix d from D
%   for i = 1:4
%     d(i) = D(edof(i));
%   end
%
```

```
%   % compute the bar length
%   [L0, delta_x, delta_y] = length(IX, X, e);
%
%   % displacement vector
%   B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';
%
%   % materials properties
%   propno = IX(e, 3);
%   E = mprop(propno, 1);
%   A = mprop(propno, 2);
%
%   strain(e) = B0' * d;
%   stress(e) = strain(e) * E;
%   N(e) = stress(e) * A;
%
%   % sum B0 after having transformed it in order to be compliant for the sum
%   % with P
%   for jj = 1:4
%       B0_sum(edof(jj)) = B0_sum(edof(jj)) + B0(jj)*N(e)*L0;
%   end
%
% end
%
% % compute the support reactions (N)
% R = B0_sum - P; % 2nnx1 (nn is node number)
%
% return

%%%%% Residuals %%%%
function [R] = residual(stress, ne,IX, X, P, D, mprop)
R_int = zeros(2*size(X,1), 1);
  for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  N(e) = A*stress(e,1);

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      R_int(edof(jj)) = R_int(edof(jj)) + B0(jj) * N(e) * L0;
  end

  end

R = R_int - P;
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
figure(1)
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
fake_zero = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > fake_zero  % tension
      col = colors(1);
    elseif stress(e) < - fake_zero  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return
```

```matlab
%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return

%% Function to build d from D
function [d] = build_d(D, edof)
  for i = 1:4
    d(i) = D(edof(i));
  end
return

%% Function to Et
function [Et] = Etfunction(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  Et = c4*(c1*(1 + 2*(1 + c4*epsilon)^(-3)) + 3*c2*(1 + c4*epsilon)^(-4) + ...
    3*c3*(-1 + (1 + c4*epsilon)^2 - 2*(1 + c4*epsilon)^(-3) + 2*(1 + c4*epsilon)^(-4)));
return

%% Stress function
function [stress] = stress_function(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  stress = c1*((1+c4*epsilon) - (1+c4*epsilon)^(-2)) + c2*(1 - (1+c4*epsilon)^(-3)) + c3 * (1 -
return
```

2.3 Newton Raphson

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Newton Raphson method   based on week 3                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
TrussExercise2_2022                 % Input file

neqn = size(X,1)*size(X,2);          % Number of equations
ne = size(IX,1);                     % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%
K=sparse(neqn,neqn);                 % Stiffness matrix
P_final=zeros(neqn,1);                   % Force vector
R=zeros(neqn,1);                         % Residual vector
strain=zeros(ne,size(incr_vector, 2));                 % Element strain vector
stress=zeros(ne,size(incr_vector, 2));                 % Element stress vector
P_plot=zeros(max(incr_vector), size(incr_vector, 2));
D_plot=zeros(max(incr_vector), size(incr_vector, 2));
VM_plot = zeros(1, 100); % vector for plotting the von mises curve

%--- Calculate displacements ---------------------------------------------%

[P_final] = buildload(X,IX,ne,P_final,loads,mprop); % vector of the external loads

rubber_param = [mprop(3) mprop(4) mprop(5) mprop(6)]; % coefficients for the ...
nonlinear material behaviour

residual_norm = zeros(1, size(incr_vector,2)); % vector for the norm of final residual

total_iteration = 0;

for j = 1:size(incr_vector,2) % cycle over the different # of load incr

  % number of increments
  nincr = incr_vector(j);

  % load increment
  delta_P = P_final / nincr;

  clear P D0 D
  % Initialize arrays
  P=zeros(neqn,1);                          % Force vector
  D0=zeros(neqn,1);                          % Displacement vector
  D=zeros(neqn,1);                          % Displacement vector


  for n = 1:nincr  % cycle to the number of increments
    P = P + delta_P;  % increment the load
    D0 = D;
    stress = zeros(ne,1);

    for i = 1:i_max
```

```
      K=zeros(neqn,neqn);

      [R] = residual(stress, ne,IX, X, P, D0, mprop);
      [~,R]=enforce(K,R,bound);       % Enforce boundary conditions on R
      residual_norm(j) = norm(R);
       if norm(R) <= eSTOP * Pfinal % break when we respect the eSTOP
         total_iteration = total_iteration + i;
         break
       end

      [K, ~]=buildstiff(X,IX,ne,mprop,K,D0,rubber_param);    % Build global...
      tangent stiffness matrix

      [K, ~] = enforce(K,R,bound);

%        [LM, UM] = lu(K);
%        delta_D0 = - UM \ (LM \ R);

      delta_D0 = - K \ R;

      D0 = D0 + delta_D0;


      [~, stress] = recover(mprop,X,IX,D0,ne,rubber_param);

    end

    D = D0;

    % save data of the point of interest
    P_plot(n, j) = P(48);
    D_plot(n, j) = D(48);

  end

   residual_norm(j) = norm(R); % norm of the final residual

end
display(total_iteration)
%--- Print the results on the command window -----------------------------%
% % External matrix
% disp('External forces applied (N)')
% P'

% % Stress
% disp('Stress on the bars (MPa)')
% stress'
%
% % Strain
% disp('Strain of the bars')
% strain'
%
% % Forces on the bars
% disp('Internal forces on the bar (N)')
% N
%
% % Support reaction
% disp('Support reactions forces (N)')
% R'

%--- Plot results ---------------------------------------------------------%
```

```
PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

save('NR.mat', 'P_plot', 'D_plot', 'residual_norm');
%save('NR_200.mat', 'P_plot', 'D_plot', 'residual_norm');

figure(2)
legend_name = strings(1, size(incr_vector,2));
for j=1:size(incr_vector,2)
  plot(D_plot(:,j), P_plot(:,j), 'o', 'LineWidth', 2.5)
  % build a vector with the name
  legend_name(j) = strcat("Number of increment n = ", num2str(incr_vector(j)));
  hold on
end
xlabel("Displacement (m)")
ylabel("Force (N)")
legend(legend_name,'Location','southeast')
hold off


figure(3)
plot(incr_vector, residual_norm, 'o');
xlabel('Number of increments')
ylabel('Norm of the final residual')
title('Norm of the final residual')

% plot of the final displacement
figure(4)
for j=1:size(incr_vector,2)
  plot(incr_vector(j), D_plot(incr_vector(j),j), 'o');
  hold on
end
hold off
xlabel('Number of load increments')
ylabel('Final displacement')

return


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K, epsilon]=buildstiff(X,IX,ne,mprop,K,D,rubber_param)
```

```
% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % linear strain displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build d vector
  [d] = build_d(D, edof); % d ia row vector

  % compute the displacement
  epsilon = B0' * d';
  Et = Etfunction(epsilon, rubber_param);

  % element stiffness matrix
  k_e = Et * A * L0 * B0 * B0'; % 4x4 matrix

  % build K by summing k_sum
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_e(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain, stress]=recover(mprop,X,IX,D,ne,rubber_param)

% This subroutine recovers the element stress, element strain, force on each element
% and nodal reaction forces
stress = zeros(ne, 1);
strain = zeros(ne,1);

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  strain(e) = B0' * d';
  stress(e) = stress_function(strain(e), rubber_param);

end

return
%%%% Residuals
function [R] = residual(stress, ne,IX, X, P, D, mprop)
R_int = zeros(2*size(X,1), 1);
  for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  N(e) = A*stress(e,1);

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      R_int(edof(jj)) = R_int(edof(jj)) + B0(jj) * N(e) * L0;
  end
```

```
  end

  R = R_int - P;
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
figure(1)
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
eSTOP = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > eSTOP  % tension
      col = colors(1);
    elseif stress(e) < - eSTOP  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
```

```matlab
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return

%% Function to build d from D
function [d] = build_d(D, edof)
  for i = 1:4
    d(i) = D(edof(i));
  end
return

%% Stress
function [stress] = stress_function(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  stress = c1*((1+c4*epsilon) - (1+c4*epsilon)^(-2)) + c2*(1 - (1+c4*epsilon)^(-3))...
  + c3 * (1 - 3*(1+c4*epsilon) + (1+c4*epsilon)^3 - 2*(1+c4*epsilon)^(-3) + ...
  3*(1+c4*epsilon)^(-2));
return

%% Function to Et
function [Et] = Etfunction(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  Et = c4*(c1*(1 + 2*(1 + c4*epsilon)^(-3)) + 3*c2*(1 + c4*epsilon)^(-4) + ...
  3*c3*(-1 + (1 + c4*epsilon)^2 - 2*(1 + c4*epsilon)^(-3) + ...
  2*(1 + c4*epsilon)^(-4)));
return
```

2.4 Modified Newton Raphson

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Newton Raphson modified method (based on result of week 3          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
TrussExercise2_2022                % Input file

neqn = size(X,1)*size(X,2);         % Number of equations
ne = size(IX,1);                    % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%
K=sparse(neqn,neqn);                % Stiffness matrix
P_final=zeros(neqn,1);                  % Force vector
R=zeros(neqn,1);                        % Residual vector
strain=zeros(ne,size(incr_vector, 2));                    % Element strain vector
stress=zeros(ne,size(incr_vector, 2));                    % Element stress vector
P_plot=zeros(max(incr_vector), size(incr_vector, 2));
D_plot=zeros(max(incr_vector), size(incr_vector, 2));
VM_plot = zeros(1, 100); % vector for plotting the von mises curve

%--- Calculate displacements ---------------------------------------------%

[P_final] = buildload(X,IX,ne,P_final,loads,mprop); % vector of the external loads

rubber_param = [mprop(3) mprop(4) mprop(5) mprop(6)]; % coefficients for the ...
nonlinear material behaviour
residual_norm = zeros(1, size(incr_vector, 2));

total_iteration = 0;

for j = 1:size(incr_vector,2) % cycle over the different # of load incr

  % number of increments
  nincr = incr_vector(j);

  % load increment
  delta_P = P_final / nincr;

  clear P D0 D
  % Initialize arrays
  P=zeros(neqn,1);                        % Force vector
  D0=zeros(neqn,1);                        % Displacement vector
  D=zeros(neqn,1);                        % Displacement vector

  for n = 1:nincr  % cycle to the number of increments
    stress = zeros(ne,1);
    P = P + delta_P;  % increment the load
    D0 = D;
    [K, ~]=buildstiff(X,IX,ne,mprop,K,D0,rubber_param);     % Build global tangent ...
    stiffness matrix
    [K, ~] = enforce(K,R,bound); % enforce boundary on K
    [LM, UM] = lu(K);
```

```
    for i = 1:i_max

      [R] = residual(stress, ne,IX, X, P, D0, mprop); % compute residuals
      [~,R]=enforce(K,R,bound);       % Enforce boundary conditions on R

       if norm(R) <= eSTOP * Pfinal % break when we respect the eSTOP
         total_iteration = total_iteration + i;
         break
       end

      delta_D0 = - UM \ (LM \ R);

      D0 = D0 + delta_D0;

      [~, stress] = recover(mprop,X,IX,D0,ne,rubber_param);
    end

    D = D0;

    % save data of the point of interest
    P_plot(n, j) = P(48);
    D_plot(n, j) = D(48);

  end

  residual_norm(j) = norm(R); % norm of the final residual

end
display(total_iteration);
%--- Print the results on the command window -----------------------------%
% % External matrix
% disp('External forces applied (N)')
% P'

