ME471 FEA Code Assignment

```
clearvars; close all;
filename= 'C:\Users\Joshu\OneDrive\Documents\College\2024 Fall\ME471\Correct
Code\Input_File_FEA.txt';
```

Txt File Converter Function

```
function tablesToMatrices(filename)
    % Open the file for reading
    fid = fopen(filename, 'r');
    if fid == -1
        error('Could not open the file.');
    end

    % Initialize variables
    tableNames = {};
    tableData = {};
    currentData = [];
    currentTableName = '';

    % Read the file line by line
    while ~feof(fid)
        line = strtrim(fgets(fid)); % Read a line and trim whitespace

        if isempty(line)
            continue; % Skip empty lines
        end

        % Check if the line is a table name (non-numeric)
        if isletter(line(1)) && all(isstrprop(line, 'alphanum'))
            % Save the current table data if a new table starts
            if ~isempty(currentTableName)
                tableNames{end+1} = currentTableName;
                tableData{end+1} = currentData;
            end

            % Update the current table name
            currentTableName = matlab.lang.makeValidName(line); % Make a valid
MATLAB variable name
            currentData = []; % Reset data for the new table
        else
            % Parse numeric data in the line
            numericRow = str2num(line); %#ok<ST2NM>
            if ~isempty(numericRow)
                currentData = [currentData; numericRow]; %#ok<AGROW>
            end
        end
```

```
    end

    % Add the last table
    if ~isempty(currentTableName)
        tableNames{end+1} = currentTableName;
        tableData{end+1} = currentData;
    end

    fclose(fid); % Close the file

    % Assign matrices to the base workspace
    for k = 1:length(tableNames)
        assignin('base', tableNames{k}, tableData{k});
        fprintf('Matrix "%s" created in the workspace.\n', tableNames{k});
    end
end
```

File Import

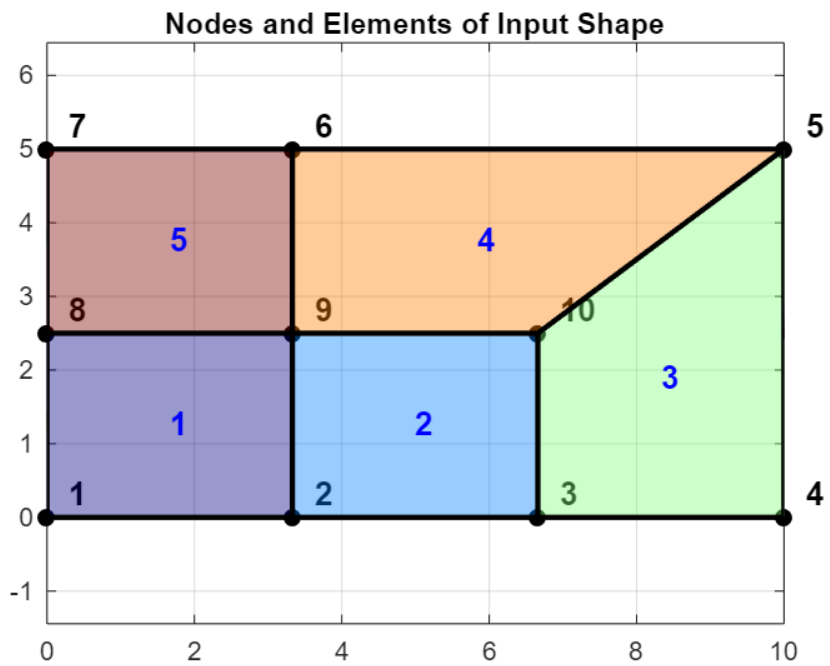
```
tablesToMatrices(filename);
```

```
Matrix "Nodes" created in the workspace.
Matrix "Elements" created in the workspace.
Matrix "Material" created in the workspace.
Matrix "Boundary" created in the workspace.
Matrix "Loading" created in the workspace.
```