% % Stress
% disp('Stress on the bars (MPa)')
% stress'
%
% % Strain
% disp('Strain of the bars')
% strain'
%
% % Forces on the bars
% disp('Internal forces on the bar (N)')
% N
%
% % Support reaction
% disp('Support reactions forces (N)')
% R'

%--- Plot results --------------------------------------------------------%

PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

 save('NR_modified_results.mat', 'P_plot', 'D_plot', 'residual_norm');
%save('NR_modified_200.mat', 'P_plot', 'D_plot', 'residual_norm');

figure(2)
legend_name = strings(1, size(incr_vector,2));
for j=1:size(incr_vector,2)
```

```
  plot(D_plot(:,j), P_plot(:,j), 'o', 'LineWidth', 2.5)
  % build a vector with the name
  legend_name(j) = strcat("Number of increment n = ", num2str(incr_vector(j)));
  hold on
end
xlabel("Displacement (m)")
ylabel("Force (N)")
legend(legend_name,'Location','southeast')
hold off

% plot the norm of the final residual
figure(3)
plot(incr_vector, residual_norm, 'o');
xlabel('Number of increments')
ylabel('Norm of the final residual')
title('Norm of the final residual')

% plot of the final displacement
figure(4)
for j=1:size(incr_vector,2)
  plot(incr_vector(j), D_plot(incr_vector(j),j), 'o');
  hold on
end
hold off
xlabel('Number of load increments')
ylabel('Final displacement')

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K, epsilon]=buildstiff(X,IX,ne,mprop,K,D,rubber_param)

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
```

```
  % linear strain displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build d vector
  [d] = build_d(D, edof); % d ia row vector

  % compute the displacement
  epsilon = B0' * d';
  Et = Etfunction(epsilon, rubber_param);

  % element stiffness matrix
  k_e = Et * A * L0 * B0 * B0'; % 4x4 matrix

  % build K by summing k_sum
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_e(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain, stress]=recover(mprop,X,IX,D,ne,rubber_param)

% This subroutine recovers the element stress, element strain, force on each element
% and nodal reaction forces
stress = zeros(ne, 1);
strain = zeros(ne,1);
```

```
for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  strain(e) = B0' * d';
  stress(e) = stress_function(strain(e), rubber_param);

end

return
%%%% Residuals
function [R] = residual(stress, ne,IX, X, P, D, mprop)
R_int = zeros(2*size(X,1), 1);
  for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  N(e) = A*stress(e,1);

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      R_int(edof(jj)) = R_int(edof(jj)) + B0(jj) * N(e) * L0;
  end

  end

  R = R_int - P;
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)
```

```
% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
figure(1)
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
eSTOP = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > eSTOP   % tension
      col = colors(1);
    elseif stress(e) < - eSTOP   % compression
      col = colors(2);
    else col = colors(3);   % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
```

```
return

%% Function to build d from D
function [d] = build_d(D, edof)
  for i = 1:4
    d(i) = D(edof(i));
  end
return

%% Stress
function [stress] = stress_function(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  stress = c1*((1+c4*epsilon) - (1+c4*epsilon)^(-2)) + c2*(1 - (1+c4*epsilon)^(-3)) + ...
  c3 * (1 - 3*(1+c4*epsilon) + (1+c4*epsilon)^3 - 2*(1+c4*epsilon)^(-3) + ...
  3*(1+c4*epsilon)^(-2));
return

%% Function to Et
function [Et] = Etfunction(epsilon, rubber_param)
  c1 = rubber_param(1);
  c2 = rubber_param(2);
  c3 = rubber_param(3);
  c4 = rubber_param(4);

  Et = c4*(c1*(1 + 2*(1 + c4*epsilon)^(-3)) + 3*c2*(1 + c4*epsilon)^(-4) +...
  3*c3*(-1 + (1 + c4*epsilon)^2 - 2*(1 + c4*epsilon)^(-3) +...
  2*(1 + c4*epsilon)^(-4)));
return
```

2.5 Code used for plotting all the methods in the same graph

```
clear
close all
clc

NR_modified;
NR_method;
euler_method;
text_size = 20;

close all
clear load
nincr = 100;
% euler = load('euler_method.mat', 'D_plot');
% NR = load('NR.mat');
% NR_modified = load('NR_modified.mat', 'D_plot');
%
% load_v = load('NR.mat', 'P_plot');
%
% close all
%
% figure()
% plot(NR.D_plot, load_v.P_plot, 'o')
% hold on
% plot(NR_modified.D_plot, load_v.P_plot,'x')
% plot(euler.D_plot, load_v.P_plot)
% plot(euler_correction.D_plot, load_v.P_plot)
% legend('NR', 'NR modified', 'euler', 'euler corrected','Location','southeast')
% xlabel('Displacement [m]')
% ylabel('Load P [N]')

euler = load('euler_method.mat');
NR = load('NR.mat');
NR_modified = load('NR_modified_results.mat');

%% Plot of the multiple method for 1 increment size (with zoom)
close all

figure_method_comparison = figure('Position', get(0, 'Screensize'));
plot(NR.D_plot, NR.P_plot, '-', 'LineWidth', 1.5, 'MarkerSize',6)
hold on
plot(NR_modified.D_plot, NR_modified.P_plot,'x', 'LineWidth', 1.5, 'MarkerSize',6)
plot(euler.D_plot, euler.P_plot, '--k', 'LineWidth', 1.5, 'MarkerSize',6)
legend('Newton-Raphson', 'Newton-Raphson modified', 'Euler','Location','southeast')
xlabel('Displacement [m]')
ylabel('Load P [N]')
grid on
title('Load-vertical displacement curve for point A', 'FontSize',16)
set(gca,'FontAngle','oblique','FontSize', text_size)
hold all
axes('position',[0.18 .58 .30 .30])
%a2.Position = [0 0 5 5]; % xlocation, ylocation, xsize, ysize
plot(NR.D_plot(51:63),NR.P_plot(51:63), 'LineWidth', 1.5, 'MarkerSize',6); %axis tight
hold on
plot(NR_modified.D_plot(51:63),NR_modified.P_plot(51:63),  'x', 'LineWidth', 1.5, ...
'MarkerSize',6); %axis tight
plot(euler.D_plot(51:63),euler.P_plot(51:63), '--k', 'LineWidth', 1.5, ...
'MarkerSize',6); axis tight
grid on
set(gca,'FontAngle','oblique','FontSize', text_size)
```

```
saveas(figure_method_comparison, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\...
assignment1\method_comparison.png','png');

%% Plot multiple increments

figure()
iteration_n = [1:1:nincr];
plot(iteration_n, NR.residual_norm);
hold on
plot(iteration_n, euler.residual_norm);
plot(iteration_n, NR_modified.residual_norm);
hold off
xlabel('Increment number')
ylabel('Norm of he residual')
title('Norm of the residuals')
```

3 Exercise 3

3.1 Input file used to determine EI

```
% Created with:       FlExtract v1.13
% Element type:       truss
% Number of nodes:    34
% Number of elements: 81

clear all;

A = 2;  % cross section
incr_vector = [50];  % number of increment
i_max = 1000; % maximum number of iterations
n_incr = 10;
eSTOP = 1e-8;
Pfinal = 0.00005;
a = 0.4;
k = 0.02; % spring stiffness

% Node coordinates: x, y
X = [
0    0
0    5
5    0
5    5
10   0
10   5
15   0
15   5
20   0
20   5
25   0
25   5
30   0
30   5
35   0
35   5
40   0
40   5
45   0
45   5
50   0
50   5
55   0
55   5
60   0
60   5
65   0
65   5
70   0
70   5
75   0
75   5
80   0
80   5
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2    1    1
4    1    1
```

3	1	1
4	2	1
3	2	1
4	3	1
6	3	1
5	3	1
6	4	1
5	4	1
6	5	1
8	5	1
7	5	1
8	6	1
7	6	1
8	7	1
10	7	1
9	7	1
10	8	1
9	8	1
10	9	1
12	9	1
11	9	1
12	10	1
11	10	1
12	11	1
14	11	1
13	11	1
14	12	1
13	12	1
14	13	1
16	13	1
15	13	1
16	14	1
15	14	1
16	15	1
18	15	1
17	15	1
18	16	1
17	16	1
18	17	1
20	17	1
19	17	1
20	18	1
19	18	1
20	19	1
22	19	1
21	19	1
22	20	1
21	20	1
22	21	1
24	21	1
23	21	1
24	22	1
23	22	1
24	23	1
26	23	1
25	23	1
26	24	1
25	24	1
26	25	1
28	25	1
27	25	1

```
28   26   1
27   26   1
28   27   1
30   27   1
29   27   1
30   28   1
29   28   1
30   29   1
32   29   1
31   29   1
32   30   1
31   30   1
32   31   1
34   31   1
33   31   1
34   32   1
33   32   1
34   33   1
];
% Element properties: Young's modulus, area
mprop = [
0.8  0.2
];
% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
%1 1 0
1    2    0
33   2    0
];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [
18   2    -Pfinal
];
% Control parameters
plotdof = 2;
```

3.2 File used to evaluate EI

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                         Basis truss program                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
vertical_disp_struct

neqn = size(X,1)*size(X,2);          % Number of equations
ne = size(IX,1);                     % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%
Kmatr=sparse(neqn,neqn);                 % Stiffness matrix
P=zeros(neqn,1);                         % Force vector
D=zeros(neqn,1);                         % Displacement vector
R=zeros(neqn,1);                         % Residual vector
strain=zeros(ne,1);                      % Element strain vector
stress=zeros(ne,1);                      % Element stress vector

%--- Calculate displacements ---------------------------------------------%
[P]=buildload(X,IX,ne,P,loads,mprop);        % Build global load vector

[Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr);     % Build global stiffness matrix

[Kmatr,Pmatr]=enforce(Kmatr,P,bound);        % Enforce boundary conditions

D = Kmatr \ Pmatr;                           % Solve system of equations

[strain,stress]=recover(mprop,X,IX,D,ne,strain,stress); % Calculate element
                                                        % stress and strain

% Calculate element axial forces and the support reactions
[strain,stress,N,R]=recover2(mprop,X,IX,D,ne,strain,stress,P);

%--- Print the results on the command window -----------------------------%
vertical_load = P(36);
vertical_D = D(36);
EI = vertical_load * 80^3/(48 * vertical_D);
P_critic = pi^2*EI/80^2;

disp(strcat('External load P = ', num2str(vertical_load)))
disp(strcat('Vertical displacement D = ', num2str(vertical_D)))
disp(strcat('EIx = ', num2str(EI)))
disp(strcat('Critical load P  = ', num2str(P_critic)))

% % External matrix
% disp('External forces applied (N)')
% P'
%
% % Stress
% disp('Stress on the bars (MPa)')
% stress'
%
```

```
% % Strain
% disp('Strain of the bars')
% strain'
%
% % Forces on the bars
% disp('Internal forces on the bar (N)')
% N
%
% % Support reaction
% disp('Support reactions forces (N)')
% R'

%--- Plot results --------------------------------------------------------%
PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_e = E * A * L0 * B0 * B0'; % 4x4 matrix

  % vector of index used for building K
  [edof] = build_edof(IX, e);
```

```
  % build K by summing k_e
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_e(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress]=recover(mprop,X,IX,D,ne,strain,stress);

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces

% allocate memory for stress and strain vectors
strain = zeros(ne, 1);
stress = zeros(ne, 1);

for e=1:ne
  d = zeros(4, 1); % allocate memory for element stiffness matrix

 [edof] = build_edof(IX, e); % index for buildg K

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
```

```
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  strain(e) = B0' * d;
  stress(e) = strain(e) * E;

end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress, N, R]=recover2(mprop,X,IX,D,ne,strain,stress,P);

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
strain = zeros(ne, 1);
stress = zeros(ne, 1);
B0_sum = zeros(2*size(X,1), 1);
for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  strain(e) = B0' * d;
  stress(e) = strain(e) * E;
  N(e) = stress(e) * A;

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      B0_sum(edof(jj)) = B0_sum(edof(jj)) + B0(jj)*N(e)*L0;
  end

end

% compute the support reactions (N)
R = B0_sum - P; % 2nnx1 (nn is node number)



return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
fake_zero = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > fake_zero  % tension
      col = colors(1);
    elseif stress(e) < - fake_zero  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e);

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e);
```

```
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return
```