Plotting for visual inspection

```
% Extract node coordinates
x_coords = Nodes(:,2);
y_coords = Nodes(:,3);

% Create figure for the plot
figure;

% Plot the nodes (use black circles for the nodes)
plot(x_coords, y_coords, 'ko', 'MarkerFaceColor', 'k');
hold on;

% Number each node with a greater offset
nodeOffset = 0.3;  % Adjust this value for greater or lesser spacing
for i = 1:size(Nodes, 1)
    % Get the coordinates of the node
    x = Nodes(i, 2);
    y = Nodes(i, 3);
    % Place the node number farther from the node (increased offset)
    text(x + nodeOffset, y + nodeOffset, num2str(Nodes(i, 1)), 'FontSize', 12,
'FontWeight', 'bold');
end
```

```
% Generate colors for the elements
colormap(jet);  % Choose a color map (e.g., 'jet', 'hsv', 'parula', etc.)
colors = colormap;  % Store the colormap colors
num_elements = size(Elements, 1);
element_colors = linspace(1, size(colors, 1), num_elements);  % Assign a unique
color to each element

% Plot the elements and color code them with transparency
for i = 1:size(Elements, 1)
    % Create coordinates for the element
    x_elem = Nodes(Elements(i, 2:end), 2);  % Get the x-coordinates of the
element's nodes
    y_elem = Nodes(Elements(i, 2:end), 3);  % Get the y-coordinates of the
element's nodes

    % Close the element by adding the first node to the end
    x_elem = [x_elem; x_elem(1)];
    y_elem = [y_elem; y_elem(1)];

    % Assign color to the element based on its number
    color_idx = round(element_colors(i));  % Get the color index for the current
element
    element_color = colors(color_idx, :);  % Get the RGB color value

    % Plot the element with semi-transparent color
    fill(x_elem, y_elem, element_color, 'FaceAlpha', 0.4, 'EdgeColor', 'none'); %
'FaceAlpha' controls transparency

    % Add solid line segments connecting the nodes of the element
    plot(x_elem, y_elem, 'Color', 'k', 'LineWidth', 2);  % Solid black lines
connecting the nodes
end

% Plot the element numbers in the center
for i = 1:size(Elements, 1)
    % Calculate the center of the element for numbering
    x_center = mean(Nodes(Elements(i, 2:end), 2));
    y_center = mean(Nodes(Elements(i, 2:end), 3));

    % Place the element number in the center
    text(x_center, y_center, num2str(Elements(i, 1)), 'FontSize', 12, 'FontWeight',
'bold', 'Color', 'blue');
end

% Final plot formatting
axis equal;
grid on;
title('Nodes and Elements of Input Shape');
hold off;
```



Nodes and Elements

```
fprintf('Nodes in the following order: \n [Node Number, X-Coordinate, Y-Coordinate]
\n'); disp(Nodes)
```

```
Nodes in the following order:
 [Node Number, X-Coordinate, Y-Coordinate]
    1.0000         0         0
    2.0000    3.3300         0
    3.0000    6.6600         0
    4.0000   10.0000         0
    5.0000   10.0000    5.0000
    6.0000    3.3300    5.0000
    7.0000         0    5.0000
    8.0000         0    2.5000
    9.0000    3.3300    2.5000
   10.0000    6.6600    2.5000
```

```
fprintf('Elements in the following order: \n [Element Number, Node 1, Node 2, Node
3, Node 4] \n'); disp(Elements)
```

```
Elements in the following order:
 [Element Number, Node 1, Node 2, Node 3, Node 4]
     1     1     2     9     8
     2     2     3    10     9
     3     3     4     5    10
     4     9    10     5     6
     5     8     9     6     7
```

Modulus Matrix

```
E= Material(1); %Material Modulus of Elasticity in MPa
v= Material(2); %Material poisson ratio
```

```
Th= Material(3); %Material Thickness

E_Matrix= (E/((1+v)*(1-(2*v))))*[1-v v 0;
                                v 1-v 0;
                                0 0 (1-2*v)/2]; %Modulus of Elasticity Matrix for
plain strain

fprintf('E_Matrix = [ %.2f  %.2f  %.2f\n', E_Matrix(1,1), E_Matrix(1,2),
E_Matrix(1,3));
```

```
E_Matrix = [ 1346.15  576.92  0.00
```

```
fprintf('             %.2f   %.2f %.2f\n', E_Matrix(2,1), E_Matrix(2,2),
E_Matrix(2,3));
```

```
             576.92  1346.15 0.00
```

```
fprintf('             %.2f     %.2f    %.2f] [MPa]', E_Matrix(3,1), E_Matrix(3,2),
E_Matrix(3,3));
```

```
             0.00     0.00    384.62] [MPa]
```