3.3 Input file used to compute buckling of column alone

```
% Created with:       FlExtract v1.13
% Element type:       truss
% Number of nodes:    34
% Number of elements: 81

clear all;

A = 2;  % cross section
%incr_vector = [100];  % number of increment
i_max = 1000; % maximum number of iterations
nincr = 100;
eSTOP = 1e-8;
Pfinal = 0.004;

% Node coordinates: x, y
X = [
0     0
0     5
5     0
5     5
10    0
10    5
15    0
15    5
20    0
20    5
25    0
25    5
30    0
30    5
35    0
35    5
40    0
40    5
45    0
45    5
50    0
50    5
55    0
55    5
60    0
60    5
65    0
65    5
70    0
70    5
75    0
75    5
80    0
80    5
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2     1     1
4     1     1
3     1     1
4     2     1
3     2     1
4     3     1
```

6	3	1
5	3	1
6	4	1
5	4	1
6	5	1
8	5	1
7	5	1
8	6	1
7	6	1
8	7	1
10	7	1
9	7	1
10	8	1
9	8	1
10	9	1
12	9	1
11	9	1
12	10	1
11	10	1
12	11	1
14	11	1
13	11	1
14	12	1
13	12	1
14	13	1
16	13	1
15	13	1
16	14	1
15	14	1
16	15	1
18	15	1
17	15	1
18	16	1
17	16	1
18	17	1
20	17	1
19	17	1
20	18	1
19	18	1
20	19	1
22	19	1
21	19	1
22	20	1
21	20	1
22	21	1
24	21	1
23	21	1
24	22	1
23	22	1
24	23	1
26	23	1
25	23	1
26	24	1
25	24	1
26	25	1
28	25	1
27	25	1
28	26	1
27	26	1
28	27	1
30	27	1

```
29   27   1
30   28   1
29   28   1
30   29   1
32   29   1
31   29   1
32   30   1
31   30   1
32   31   1
34   31   1
33   31   1
34   32   1
33   32   1
34   33   1
];
% Element properties: Young's modulus, area
mprop = [
0.8  0.2
];
% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
1    2    0
2 2 0
33   2    0
34 2 0
1 1 0
];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [
1 1 Pfinal*(0.5+1e-3)
33   1    -Pfinal*(0.5+1e-3)
2 1 Pfinal/2
34   1    -Pfinal/2
];
% Control parameters
plotdof = 5;
```

3.4 File used to evaluate the buckling of the colum alone

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Newton Raphson method                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file -----------------------------------------------------------%
buckling_struct
%week3_ex_3_struct

neqn = size(X,1)*size(X,2);         % Number of equations
ne = size(IX,1);                    % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ----------------------------------------------------%
K=sparse(neqn,neqn);                % Stiffness matrix
P_final=zeros(neqn,1);                  % Force vector
R=zeros(neqn,1);                        % Residual vector
strain=zeros(ne,1);                     % Element strain vector
stress=zeros(ne,1);                     % Element stress vector
P_plot=zeros(nincr, 1);
D_plot=zeros(nincr, 1);


%--- Calculate displacements ----------------------------------------------%

[P_final] = buildload(X,IX,ne,P_final,loads,mprop); % vector of the external loads

% load increment
delta_P = P_final / nincr;

clear P D0 D
% Initialize arrays
P=zeros(neqn,1);                        % Force vector
D0=zeros(neqn,1);                        % Displacement vector
D=zeros(neqn,1);                        % Displacement vector

for n = 1:nincr  % cycle to the number of increments
  P = P + delta_P;  % increment the load
  D0 = D;

  for i = 1:i_max
    K=zeros(neqn,neqn);

    [R] = residual(stress, ne,IX, X, P, D0, mprop);
    [~,R]=enforce(K,R,bound);       % Enforce boundary conditions on R

     if norm(R) <= eSTOP * norm(Pfinal) % break when we respect the eSTOP
       %CONV=i
       %pause
      break
     end

    [K, ~]=buildstiff(X,IX,ne,mprop,K,D0);     % Build global tangent stiffness matrix
```

```
    [K, ~] = enforce(K,R,bound);

%        [LM, UM] = lu(K);
%        delta_D0 = - UM \ (LM \ R);

    delta_D0 = - K \ R;

    D0 = D0 + delta_D0;

    [~, stress] = recover(mprop,X,IX,D0,ne);

  end

  D = D0;

  % save data of the point of interest
  P_plot(n) = 2*P(1);
  D_plot(n) = D(1) - D(65);

end


%--- Print the results on the command window -----------------------------%

%disp(strcat('The maximum vertical displacement is d = ', num2str(v_disp)))
% % External matrix
% disp('External forces applied (N)')
% P'

% % Stress
% disp('Stress on the bars (MPa)')
% stress'
%
% % Strain
% disp('Strain of the bars')
% strain'
%
% % Forces on the bars
% disp('Internal forces on the bar (N)')
% N
%
% % Support reaction
% disp('Support reactions forces (N)')
% R'

%--- Plot results --------------------------------------------------------%

%save('NR.mat', 'P_plot', 'D_plot');

PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

[L0, ~, ~] = length(IX, X, 1);
lin_space = linspace(0, 0.9);

figure(2)
plot(D_plot, P_plot, '-', 'LineWidth', 2.5)
% build a vector with the name
xlabel("Displacement (m)")
ylabel("Force (N)")
yline(0.00299, '--r', 'Theoretical bouckling load', 'LineWidth',1.5)
```

```
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K, epsilon_G]=buildstiff(X,IX,ne,mprop,K,D);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % linear strain displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build d vector
  [d] = build_d(D, edof); % d ia row vector

   % displacement dependent vector Bd
  Bd = 1/L0^2 * [ 1 0 -1 0;
                  0 1 0 -1;
                  -1 0 1 0;
                  0 -1 0 1] * d';

  % compute the displacement
  epsilon_G = B0' * d' + 1/2*Bd'*d';

  % bar non-physical force
  N_G = A*E*epsilon_G;

  % element stiffness matrix
```

```
  k_0 = E * A * L0 * B0 * B0'; % 4x4 matrix
  k_d = E * A * L0 * B0 * Bd' + E * A * L0 * Bd * B0'+ E * A * L0 * Bd * Bd';
  k_sigma = 1/L0^2 * [ 1 0 -1 0; 0 1 0 -1; -1 0 1 0; 0 -1 0 1]*N_G*L0;

  k_sum = k_0 + k_d + k_sigma;

  % build K by summing k_sum
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_sum(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [epsilon_G, stress_G]=recover(mprop,X,IX,D,ne)

% This subroutine recovers the element stress, element strain, force on each element
% and nodal reaction forces
stress_G = zeros(ne, 1);

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % displacement dependent vector Bd
```

```
  Bd = 1/L0^2 * [ 1 0 -1 0;
                  0 1 0 -1;
                  -1 0 1 0;
                  0 -1 0 1] * d';


  % compute the displacement
  epsilon_G = B0' * d' + 1/2 * Bd' * d';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);
  E = mprop(propno, 1);

  stress_G(e) = E*epsilon_G;

end

return
%%%% Residuals
function [R] = residual(stress, ne,IX, X, P, D, mprop)
R_int = zeros(2*size(X,1), 1);
  for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % displacement dependent vector Bd
  Bd = 1/L0^2 * [ 1 0 -1 0;
                  0 1 0 -1;
                  -1 0 1 0;
                  0 -1 0 1] * d';

  B_bar = B0 + Bd;

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  N_G = A*stress(e,1);

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      R_int(edof(jj)) = R_int(edof(jj)) + B_bar(jj) * N_G * L0;
  end

  end

  R = R_int - P;
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
figure(1)
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
eSTOP = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > eSTOP  % tension
      col = colors(1);
    elseif stress(e) < - eSTOP  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
```

```
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return

%% Function to build d from D
function [d] = build_d(D, edof)
  for i = 1:4
    d(i) = D(edof(i));
  end
return
```

3.5 Input file for question 3

```
% Created with:       FlExtract v1.13
% Element type:       truss
% Number of nodes:    156
% Number of elements: 443

close all
clear all
clc

%SCALE = 1; % Ec = 0.8
%SCALE = 0.5; % Ec = 0.4
SCALE = 2; % Ec = 1.6

nincr  = 200;
i_max  = 1000;
eSTOP  = 10^(-9);
Pfinal = -0.004*SCALE;


% Node coordinates: x, y
X = [
0    0
0    5
0    10
0    15
0    20
0    25
0    30
0    35
0    40
0    45
0    50
0    55
0    60
0    65
0    70
0    75
0    80
0    85
0    90
0    95
0    100
5    0
5    5
5    10
5    15
5    20
5    25
5    30
5    35
5    40
5    45
5    50
5    55
5    60
5    65
5    70
5    75
5    80
```

5	85
5	90
5	95
5	100
10	0
10	5
10	10
10	90
10	95
10	100
15	0
15	5
15	10
15	90
15	95
15	100
20	0
20	5
20	10
20	90
20	95
20	100
25	0
25	5
25	10
25	90
25	95
25	100
30	0
30	5
30	10
30	90
30	95
30	100
35	0
35	5
35	10
35	90
35	95
35	100
40	0
40	5
40	10
40	90
40	95
40	100
45	0
45	5
45	10
45	90
45	95
45	100
50	0
50	5
50	10
50	90
50	95
50	100
55	0
55	5
55	10

```
55   90
55   95
55   100
60   0
60   5
60   10
60   90
60   95
60   100
65   0
65   5
65   10
65   90
65   95
65   100
70   0
70   5
70   10
70   90
70   95
70   100
75   0
75   5
75   10
75   90
75   95
75   100
80   0
80   5
80   10
80   90
80   95
80   100
85   0
85   5
85   10
85   90
85   95
85   100
90   0
90   5
90   10
90   90
90   95
90   100
95   0
95   5
95   10
95   90
95   95
95   100
100  0
100  5
100  10
100  90
100  95
100  100
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2    1    1
```

23	1	1
22	1	1
3	2	1
24	2	1
23	2	1
22	2	1
4	3	2
25	3	2
24	3	1
23	3	1
5	4	2
26	4	2
25	4	2
24	4	2
6	5	2
27	5	2
26	5	2
25	5	2
7	6	2
28	6	2
27	6	2
26	6	2
8	7	2
29	7	2
28	7	2
27	7	2
9	8	2
30	8	2
29	8	2
28	8	2
10	9	2
31	9	2
30	9	2
29	9	2
11	10	2
32	10	2
31	10	2
30	10	2
12	11	2
33	11	2
32	11	2
31	11	2
13	12	2
34	12	2
33	12	2
32	12	2
14	13	2
35	13	2
34	13	2
33	13	2
15	14	2
36	14	2
35	14	2
34	14	2
16	15	2
37	15	2
36	15	2
35	15	2
17	16	2
38	16	2
37	16	2

36	16	2
18	17	2
39	17	2
38	17	2
37	17	2
19	18	2
40	18	2
39	18	2
38	18	2
20	19	1
41	19	1
40	19	1
39	19	2
21	20	1
42	20	1
41	20	1
40	20	1
42	21	1
41	21	1
23	22	1
44	22	1
43	22	1
24	23	1
45	23	1
44	23	1
43	23	1
25	24	2
45	24	1
44	24	1
26	25	2
27	26	2
28	27	2
29	28	2
30	29	2
31	30	2
32	31	2
33	32	2
34	33	2
35	34	2
36	35	2
37	36	2
38	37	2
39	38	2
40	39	2
41	40	1
47	40	1
46	40	1
42	41	1
48	41	1
47	41	1
46	41	1
48	42	1
47	42	1
44	43	1
50	43	1
49	43	1
45	44	1
51	44	1
50	44	1
49	44	1
51	45	1

50	45	1
47	46	1
53	46	1
52	46	1
48	47	1
54	47	1
53	47	1
52	47	1
54	48	1
53	48	1
50	49	1
56	49	1
55	49	1
51	50	1
57	50	1
56	50	1
55	50	1
57	51	1
56	51	1
53	52	1
59	52	1
58	52	1
54	53	1
60	53	1
59	53	1
58	53	1
60	54	1
59	54	1
56	55	1
62	55	1
61	55	1
57	56	1
63	56	1
62	56	1
61	56	1
63	57	1
62	57	1
59	58	1
65	58	1
64	58	1
60	59	1
66	59	1
65	59	1
64	59	1
66	60	1
65	60	1
62	61	1
68	61	1
67	61	1
63	62	1
69	62	1
68	62	1
67	62	1
69	63	1
68	63	1
65	64	1
71	64	1
70	64	1
66	65	1
72	65	1
71	65	1

70	65	1
72	66	1
71	66	1
68	67	1
74	67	1
73	67	1
69	68	1
75	68	1
74	68	1
73	68	1
75	69	1
74	69	1
71	70	1
77	70	1
76	70	1
72	71	1
78	71	1
77	71	1
76	71	1
78	72	1
77	72	1
74	73	1
80	73	1
79	73	1
75	74	1
81	74	1
80	74	1
79	74	1
81	75	1
80	75	1
77	76	1
83	76	1
82	76	1
78	77	1
84	77	1
83	77	1
82	77	1
84	78	1
83	78	1
80	79	1
86	79	1
85	79	1
81	80	1
87	80	1
86	80	1
85	80	1
87	81	1
86	81	1
83	82	1
89	82	1
88	82	1
84	83	1
90	83	1
89	83	1
88	83	1
90	84	1
89	84	1
86	85	1
92	85	1
91	85	1
87	86	1

93	86	1
92	86	1
91	86	1
93	87	1
92	87	1
89	88	1
95	88	1
94	88	1
90	89	1
96	89	1
95	89	1
94	89	1
96	90	1
95	90	1
92	91	1
98	91	1
97	91	1
93	92	1
99	92	1
98	92	1
97	92	1
99	93	1
98	93	1
95	94	1
101	94	1
100	94	1
96	95	1
102	95	1
101	95	1
100	95	1
102	96	1
101	96	1
98	97	1
104	97	1
103	97	1
99	98	1
105	98	1
104	98	1
103	98	1
105	99	1
104	99	1
101	100	1
107	100	1
106	100	1
102	101	1
108	101	1
107	101	1
106	101	1
108	102	1
107	102	1
104	103	1
110	103	1
109	103	1
105	104	1
111	104	1
110	104	1
109	104	1
111	105	1
110	105	1
107	106	1
113	106	1

112	106	1
108	107	1
114	107	1
113	107	1
112	107	1
114	108	1
113	108	1
110	109	1
116	109	1
115	109	1
111	110	1
117	110	1
116	110	1
115	110	1
117	111	1
116	111	1
113	112	1
119	112	1
118	112	1
114	113	1
120	113	1
119	113	1
118	113	1
120	114	1
119	114	1
116	115	1
122	115	1
121	115	1
117	116	1
123	116	1
122	116	1
121	116	1
123	117	1
122	117	1
119	118	1
125	118	1
124	118	1
120	119	1
126	119	1
125	119	1
124	119	1
126	120	1
125	120	1
122	121	1
128	121	1
127	121	1
123	122	1
129	122	1
128	122	1
127	122	1
129	123	1
128	123	1
125	124	1
131	124	1
130	124	1
126	125	1
132	125	1
131	125	1
130	125	1
132	126	1
131	126	1

128	127	1
134	127	1
133	127	1
129	128	1
135	128	1
134	128	1
133	128	1
135	129	1
134	129	1
131	130	1
137	130	1
136	130	1
132	131	1
138	131	1
137	131	1
136	131	1
138	132	1
137	132	1
134	133	1
140	133	1
139	133	1
135	134	1
141	134	1
140	134	1
139	134	1
141	135	1
140	135	1
137	136	1
143	136	1
142	136	1
138	137	1
144	137	1
143	137	1
142	137	1
144	138	1
143	138	1
140	139	1
146	139	1
145	139	1
141	140	1
147	140	1
146	140	1
145	140	1
147	141	1
146	141	1
143	142	1
149	142	1
148	142	1
144	143	1
150	143	1
149	143	1
148	143	1
150	144	1
149	144	1
146	145	1
152	145	1
151	145	1
147	146	1
153	146	1
152	146	1
151	146	1

```
153  147  1
152  147  1
149  148  1
155  148  1
154  148  1
150  149  1
156  149  1
155  149  1
154  149  1
156  150  1
155  150  1
152  151  1
153  152  1
155  154  1
156  155  1
];
% Element properties: Young's modulus, area
mprop = [0.2              0.1 % beam
         0.8*SCALE       0.2]; % column
% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
151  1    0
151  2    0
152  1    0
152  2    0
153  1    0
153  2    0
154  1    0
154  2    0
155  1    0
155  2    0
156  1    0
156  2    0
];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [
21   2    Pfinal
1    2    -Pfinal
];
% Control parameters
plotdof = 21;
```

3.6 File for question 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Newton Raphson method                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
TrussExercise3_2022

neqn = size(X,1)*size(X,2);          % Number of equations
ne = size(IX,1);                     % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays --------------------------------------------------%
K=sparse(neqn,neqn);                 % Stiffness matrix
P_final=zeros(neqn,1);                   % Force vector
R=zeros(neqn,1);                         % Residual vector
strain=zeros(ne,1);                      % Element strain vector
stress=zeros(ne,1);                      % Element stress vector
P_plot=zeros(nincr, 1);
D_plot=zeros(nincr, 1);


%--- Calculate displacements --------------------------------------------%

[P_final] = buildload(X,IX,ne,P_final,loads,mprop); % vector of the external loads

% load increment
delta_P = P_final / nincr;

clear P D0 D
% Initialize arrays
P=zeros(neqn,1);                         % Force vector
D0=zeros(neqn,1);                         % Displacement vector
D=zeros(neqn,1);                         % Displacement vector

for n = 1:nincr  % cycle to the number of increments
  P = P + delta_P;  % increment the load
  D0 = D;

  for i = 1:i_max
    K=zeros(neqn,neqn);

    [R] = residual(stress, ne,IX, X, P, D0, mprop);
    [~,R]=enforce(K,R,bound);        % Enforce boundary conditions on R

     if norm(R) <= eSTOP * norm(Pfinal) % break when we respect the eSTOP
      break
     end

    [K, ~]=buildstiff(X,IX,ne,mprop,K,D0);     % Build global tangent stiffness matrix

    [K, ~] = enforce(K,R,bound);

    [LM, UM] = lu(K);
```

```
    delta_D0 = - UM \ (LM \ R);

    %delta_D0 = - K \ R;

    D0 = D0 + delta_D0;

    [~, stress] = recover(mprop,X,IX,D0,ne);

  end

  D = D0;

  % save data of the point of interest
  P_plot(n) = 2*P(2);
  D_plot(n) = - D(42) + D(2);

end

%save('E3_case1.mat', 'P_plot', 'D_plot'); % Ec = 0.8
%save('E3_case2.mat', 'P_plot', 'D_plot'); % Ec = 0.4
save('E3_case3.mat', 'P_plot', 'D_plot'); % Ec = 1.6

%--- Print the results on the command window -----------------------------%

%disp(strcat('The maximum vertical displacement is d = ', num2str(v_disp)))
% % External matrix
% disp('External forces applied (N)')
% P'

% % Stress
% disp('Stress on the bars (MPa)')
% stress'
%
% % Strain
% disp('Strain of the bars')
% strain'
%
% % Forces on the bars
% disp('Internal forces on the bar (N)')
% N
%
% % Support reaction
% disp('Support reactions forces (N)')
% R'

%--- Plot results ---------------------------------------------------------%

PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)        % Plot structure

[L0, ~, ~] = length(IX, X, 1);
lin_space = linspace(0, 0.9);

force_vs_disp_ex3_3 = figure('Position', get(0, 'Screensize'));
plot(D_plot, P_plot, '-', 'LineWidth', 2.5)
xlabel("Displacement (m)")
ylabel("Force (N)")
legend('disp','Location','southeast')
set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
saveas(force_vs_disp_ex3_3 , 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\...
assignment1\force_vs_disp_ex3_3.png','png');
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K, epsilon_G]=buildstiff(X,IX,ne,mprop,K,D);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % linear strain displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build d vector
  [d] = build_d(D, edof); % d ia row vector

   % displacement dependent vector Bd
  Bd = 1/L0^2 * [ 1 0 -1 0;
                  0 1 0 -1;
                  -1 0 1 0;
                  0 -1 0 1] * d';

  % compute the displacement
  epsilon_G = B0' * d' + 1/2*Bd'*d';

  % bar non-physical force
  N_G = A*E*epsilon_G;

  % element stiffness matrix
  k_0 = E * A * L0 * B0 * B0'; % 4x4 matrix
```

```
  k_d = E * A * L0 * B0 * Bd' + E * A * L0 * Bd * B0'+ E * A * L0 * Bd * Bd';
  k_sigma = 1/L0^2 * [ 1 0 -1 0; 0 1 0 -1; -1 0 1 0; 0 -1 0 1]*N_G*L0;

  k_sum = k_0 + k_d + k_sigma;

  % build K by summing k_sum
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + k_sum(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [epsilon_G, stress_G]=recover(mprop,X,IX,D,ne)

% This subroutine recovers the element stress, element strain, force on each element
% and nodal reaction forces
stress_G = zeros(ne, 1);

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % displacement dependent vector Bd
  Bd = 1/L0^2 * [ 1 0 -1 0;
```

```
                0 1 0 -1;
                -1 0 1 0;
                0 -1 0 1] * d';


  % compute the displacement
  epsilon_G = B0' * d' + 1/2 * Bd' * d';

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);
  E = mprop(propno, 1);

  stress_G(e) = E*epsilon_G;

end

return
%%%% Residuals
function [R] = residual(stress, ne,IX, X, P, D, mprop)
R_int = zeros(2*size(X,1), 1);
  for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  d = build_d(D, edof);

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);

  % displacement vector
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % displacement dependent vector Bd
  Bd = 1/L0^2 * [ 1 0 -1 0;
                0 1 0 -1;
                -1 0 1 0;
                0 -1 0 1] * d';

  B_bar = B0 + Bd;

  % materials properties
  propno = IX(e, 3);
  A = mprop(propno, 2);

  N_G = A*stress(e,1);

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      R_int(edof(jj)) = R_int(edof(jj)) + B_bar(jj) * N_G * L0;
  end

  end

  R = R_int - P;
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
figure(1)
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
eSTOP = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > eSTOP   % tension
      col = colors(1);
    elseif stress(e) < - eSTOP   % compression
      col = colors(2);
    else col = colors(3);   % un-loaded
    end

    h2=plot(xx,yy, col,'LineWidth',3.5); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
```

```
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return

%% Function to build d from D
function [d] = build_d(D, edof)
  for i = 1:4
    d(i) = D(edof(i));
  end
return
```