Global Stiffness

```
elmSize= size(Elements,2)-1; %size of each element
dof= 2*size(Nodes, 1); %Total Degrees of Freedom for the system

guassQty= 4;   %Number of Guass Points
g= 1/sqrt(3);  %Variable to simplify Guass Point Calculation
guassPts= [g g; -g g; -g -g; g, -g];  %Guass Points for 4 node element and 4 points
kGlobal= zeros(dof); %Global stiffness matrix population
kLocalGuass= cell(size(Elements,1), guassQty); %Cell matrix to store local
stiffness at guass points

for i=1:size(Elements,1) %increments through all elements
    xCoord= []; %X-coordinates present in 1 element
    yCoord= []; %Y-coordinates present in 1 element
    nodeOrder= []; %nodal ordering in element for reconstruction
    for j=2:size(Elements,2) %increments through each individual node
        nodeNum= Elements(i,j);  %Node of interest
        xCoord= [xCoord, Nodes(nodeNum,2)]; %Adds x-coordinate to matrix for
specific element
        yCoord= [yCoord, Nodes(nodeNum,3)]; %Adds y-coordinate to matrix for
specific element
        nodeOrder= [nodeOrder,2*nodeNum-1, 2*nodeNum]; %creates a nodal order to
assemble global stiffness
    end
    kLocal=0;
    for G=1:guassQty %Guass Quad integration increment
        z= guassPts(G,1); %zeta value for specific guass point
```

```
        n= guassPts(G,2); %eta value for specific guass point
        dNdz= .25*[-1+n 1-n 1+n -1-n]; % shape function derivative w.r.t zeta in 
row vector
        dNdn= .25*[-1+z -1-z 1+z 1-z]; % shape function derivative w.r.t eta in row 
vector
        Jac= [dot(xCoord, dNdz) dot(yCoord, dNdz);
              dot(xCoord, dNdn) dot(yCoord, dNdn)]; %Jacobian based on specific 
point and element
        detJac= det(Jac); %Determinate of the Jacobian
        invJ= inv(Jac); %inverse of the Jacobian
        dN = struct(); %creates a structured array for dN1,dN2...
        for m = 1:elmSize
            dN.(sprintf('dN%d', m)) = invJ * [dNdz(1, m); dNdn(1, m)]; %stores 
derivitives w.r.t. x & y
        end
        B = zeros(3, 2 * elmSize); %creates and populates B matrix
        for p = 1:elmSize %Increments through each element based on element size
            colX = 2 * p - 1; % Counts every other column for the B matrix
            colY = 2 * p;     % Counts every other column for the b matrix

            % Populate the B matrix
            B(1, colX) = dN.(sprintf('dN%d', p))(1); % dNn/dx in B(1,:)
            B(2, colY) = dN.(sprintf('dN%d', p))(2); % dNn/dy in B(2,:)
            B(3, colX) = dN.(sprintf('dN%d', p))(2); % dNn/dy in B(3,:)
            B(3, colY) = dN.(sprintf('dN%d', p))(1); % dNn/dx in B(3,:)
        end
        kLocal= kLocal+(B'*E_Matrix*B*detJac); %solves for local stiffness matrix 
for each guass point and adds together
        kLocalGuass{i,G}=kLocal; %Stores local stiffness matrices for future stress 
calculations
    end
    %Jac
    for s=1:size(kLocal,1) %increments along the rows in local stiffness
        for t=1:size(kLocal,2) %increments along the columns in local stiffness
            kElement= kLocal(s,t); %isolates specific value from local stiffness
            kGlobal(nodeOrder(1,s),nodeOrder(1,t))=
kGlobal(nodeOrder(1,s),nodeOrder(1,t))+kElement; %stores value based on nodeOrder 
in global stiffness matrix
        end
    end
end
kGlobal = kGlobal*Th; %Modifies global stiffness matrix to account for plate 
thickness
disp('Global Stiffness Matrix');disp(kGlobal)
```

```
Global Stiffness Matrix
   1.0e+04 *

    0.2538    0.1202   -0.1257    0.0240         0         0         0         0         0         0         0
    0.1202    0.3470   -0.0240    0.1013         0         0         0         0         0         0         0
   -0.1257   -0.0240    0.5076         0   -0.1257    0.0240         0         0         0         0         0
```

```
    0.0240    0.1013         0    0.6939   -0.0240    0.1013         0         0         0         0         0
         0         0   -0.1257   -0.0240    0.5361   -0.0647   -0.2029   -0.0037   -0.0490   -0.0925         0
         0         0    0.0240    0.1013   -0.0647    0.6500   -0.0518    0.0373   -0.0925   -0.1093         0
         0         0         0         0   -0.2029   -0.0518    0.4176   -0.1664    0.0862    0.0703         0
         0         0         0         0   -0.0037    0.0373   -0.1664    0.2781    0.0222   -0.1342         0
         0         0         0         0   -0.0490   -0.0925    0.0862    0.0222    0.3218    0.1478   -0.0280    0.
         0         0         0         0   -0.0925   -0.1093    0.0703   -0.1342    0.1478    0.4406    0.0223    0.
         0         0         0         0         0         0         0         0   -0.0280    0.0223    0.5384   -0.
         0         0         0         0         0         0         0         0    0.0703    0.2138   -0.0463    1.
         0         0         0         0         0         0         0         0         0         0   -0.1257   -0.
         0         0         0         0         0         0         0         0         0         0    0.0240    0.
   -0.0012    0.0240   -0.1269    0.1202         0         0         0         0         0         0   -0.1269   -0.
   -0.0240   -0.2748    0.1202   -0.1735         0         0         0         0         0         0   -0.1202   -0.
   -0.1269   -0.1202   -0.0023         0   -0.1269    0.1202         0         0   -0.0680   -0.0924   -0.0612   -0.
   -0.1202   -0.1735         0   -0.5496    0.1202   -0.1735         0         0   -0.0924   -0.0509   -0.0278   -0.
         0         0   -0.1269   -0.1202   -0.0316    0.0647   -0.3009    0.1479   -0.2629   -0.0555   -0.1965    0.
         0         0   -0.1202   -0.1735    0.0647   -0.5058    0.1479   -0.1813   -0.0555   -0.3601    0.1480   -0.
```