3.7 File used to plot all the case in the same graph

```
case1 = load('E3_case1.mat', 'D_plot', 'P_plot');
case2 = load('E3_case2.mat', 'D_plot', 'P_plot');
case3 = load('E3_case3.mat', 'D_plot', 'P_plot');

E3_comparison = figure('Position', get(0, 'Screensize'));
plot(case2.D_plot, case2.P_plot, 'LineWidth', 1.5)
hold on
plot(case1.D_plot, case1.P_plot, 'LineWidth', 1.5)
plot(case3.D_plot, case3.P_plot, 'LineWidth', 1.5)
hold off
xlabel('\Delta_Y [m]')
ylabel('Force [N]')
legend('E_{c}=0.4, P_{f}=0.002', 'E_{c}=0.8, P_{f}=0.004', 'E_{c}=1.6, P_{f}=0.008',...
'Location','northwest')
title('Influence of materila properties')
set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
saveas(E3_comparison, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\...
E3_comparison.png','png');
```

4 Exercise 4

4.1 Bisect function

```
function [rho] = bisect(rho_old, V_star, f_grad, g_grad, rho_min, eta, epsilon)
% Function to compute rho with the bisection algorithm

lambda_1 = 1e-10; % guess value for low extreme value
lambda_2 = 1e10;  % guess value for high extreme value


while( (lambda_2 - lambda_1)/( lambda_1 + lambda_2) > epsilon)
  lambda_mid = (lambda_2 + lambda_1) / 2;

  ne = size(f_grad, 1); % number of elements in the structure
  rho = zeros(ne, 1);
  for e = 1:ne
    Be = - f_grad(e) / (lambda_mid * g_grad(e));

    if rho_old(e) * Be^eta <= rho_min
      rho(e) = rho_min;
    elseif rho_min < rho_old(e) * Be^eta &&  rho_old(e) * Be^eta < 1
      rho(e) = rho_old(e) * Be^eta;
    elseif rho_old(e) * Be^eta >= 1
      rho(e) = 1;
    end

  end

  if rho' * g_grad - V_star > 0
    lambda_1 = lambda_mid;
  else
    lambda_2 = lambda_mid;
  end

end


end
```

4.2 Code for η optimization

Structure 3 with 40% connectivity has been used for the optimization, for $\eta \in [0.65, 0.69]$.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Basic truss program                              %
%     Exercise 4.2 - Assignment 1: eta optimization before varying p      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
mesh3_40

neqn = size(X,1)*size(X,2);          % Number of equations
ne = size(IX,1);                     % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

iterations = zeros(size(eta_vector,2),1);

for j = 1:size(eta_vector,2)

    eta = eta_vector(j);
    clear f_vector

%--- Initialize arrays ---------------------------------------------------%
Kmatr=sparse(neqn,neqn);                 % Stiffness matrix
P=zeros(neqn,1);                         % Force vector
D=zeros(neqn,1);                         % Displacement vector
R=zeros(neqn,1);                         % Residual vector
strain=zeros(ne,1);                      % Element strain vector
stress=zeros(ne,1);                      % Element stress vector
rho = zeros(ne,1);

[~,~,~,~,~,g_grad]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

v = g_grad; % vector of element volumes and g_grad are the same

% Initialize the vector of relative element densities:
% We need the pseudo inverse matrix in order to initialize rho satisfying
% g=0:

v_sum = sum(v);
rho(:) = V_star/v_sum; % initially think rho as constant

[P]=buildload(X,IX,ne,P,loads,mprop);        % Build global load vector

for iopt = 1:iopt_max

    rho_old = rho;

    %--- Calculate displacements -------------------------------------------%
    [Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr,p,rho_old);    % Build global stiffness matrix

    [Kmatr,P]=enforce(Kmatr,P,bound);              % Enforce boundary conditions

    D = Kmatr \ P;                                     % Solve system of equations
```

```
    [~,~,~,~,f_grad,~,~]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho_old);

    rho = bisect(rho_old,V_star,f_grad,g_grad,rho_min,eta);

    [~,~,~,~,~,~,f]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

    f_vector(iopt) = f; % vector for convergence plot, cell j and entrance i

    if norm(rho_old - rho) < norm(rho)*epsilon
        break
    end

end

f_cell{j} = f_vector;
iterations(j) = iopt;

% Save f_vector for convergence plot for different eta values:

[strain,stress,~,~,~,~,~]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

% Check on the strain energy density = constant for active members:
energy_density = zeros(ne,1);

for e = 1:ne
    energy_density(e) = (strain(e)*stress(e))/rho(e);
end

end

iterations % stamp number of iterations for the current eta value

figure('color','white')
for s = 1:size(eta_vector,2)
    x_step = [1:1:iterations(s)];
    plot(x_step,f_cell{s},'.-','MarkerSize', 8)
    legend_name(s) = strcat("\eta = ",num2str(eta_vector(s)));
    hold on
end

grid on
set(gca,'FontAngle','oblique','FontSize',14)
legend(legend_name,'location','northeast')
title('Convergence plot for different eta values', 'FontSize', 16)
xlabel('Topology Optimization iteration')
ylabel('Compliance f(\rho)')

% % Convergence plot:
% figure(1)
% x_step = [1:1:(iopt)];
% plot(x_step,f_vector,'-rx')
% grid on
% set(gca,'FontAngle','oblique','FontSize',14)
% title('Convergence plot', 'FontSize', 16)
% xlabel('Topology Optimization iteration')
% ylabel('Compliance f')

%--- Plot results ---------------------------------------------------------%
% figure(2)
% PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)        % Plot structure
```

```
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop)

P=zeros(size(X,1)*size(X,2),1);
rowX = size(X,1);
columnX = size(X,2);

for i=1:size(loads,1)

    pos = loads(i,1)*columnX - (columnX - loads(i,2));
    P(pos, 1) = loads(i,3);

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K,p,rho)

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

    K = zeros(2*size(X,1)); % global stiffness matrix def., 6x6, memory allocation

for e=1:ne

    i = IX(e,1);
    j = IX(e,2);
    xi = X(i,1);
    yi = X(i,2);
    xj = X(j,1);
    yj = X(j,2);

    delta_x = (xj-xi);
    delta_y = (yj-yi);

    L0 = sqrt(delta_x^2 + delta_y^2);

    B0 = (1/(L0^2))*[-delta_x, -delta_y, delta_x, delta_y]';

    % material properties of the current beam
    propno = IX(e,3);
    E0 = mprop(propno,1);
    A = mprop(propno,2);

    k_0e = E0*A*L0*(B0)*(B0'); % 4x4 matrix, local

    k_e = (rho(e))^p * k_0e;

    edof = [IX(e,1)*2 - 1, IX(e,1)*2 - 0, IX(e,2)*2 - 1, IX(e,2)*2 - 0];

    for ii = 1:4
        for jj = 1:4
            K(edof(ii),edof(jj)) = K(edof(ii),edof(jj)) + k_e(ii,jj);
        end
    end
```

```
end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound)

% This subroutine enforces the support boundary conditions

% 0 and 1 Method:

for i=1:size(bound,1)
    node = bound(i,1); % from example1.m data
    ldof = bound(i,2);
    K(:,2*node - (2-ldof)) = 0; % zeros in the column
    K(2*node - (2-ldof),:) = 0; % zeros in the row
    K(2*node - (2-ldof),2*node - (2-ldof)) = 1; % put a 1 in the diagonal

    P(2*node - (2-ldof)) = 0; % updating P vector
end

return

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress,N,R,f_grad,g_grad,f]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho)

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
d = zeros(4,1);
strain = zeros(ne,1); % allocation strain vector
stress = zeros(ne,1); % allocation stress vector
N = zeros(ne,1); % allocation element forces vector
B0sum = zeros(2*size(X,1),1);
f_grad = zeros(ne,1);
g_grad = zeros(ne,1);
f = 0; % initialize outside of the loop

for e=1:ne
    edof = [IX(e,1)*2 - 1, IX(e,1)*2 - 0, IX(e,2)*2 - 1, IX(e,2)*2 - 0];
    % define small d
    for i=1:4
        d(i) = D(edof(i));
    end

    i = IX(e,1);
    j = IX(e,2);
    xi = X(i,1);
    yi = X(i,2);
    xj = X(j,1);
    yj = X(j,2);

    delta_x = (xj-xi);
    delta_y = (yj-yi);

    L0 = sqrt(delta_x^2 + delta_y^2);

    B0 = (1/(L0^2))*[-delta_x, -delta_y, delta_x, delta_y]';

    propno = IX(e,3);
    E0 = mprop(propno,1);
```

```
    A = mprop(propno,2);

    % local k matrix for gradient calculation
    k_0e = E0*A*L0*(B0)*(B0'); % 4x4 matrix, local

    strain(e) = (B0') * d;
    stress(e) = rho(e)^p*E0*strain(e);

    N(e) = stress(e)*A;

    % Element volume inside the respective vector:
    g_grad(e) = A*L0;

    % Calculation of the gradients of f and g:
    f_grad(e) = -p*((rho(e))^(p-1)) * (d') * k_0e * d;

    f = f + (d')*(rho(e)^p)*k_0e*d; % objective function

    % Calculation of reaction forces R:
    for jj=1:4
        B0sum(edof(jj)) = B0sum(edof(jj)) + B0(jj)*N(e)*L0;
    end

end

% Reaction forces vector:
R = B0sum - P;

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
clf
hold on
box on

color_vector = ['r', 'b', 'g']; % vector for colors of the trusses
fake_zero = 1e-10; % fake zero to enter the if cycle below

for e = 1:ne
    xx = X(IX(e,1:2),1); % x-coord. of the nodes of the selected element (bar, truss)
    yy = X(IX(e,1:2),2);
    % plot of undeformed structure!
    h1=plot(xx,yy,'k:','LineWidth',1);

    edof = [2*IX(e,1)-1 2*IX(e,1) 2*IX(e,2)-1 2*IX(e,2)];
    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    if stress(e) > fake_zero
        col = color_vector(2); % element in tension
    elseif stress(e) < - fake_zero
        col = color_vector(1); % element in compression
    else
        col = color_vector(3); % unloaded element
```

```
    end

    h2=plot(xx,yy,col,'LineWidth',7.5*rho(e));

end
plotsupports
plotloads

title('Non-Active elements are plotted in white color', 'FontSize', 16)
legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return
```

4.3 Code for testing different p values

Structure 3 with 40% connectivity has been used for $\eta \in [0.65, 0.69]$ and $p \in [1.5 : 0.2 : 1.9]$.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Basic truss program                              %
%            Exercise 4.2 - Assignment 1: different p value               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
mesh3_40

neqn = size(X,1)*size(X,2);         % Number of equations
ne = size(IX,1);                    % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

iterations = zeros(size(p_vector,2),1);
f_vector_plot = zeros(size(p_vector,2),iopt_max);

for j = 1:size(p_vector,2)

    p = p_vector(j);
    clear f_vector

%--- Initialize arrays ---------------------------------------------------%
Kmatr=sparse(neqn,neqn);                % Stiffness matrix
P=zeros(neqn,1);                        % Force vector
D=zeros(neqn,1);                        % Displacement vector
R=zeros(neqn,1);                        % Residual vector
strain=zeros(ne,1);                     % Element strain vector
stress=zeros(ne,1);                     % Element stress vector
rho = zeros(ne,1);

[~,~,~,~,~,g_grad]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

v = g_grad; % vector of element volumes and g_grad are the same

% Initialize the vector of relative element densities:
% We need the pseudo inverse matrix in order to initialize rho satisfying
% g=0:

v_sum = sum(v);
rho(:) = V_star/v_sum; % initially think rho as constant

[P]=buildload(X,IX,ne,P,loads,mprop);        % Build global load vector

for iopt = 1:iopt_max

    rho_old = rho;

    %--- Calculate displacements ---------------------------------------------%
    [Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr,p,rho_old);     % Build global stiffness matrix

    [Kmatr,P]=enforce(Kmatr,P,bound);             % Enforce boundary conditions
```

```
    D = Kmatr \ P;                                        % Solve system of equations

    [~,~,~,~,f_grad,~,~]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho_old);

    rho = bisect(rho_old,V_star,f_grad,g_grad,rho_min,eta);

    [~,~,~,~,~,~,f]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

    f_vector(iopt) = f; % vector for convergence plot, cell j and entrance i

    f_vector_plot(j,iopt) = f;

    if norm(rho_old - rho) < norm(rho)*epsilon
        break
    end

end

f_cell{j} = f_vector;
iterations(j) = iopt;

% Save f_vector for convergence plot for different eta values:

[strain,stress,~,~,~,~,~]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho);

% Check on the strain energy density = constant for active members:
energy_density = zeros(ne,1);

for e = 1:ne
    energy_density(e) = (strain(e)*stress(e))/rho(e);
end

end

iterations % stamp number of iterations for the current eta value

figure('color','white')
for s = 1:size(p_vector,2)
    x_step = [1:1:iterations(s)];
    plot(x_step,f_cell{s},'.-','MarkerSize', 8)
    legend_name(s) = strcat("p = ",num2str(p_vector(s)));
    hold on
end

hold off
grid on
set(gca,'FontAngle','oblique','FontSize',14)
legend(legend_name,'location','northeast')
title('Convergence plot for different p values', 'FontSize', 16)
xlabel('Topology Optimization iteration')
ylabel('Compliance f(\rho)')

figure('color','white')
for s = 1:size(p_vector,2)
    x_step = [1:1:iterations(s)];
    semilogy(x_step,f_cell{s},'.-','MarkerSize', 8)
    legend_name(s) = strcat("p = ",num2str(p_vector(s)));
    hold on
end

hold off
```

```
axis([0 100 0 10^(-4)])
grid on
set(gca,'FontAngle','oblique','FontSize',14)
legend(legend_name,'location','northeast')
title('Convergence plot for different p values in log scale', 'FontSize', 16)
xlabel('Topology Optimization iteration')
ylabel('Compliance f(\rho)')

% % Convergence plot:
% figure(1)
% x_step = [1:1:(iopt)];
% plot(x_step,f_vector,'-rx')
% grid on
% set(gca,'FontAngle','oblique','FontSize',14)
% title('Convergence plot', 'FontSize', 16)
% xlabel('Topology Optimization iteration')
% ylabel('Compliance f')

%--- Plot results --------------------------------------------------------%
%x_step = [1:1:100];
%save('E4_case_3.mat','f_vector_plot','x_step');

% figure
% PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)        % Plot structure

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop)

P=zeros(size(X,1)*size(X,2),1);
rowX = size(X,1);
columnX = size(X,2);

for i=1:size(loads,1)

    pos = loads(i,1)*columnX - (columnX - loads(i,2));
    P(pos, 1) = loads(i,3);

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K,p,rho)

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

    K = zeros(2*size(X,1)); % global stiffness matrix def., 6x6, memory allocation

for e=1:ne

    i = IX(e,1);
    j = IX(e,2);
    xi = X(i,1);
    yi = X(i,2);
    xj = X(j,1);
    yj = X(j,2);
```

```
    delta_x = (xj-xi);
    delta_y = (yj-yi);

    L0 = sqrt(delta_x^2 + delta_y^2);

    B0 = (1/(L0^2))*[-delta_x, -delta_y, delta_x, delta_y]';

    % material properties of the current beam
    propno = IX(e,3);
    E0 = mprop(propno,1);
    A = mprop(propno,2);

    k_0e = E0*A*L0*(B0)*(B0'); % 4x4 matrix, local

    k_e = (rho(e))^p * k_0e; % now the stiffness is a function of the design variable density

    edof = [IX(e,1)*2 - 1, IX(e,1)*2 - 0, IX(e,2)*2 - 1, IX(e,2)*2 - 0];

    for ii = 1:4
        for jj = 1:4
            K(edof(ii),edof(jj)) = K(edof(ii),edof(jj)) + k_e(ii,jj);
        end
    end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound)

% This subroutine enforces the support boundary conditions

% 0 and 1 Method:

for i=1:size(bound,1)
    node = bound(i,1); % from example1.m data
    ldof = bound(i,2);
    K(:,2*node - (2-ldof)) = 0; % zeros in the column
    K(2*node - (2-ldof),:) = 0; % zeros in the row
    K(2*node - (2-ldof),2*node - (2-ldof)) = 1; % put a 1 in the diagonal

    P(2*node - (2-ldof)) = 0; % updating P vector
end

return

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress,N,R,f_grad,g_grad,f]=recover(mprop,X,IX,D,ne,strain,stress,P,p,rho)

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
d = zeros(4,1);
strain = zeros(ne,1); % allocation strain vector
stress = zeros(ne,1); % allocation stress vector
N = zeros(ne,1); % allocation element forces vector
B0sum = zeros(2*size(X,1),1);
f_grad = zeros(ne,1);
g_grad = zeros(ne,1);
```

```
f = 0; % initialize outside of the loop

for e=1:ne
    edof = [IX(e,1)*2 - 1, IX(e,1)*2 - 0, IX(e,2)*2 - 1, IX(e,2)*2 - 0];
    % define small d
    for i=1:4
        d(i) = D(edof(i));
    end

    i = IX(e,1);
    j = IX(e,2);
    xi = X(i,1);
    yi = X(i,2);
    xj = X(j,1);
    yj = X(j,2);

    delta_x = (xj-xi);
    delta_y = (yj-yi);

    L0 = sqrt(delta_x^2 + delta_y^2);

    B0 = (1/(L0^2))*[-delta_x, -delta_y, delta_x, delta_y]';

    propno = IX(e,3);
    E0 = mprop(propno,1);
    A = mprop(propno,2);

    % local k matrix for gradient calculation
    k_0e = E0*A*L0*(B0)*(B0'); % 4x4 matrix, local

    strain(e) = (B0') * d;
    stress(e) = rho(e)^p*E0*strain(e);

    N(e) = stress(e)*A;

    % Element volume inside the respective vector:
    g_grad(e) = A*L0;

    % Calculation of the gradients of f and g:
    f_grad(e) = -p*((rho(e))^(p-1)) * (d') * k_0e * d;

    f = f + (d')*(rho(e)^p)*k_0e*d; % objective function

    % Calculation of reaction forces R:
    for jj=1:4
        B0sum(edof(jj)) = B0sum(edof(jj)) + B0(jj)*N(e)*L0;
    end

end

% Reaction forces vector:
R = B0sum - P;

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)

% This subroutine plots the undeformed and deformed structure
```

```
h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
clf
hold on
box on

color_vector = ['r', 'b', 'g']; % vector for colors of the trusses
fake_zero = 1e-10; % fake zero to enter the if cycle below

for e = 1:ne
    xx = X(IX(e,1:2),1); % x-coord. of the nodes of the selected element (bar, truss)
    yy = X(IX(e,1:2),2);
    % plot of undeformed structure!
    h1=plot(xx,yy,'k:','LineWidth',1);

    edof = [2*IX(e,1)-1 2*IX(e,1) 2*IX(e,2)-1 2*IX(e,2)];
    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    if stress(e) > fake_zero
        col = color_vector(2); % element in tension
    elseif stress(e) < - fake_zero
        col = color_vector(1); % element in compression
    else
        col = color_vector(3); % unloaded element
    end

    h2=plot(xx,yy,col,'LineWidth',7.5*rho(e));

end
plotsupports
plotloads

title('Non-Active elements are plotted in white color', 'FontSize', 16)
legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return
```

4.4 Input file for one node configuration

```
% Created with:       FlExtract v1.13
% Element type:       truss
% Number of nodes:    24
% Number of elements: 157

%clear all

eta = 0.67;
p = 1.5;
V_star = 6;
iopt_max = 100;
rho_min = 1e-6;
epsilon = 1e-5;

% Node coordinates: x, y
X = [
0     0
0     0.5
0     1
0.25  0
0.25  0.5
0.25  1
0.5   0
0.5   0.5
0.5   1
0.75  0.5
0.75  1
1     0.5
1     1
1.25  0.5
1.25  1
1.5   0
1.5   0.5
1.5   1
1.75  0
1.75  0.5
1.75  1
2     0
2     0.5
2     1
];
% Element connectivity: node1_id, node2_id, material_id
IX = [
2     1     1
3     1     1
4     1     1
5     1     1
6     1     1
7     1     1
8     1     1
9     1     1
3     2     1
4     2     1
5     2     1
6     2     1
7     2     1
8     2     1
9     2     1
10    2     1
```