Load Vector Calculation

```
rGlobal= zeros(dof, 1); %Defines and populates global load vector
for i=1:size(Loading,1) %incremenets through inputted load vector
    direct= Loading(i,2)-2; %Accounts for direction specified (either direction 1
or 2)
    rGlobal(2*Loading(i,1)+direct,1)=rGlobal(2*Loading(i,1)+direct,1)+Loading(i,3);
%stores inputed value in global load vector
end
disp('Global Load Vector'); disp(rGlobal)
```

```
Global Load Vector
         0
         0
         0
         0
         0
         0
   18.0000
         0
   42.0000
  333.0000
         0
  499.5000
         0
  166.5000
         0
         0
         0
         0
         0
         0
```

Boundary Conditions

```
zeroBoundary= []; %creates a matrix for zero displacement boundary conditions
for i=1:size(Boundary,1) %increments through boundary condition matrix
    direct= Boundary(i,2)-2; %accounts for specified direction (1 or 2)
```

```
    zeroBoundary= [zeroBoundary; 2*Boundary(i,1)+direct]; %records dof number of
zero displacement boundary conditions
end
remainNodes= setdiff((1:dof)', zeroBoundary); %removes the zero displacement
boundary condition dof number
rReduced= rGlobal(remainNodes); %creates a reduced global load vector
kReduced= kGlobal(remainNodes,remainNodes); %creates a reduced global stiffness
matrix
```