11	2	1
12	2	1
13	2	1
4	3	1
5	3	1
6	3	1
7	3	1
8	3	1
9	3	1
10	3	1
11	3	1
12	3	1
13	3	1
5	4	1
6	4	1
7	4	1
8	4	1
9	4	1
11	4	1
6	5	1
7	5	1
8	5	1
9	5	1
10	5	1
11	5	1
12	5	1
13	5	1
14	5	1
15	5	1
7	6	1
8	6	1
9	6	1
10	6	1
11	6	1
12	6	1
13	6	1
14	6	1
15	6	1
8	7	1
9	7	1
9	8	1
10	8	1
11	8	1
12	8	1
13	8	1
14	8	1
15	8	1
17	8	1
18	8	1
10	9	1
11	9	1
12	9	1
13	9	1
14	9	1
15	9	1
17	9	1
18	9	1
11	10	1
12	10	1
13	10	1
14	10	1

15	10	1
17	10	1
18	10	1
20	10	1
21	10	1
12	11	1
13	11	1
14	11	1
15	11	1
17	11	1
18	11	1
20	11	1
21	11	1
13	12	1
14	12	1
15	12	1
17	12	1
18	12	1
20	12	1
21	12	1
23	12	1
24	12	1
14	13	1
15	13	1
17	13	1
18	13	1
20	13	1
21	13	1
23	13	1
24	13	1
15	14	1
17	14	1
18	14	1
20	14	1
21	14	1
23	14	1
24	14	1
17	15	1
18	15	1
19	15	1
20	15	1
21	15	1
23	15	1
24	15	1
17	16	1
18	16	1
19	16	1
20	16	1
21	16	1
22	16	1
23	16	1
24	16	1
18	17	1
19	17	1
20	17	1
21	17	1
22	17	1
23	17	1
24	17	1
19	18	1
20	18	1

```
21   18   1
22   18   1
23   18   1
24   18   1
20   19   1
21   19   1
22   19   1
23   19   1
24   19   1
21   20   1
22   20   1
23   20   1
24   20   1
22   21   1
23   21   1
24   21   1
23   22   1
24   22   1
24   23   1
];
% Element properties: Young's modulus, area
mprop = [
1    1
2    2
];
% Nodal diplacements: node_id, degree of freedom (1 - x, 2 - y), displacement
bound = [
1    1    0
1    2    0
4    1    0
4    2    0
7    1    0
7    2    0
16   1    0
16   2    0
19   1    0
19   2    0
22   1    0
22   2    0
];
% Nodal loads: node_id, degree of freedom (1 - x, 2 - y), load
loads = [
10   2    -0.01
];
% Control parameters
plotdof = 2;
```

4.5 Code for running the method on the same node configuration with different connectivities

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  RUN OVER DIFFERENT NUMBER OF NODES WITH ONLY NEIGHBOR CONNECTIONS       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
file_vector = [ ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh5.m"
  ];

file_number = size(file_vector, 2);
node_number_vector = [10 29 90 223 406]; % number of the node where the force is applied
disp_vector = zeros(1, file_number); % vector of final displacements
comple_vector = zeros(1, file_number);  % vector of final compliance
total_nodes = zeros(1, file_number); % number of nodes of the structure

for j=1:file_number
%loadfile(file_vector(j));
run(file_vector(j));
node_number = node_number_vector(j);
%ex4_1

neqn = size(X,1)*size(X,2);         % Number of equations
ne = size(IX,1);                    % Number of elements
disp(['Number of DOF ' sprintf('%d',neqn) ...
    ' Number of elements ' sprintf('%d',ne)]);

%--- Initialize arrays ---------------------------------------------------%

Kmatr=sparse(neqn,neqn);                % Stiffness matrix
P=zeros(neqn,1);                        % Force vector
D=zeros(neqn,1);                        % Displacement vector
R=zeros(neqn,1);                        % Residual vector
strain=zeros(ne,1);                     % Element strain vector
stress=zeros(ne,1);                     % Element stress vector
rho = zeros(ne,1);
rho_old = zeros(ne,1);

[~,~,~,~,~,g_grad,~] =recover(mprop,X,IX,D,ne,strain,stress,P, p, rho); % volume vector

  % initialize the density vector, supposing each element equal
  rho(:) = V_star / sum(g_grad); % v and g_grad are the same
  [P]=buildload(X,IX,ne,P,loads,mprop);       % Build global load vector

  for i=1:iopt_max
    rho_old = rho;

    [Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr,p,rho_old);     % Build global stiffness matrix

    [Kmatr,Pmatr]=enforce(Kmatr,P,bound);        % Enforce boundary conditions
```

```
    D = Kmatr \ Pmatr;                              % Solve system of equations

    [~,~,~,~,f_grad,~,~] =recover(mprop,X,IX,D,ne,strain,stress,P, p, rho_old); ...
    % compute f_grad

    rho = bisect(rho_old, V_star, f_grad, g_grad, rho_min, eta, epsilon); ...
    % compute the better guess for rho

    [~,~,~,~,~,~,f(i)] =recover(mprop,X,IX,D,ne,strain,stress,P, p, rho); ...
    % compute the compliance f

     if norm(rho_old - rho) < norm(rho) * epsilon % check the convergence
      break
     end
  end
  disp_vector(j) = D(node_number * 2); % y displacement of the node of interest
  compl_vector(j) = f(end); % final compliance of the method
  total_nodes(j) = size(X, 1); % number of nodes

end
%%
%--- Plot results ---------------------------------------------------------%
compliance_vs_node_neighbor = figure('Position', get(0, 'Screensize'));
plot(total_nodes, compl_vector, 'o', 'LineWidth', 3.5);
xlabel('Nodes')
ylabel('Compliance [J]')
title('Compliance for only neighbor connections')
set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
saveas(compliance_vs_node_neighbor, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment
compliance_vs_node_neighbor.png','png');

displacement_vs_node_neighbor = figure('Position', get(0, 'Screensize'));
plot(total_nodes, disp_vector, 'o', 'LineWidth', 3.5)
xlabel('Nodes')
ylabel('Displacement [m]')
title('Displacement for only neighbor connection')
set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
saveas(compliance_vs_node_neighbor, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment
compliance_vs_node_neighbor.png','png');
%%
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K,p,rho);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_0e = E * A * L0 * B0 * B0'; % 4x4 matrix

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build K by summing k_e
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + rho(e)^p*k_0e(ii, jj);
    end
  end

end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [strain,stress]=recover(mprop,X,IX,D,ne,strain,stress);
```

```
%
% % This subroutine recovers the element stress, element strain,
% % and nodal reaction forces
%
% % allocate memory for stress and strain vectors
% strain = zeros(ne, 1);
% stress = zeros(ne, 1);
%
% for e=1:ne
%   d = zeros(4, 1); % allocate memory for element stiffness matrix
%
%  [edof] = build_edof(IX, e); % index for buildg K
%
%   % build the matrix d from D
%   for i = 1:4
%     d(i) = D(edof(i));
%   end
%
%   % compute the bar length
%   [L0, delta_x, delta_y] = length(IX, X, e);
%
%   % displacement vector
%   B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';
%
%   % materials properties
%   propno = IX(e, 3);
%   E = mprop(propno, 1);
%   A = mprop(propno, 2);
%
%   strain(e) = B0' * d;
%   stress(e) = strain(e) * E;
%
% end
%
%
% return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress, N, R, f_grad, g_grad, f]=recover(mprop,X,IX,D,ne, ...
strain,stress,P,p,rho)

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
strain = zeros(ne, 1);
stress = zeros(ne, 1);
B0_sum = zeros(2*size(X,1), 1);
g_grad = zeros(ne, 1); % vector of volumes IT IS THE SAME AS v
f_grad = zeros(ne,1); % gradient of f
f = 0; % compliance

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end
```

```
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E0 = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_0e = E0 * A * L0 * B0 * B0'; % 4x4 matrix

  % compute the volume
  g_grad(e) = L0 * A;

  strain(e) = B0' * d;
  stress(e) = rho(e)^p*strain(e) * E0;
  N(e) = stress(e) * A;

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      B0_sum(edof(jj)) = B0_sum(edof(jj)) + B0(jj)*N(e)*L0;
  end


  f_grad(e) = -p*rho(e)^(p - 1)*d'*k_0e*d; % ATTENTION TO DIMENSION OF d

  f = f + d' * rho(e)^p * k_0e * d; % compute the compliance
end

% compute the support reactions (N)
R = B0_sum - P; % 2nnx1 (nn is node number)

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
fake_zero = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));
```

```
    % choice of thhe color according to the state
    if stress(e) > fake_zero  % tension
      col = colors(1);
    elseif stress(e) < - fake_zero  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    thick = 3.5*rho(e);

    h2=plot(xx,yy, col,'LineWidth',thick); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return
```

4.6 Code for computing different connectivities and different configurations

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  RUN OVER DIFFERENT NUMBER OF NODES AND DIFFERENT CONNECTIVITY FACTORS      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fea()

close all
clc

%--- Input file ----------------------------------------------------------%
file_vector = [
  ["C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_30.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_30.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_40.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_50.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_70.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh1_90.m"] ...

  ["C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2_30.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2_40.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2_50.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2_70.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh2_90.m"]

  ["C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3_30.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3_40.m", ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3_50.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3_70.m" ...
  "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh3_90.m"]

%    ["C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4_30.m", ...
%    "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4_40.m", ...
%    "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4_50.m" ...
%    "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4_70.m" ...
%    "C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\assignment1\ex4\mesh\mesh4_90.m"]
  ];


node_item = size(file_vector, 1);
conn_item = size(file_vector, 2);

percentage_rate = [0 30 40 50 70 90];

file_number = size(file_vector, 2);
node_number_vector = [10 29 58 223]; % number of the node where the force is applied
disp_mat = zeros(node_item, conn_item); % vector of final displacements
compl_mat = zeros(node_item, conn_item);  % vector of final compliance
total_nodes = zeros(node_item, conn_item); % number of nodes of the structure
f = zeros(node_item,4);

for n = 1:node_item % run over number of nodes
  for j = 1:conn_item % run over different connectivitues
    run(file_vector(n, j)); % load the file
    neqn = size(X,1)*size(X,2);          % Number of equations
    ne = size(IX,1);                     % Number of elements
    node_number = node_number_vector(n); % node number
    Kmatr=sparse(neqn,neqn);                 % Stiffness matrix
```

```
    P=zeros(neqn,1);                        % Force vector
    D=zeros(neqn,1);                        % Displacement vector
    R=zeros(neqn,1);                        % Residual vector
    strain=zeros(ne,1);                     % Element strain vector
    stress=zeros(ne,1);                     % Element stress vector
    rho = zeros(ne,1);
    rho_old = zeros(ne,1);

    [~,~,~,~,~,g_grad,~] =recover(mprop,X,IX,D,ne,strain,stress,P, p, rho); % volume vector

    % initialize the density vector, supposing each element equal
    rho(:) = V_star / sum(g_grad); % v and g_grad are the same
    [P]=buildload(X,IX,ne,P,loads,mprop);       % Build global load vector

    for i=1:iopt_max
      rho_old = rho;

      [Kmatr]=buildstiff(X,IX,ne,mprop,Kmatr,p,rho_old);    % Build global stiffness matrix

      [Kmatr,Pmatr]=enforce(Kmatr,P,bound);       % Enforce boundary conditions

      D = Kmatr \ Pmatr;                          % Solve system of equations

      [~,~,~,~,f_grad,~,~] = recover(mprop,X,IX,D,ne,strain,stress,P, p, rho_old); ...
      % compute f_grad

      rho = bisect(rho_old, V_star, f_grad, g_grad, rho_min, eta, epsilon); ...
      % compute the better guess for rho

      [~,~,~,~,~,~,f] = recover(mprop,X,IX,D,ne,strain,stress,P, p, rho);...
      % compute the compliance f

       if norm(rho_old - rho) < norm(rho) * epsilon % check the convergence
        break
       end
      disp_mat(n, j) = D(node_number * 2); % y displacement of the node of interest
      compl_mat(n, j) = f; % final compliance of the method
    end
  end
  total_nodes(n) = size(X, 1); % number of nodes
end

% manually add results for mesh 4
mesh4_results_disp = [0.00923402 0.00683920, 0.00633002, 0.00691394, 0.00725176, 0.00636457]; .
% displacement of mesh 4
mesh4_results_complinace = [0.00009234 6.8392e-5, 6.3300e-5, 6.9139e-5, 7.2517e-5, 6.3646e-5];
% complinace of mesh 4

%%
%--- Plot results ---------------------------------------------------------%
save("results", "compl_mat", "disp_mat", "total_nodes");
%%
res = load("results.mat")
node_item = 3;
percentage_rate = [0 30 40 50 70 90];

% manually add results for mesh 4
mesh4_results_disp = [0.00923402 0.00683920, 0.00633002, 0.00691394, 0.00725176, 0.00636457]; %
mesh4_results_complinace = [0.00009234 6.8392e-5, 6.3300e-5, 6.9139e-5, 7.2517e-5, 6.3646e-5]; ·

compl_mat = res.compl_mat;
```

```
%disp_mat = res.disp_mat;
total_nodes = res.total_nodes;
%compliance_vs_percentage_vs_node = figure('Position', get(0, 'Screensize'));
compliance_vs_percentage_vs_node = figure('Color','White');
legend_vector = strings(1, node_item);

for i =1:node_item
  plot(percentage_rate, compl_mat(i,:),  '-', 'LineWidth',1,...
  'Marker', '.', 'MarkerSize',35);
  legend_vector(i) = strcat("Structure ", num2str(i), ": ", num2str(total_nodes(i)), " nodes");
  hold on
end
plot(percentage_rate, mesh4_results_complinace, '-',  'LineWidth',1, 'Marker', '.', 'MarkerSize
grid on
hold off
xlabel('Connectivity [%]')
ylabel('Compliance [J]')
legend_vector(end + 1) = "Structure 4: 537 nodes";
legend(legend_vector)
legend(legend_vector)
title('Compliance for different nodes and connectivity radius', 'FontSize',16)
set(gca, 'FontAngle', 'oblique', 'FontSize', 14)
saveas(compliance_vs_percentage_vs_node , 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM ...
\assignment1\compliance_vs_percentage_vs_node.png','png');
%
% displacement_vs_percentage_vs_node  = figure('Position', get(0, 'Screensize'));
% for i=1:node_item
%   plot(percentage_rate, disp_mat(i,:), 'o', 'LineWidth', 3.5)
%   hold on
% end
% hold off
% xlabel('Nodes')
% ylabel('Displacement [m]')
% legend(legend_vector)
% title('Displacement for 71 nodes - different connection percentage')
% set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
% saveas(displacement_vs_percentage_vs_node, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\a
% marker = ['o', '+', '*', 'x' ]; %markers for different plots
% legend_vector = strings(1, node_item);
% displacement_vs_percentage_vs_node  = figure('Position', get(0, 'Screensize'));
% for i=1:node_item
%   yyaxis left
%   plot(percentage_rate, compl_mat(i,:), marker(i), 'LineWidth', 1.5);
%   hold on
%   legend_vector(i) = strcat(num2str(total_nodes(i)), " nodes");
% end
% for i = 1:node_item
%   yyaxis right
%   plot(percentage_rate, abs(disp_mat(i,:)), marker(i), 'LineWidth', 1.5)
%   hold on
% end
% yyaxis left
% plot(percentage_rate, mesh4_results_complinace, marker(end), 'LineWidth', 1.5);
% yyaxis right
% plot(percentage_rate, mesh4_results_disp, marker(end), 'LineWidth', 1.5);
% hold off
% xlabel('Nodes')
% yyaxis right
% ylabel('Displacement [m]')
% yyaxis left
% ylabel('Compliance [J]')
```

```
% legend_vector(end + 1) = "537 nodes";
% legend(legend_vector)
% title('Compliance and disp. for different nodes and connectivities')
% set(gca, 'FontAngle', 'oblique', 'FontSize', 20)
% saveas(displacement_vs_percentage_vs_node, 'C:\Users\Niccolo\Documents\UNIVERSITA\5ANNO\FEM\a

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global load vector %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P]=buildload(X,IX,ne,P,loads,mprop);

rowX = size(X, 1);
columnX = size(X, 2);

% P contains the external forces applied by the load
% P is a 2nnx1 matrix (nn is the number of nodes)
P = zeros(columnX * rowX, 1); % assign the memory for P

for i=1:size(loads,1)
  pos = loads(i, 1)*columnX - (columnX - loads(i, 2));
  P(pos, 1) = loads(i, 3);
end


return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Build global stiffness matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K]=buildstiff(X,IX,ne,mprop,K,p,rho);

% This subroutine builds the global stiffness matrix from
% the local element stiffness matrices

K = zeros(2*size(X, 1)); % allocate memory

for e = 1:ne % cycle on the different bar
  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_0e = E * A * L0 * B0 * B0'; % 4x4 matrix

  % vector of index used for building K
  [edof] = build_edof(IX, e);

  % build K by summing k_e
  for ii = 1:4
    for jj = 1:4
      K(edof(ii), edof(jj)) = K(edof(ii), edof(jj)) + rho(e)^p*k_0e(ii, jj);
    end
  end
```

```
end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Enforce boundary conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [K,P]=enforce(K,P,bound);

% This subroutine enforces the support boundary conditions

% 0-1 METHOD
for i=1:size(bound,1)
    node = bound(i, 1); % number of node
    ldof = bound(i, 2); % direction
    pos = 2*node - (2 - ldof);
    K(:, pos) = 0; % putting 0 on the columns
    K(pos, :) = 0; % putting 0 on the rows
    K(pos, pos) = 1; % putting 1 in the dyagonal

    P(pos) = 0; % putting 0 in P
end

return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [strain,stress]=recover(mprop,X,IX,D,ne,strain,stress);
%
% % This subroutine recovers the element stress, element strain,
% % and nodal reaction forces
%
% % allocate memory for stress and strain vectors
% strain = zeros(ne, 1);
% stress = zeros(ne, 1);
%
% for e=1:ne
%   d = zeros(4, 1); % allocate memory for element stiffness matrix
%
%  [edof] = build_edof(IX, e); % index for buildg K
%
%   % build the matrix d from D
%   for i = 1:4
%     d(i) = D(edof(i));
%   end
%
%   % compute the bar length
%   [L0, delta_x, delta_y] = length(IX, X, e);
%
%   % displacement vector
%   B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';
%
%   % materials properties
%   propno = IX(e, 3);
%   E = mprop(propno, 1);
%   A = mprop(propno, 2);
%
%   strain(e) = B0' * d;
%   stress(e) = strain(e) * E;
%
% end
%
```

```
%
% return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Calculate element strain and stress %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [strain,stress, N, R, f_grad, g_grad, f]=recover(mprop,X,IX,D,ne, ...
strain,stress,P,p,rho)

% This subroutine recovers the element stress, element strain,
% and nodal reaction forces
strain = zeros(ne, 1);
stress = zeros(ne, 1);
B0_sum = zeros(2*size(X,1), 1);
g_grad = zeros(ne, 1); % vector of volumes IT IS THE SAME AS v
f_grad = zeros(ne,1); % gradient of f
f = 0; % compliance

for e=1:ne
  d = zeros(4, 1);
  [edof] = build_edof(IX, e);

  % build the matrix d from D
  for i = 1:4
    d(i) = D(edof(i));
  end

  % compute the bar length
  [L0, delta_x, delta_y] = length(IX, X, e);
  % displacement vector (4x1)
  B0 = 1/L0^2 * [-delta_x -delta_y delta_x delta_y]';

  % materials properties
  propno = IX(e, 3);
  E0 = mprop(propno, 1);
  A = mprop(propno, 2);

  % element stiffness matrix
  k_0e = E0 * A * L0 * B0 * B0'; % 4x4 matrix

  % compute the volume
  g_grad(e) = L0 * A;

  strain(e) = B0' * d;
  stress(e) = rho(e)^p*strain(e) * E0;
  N(e) = stress(e) * A;

  % sum B0 after having transformed it in order to be compliant for the sum
  % with P
  for jj = 1:4
      B0_sum(edof(jj)) = B0_sum(edof(jj)) + B0(jj)*N(e)*L0;
  end


  f_grad(e) = -p*rho(e)^(p - 1)*d'*k_0e*d; % ATTENTION TO DIMENSION OF d

  f = f + d' * rho(e)^p * k_0e * d; % compute the compliance
end

% compute the support reactions (N)
R = B0_sum - P; % 2nnx1 (nn is node number)
```

```
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot structure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PlotStructure(X,IX,ne,neqn,bound,loads,D,stress,rho)

% This subroutine plots the undeformed and deformed structure

h1=0;h2=0;
% Plotting Un-Deformed and Deformed Structure
clf
hold on
box on

colors = ['b', 'r', 'g']; % vector of colors for the structure
fake_zero = 1e-8; % fake zero for tension sign decision

for e = 1:ne
    xx = X(IX(e,1:2),1); % vector of x-coords of the nodes
    yy = X(IX(e,1:2),2); % vector of y-coords of the nodes
    h1=plot(xx,yy,'k:','LineWidth',1.); % Un-deformed structure
    [edof] = build_edof(IX, e);

    xx = xx + D(edof(1:2:4));
    yy = yy + D(edof(2:2:4));

    % choice of thhe color according to the state
    if stress(e) > fake_zero  % tension
      col = colors(1);
    elseif stress(e) < - fake_zero  % compression
      col = colors(2);
    else col = colors(3);  % un-loaded
    end

    thick = 3.5*rho(e);

    h2=plot(xx,yy, col,'LineWidth',thick); % Deformed structure
end
plotsupports
plotloads

legend([h1 h2],{'Undeformed state',...
                'Deformed state'})

axis equal;
hold off

return

%% Function to compute the length of a bar
function [L0, delta_x, delta_y] = length(IX, X, e)

  i = IX(e, 1);
  j = IX(e, 2);
  xi = X(i, 1);
  xj = X(j, 1);
  yi = X(i, 2);
  yj = X(j, 2);
  delta_x = xj - xi;
  delta_y = yj - yi;
```

```
  L0 = sqrt(delta_x^2 + delta_y^2);

return

%% Function to provide the edof vector
function [edof] = build_edof(IX, e)
  edof = zeros(4, 1);
  edof = [IX(e,1)*2-1 IX(e,1)*2 IX(e,2)*2-1 IX(e,2)*2];
return
```