Finding Displacements

```
uReduced= kReduced \ rReduced; %solves to find displacements in reduced dof
uGlobal= zeros(dof,1); %creates and populates global displacements
uGlobal(remainNodes)= uReduced; %adds reduced matrix displacement values to a
global sized matrix
uNodal=zeros(dof/2,3); %creates and populates a matrix for nodal ordered
displacement
for i=1:size(Nodes,1) %increments through each node
    uNodal(i,1)=i; %stores node number
    uNodal(i,2)= uGlobal(i*2-1,1); %stores x displacement
    uNodal(i,3)= uGlobal(i*2,1); %stores y displacement
end
fprintf('Nodal Displacement in the following order: \n [Node Number, X-
Displacement, Y-Displacement \n'); disp(uNodal)
```

```
Nodal Displacement in the following order:
 [Node Number, X-Displacement, Y-Displacement
    1.0000         0         0
    2.0000   -0.0193         0
    3.0000   -0.0395         0
    4.0000   -0.0613         0
    5.0000   -0.0508    0.0849
    6.0000   -0.0185    0.0870
    7.0000         0    0.0867
    8.0000         0    0.0434
    9.0000   -0.0188    0.0439
   10.0000   -0.0377    0.0436
```

Finding Reaction Forces

```
reactGlobal= kGlobal*uGlobal-rGlobal; %solves for global reaction forces
reactNodal= zeros(dof/2, 3); %creates and populates matrix for nodal reaction forces
for i=1:size(Nodes,1) %increments through each node
    reactNodal(i,1)= i; %stores node number
    reactNodal(i,2)= reactGlobal(i*2-1,1); %stores reaction froce in x direction
    reactNodal(i,3)= reactGlobal(i*2,1); %stores reaction force in y direction
end
fprintf('Reaction Forces in the following order: \n [Node Number, X-Force, Y-Force]
\n'); disp(reactNodal)
```

```
Reaction Forces in the following order:
 [Node Number, X-Force, Y-Force]
```

```
     1.0000  -15.0997 -168.1670
     2.0000   -0.0000 -337.4133
     3.0000    0.0000 -336.9035
     4.0000         0 -156.5162
     5.0000   -0.0000    0.0000
     6.0000    0.0000   -0.0000
     7.0000  -14.7758         0
     8.0000  -30.1245    0.0000
     9.0000   -0.0000    0.0000
    10.0000    0.0000   -0.0000
```

Stresses in Node A

```
ElementA= 3; %Element of Interest
nodeNumbersA= Elements(ElementA,2:5); %node numbers of specific element of interest
xCoordA= Nodes(nodeNumbersA,2); %individual x coordinates of all nodes in element
yCoordA= Nodes(nodeNumbersA,3); %individual y coordinates of all nodes in element
labels= [];

uElement= zeros(2*size(nodeNumbersA,2),1); %displacement vector for element of
interest
for i=1:size(nodeNumbersA,2)
    uElement(2*i-1)=uGlobal(2*nodeNumbersA(1,i)-1);
    uElement(2*i)= uGlobal(2*nodeNumbersA(1,i));
end
guassQty= 4;   %Number of Guass Points
g= 1/sqrt(3);  %Variable to simplify Guass Point Calculation
guassPts= [g g; -g g; -g -g; g, -g];  %Guass Points for 4 node element and 4 points
for G=1:guassQty %Guass Quad integration increment
        z= guassPts(G,1); %zeta value for specific guass point
        n= guassPts(G,2); %eta value for specific guass point
        dNdz= .25*[-1+n 1-n 1+n -1-n]; % shape function derivative w.r.t zeta in
row vector
        dNdn= .25*[-1+z -1-z 1+z 1-z]; % shape function derivative w.r.t eta in row
vector
        Jac= [dot(xCoordA, dNdz) dot(yCoordA, dNdz);
              dot(xCoordA, dNdn) dot(yCoordA, dNdn)]; %Jacobian based on specific
point and element
        detJac= det(Jac); %Determinate of the Jacobian
        invJ= inv(Jac); %inverse of the Jacobian
        dN = struct(); %creates a structured array for dN1,dN2...
        for m = 1:elmSize
            dN.(sprintf('dN%d', m)) = invJ * [dNdz(1, m); dNdn(1, m)]; %stores
derivitives w.r.t. x & y
        end
        B = zeros(3, 2 * elmSize); %creates and populates B matrix
        for p = 1:elmSize %Increments through each element based on element size
            colX = 2 * p - 1; % Counts every other column for the B matrix
            colY = 2 * p;     % Counts every other column for the b matrix

            % Populate the B matrix
```

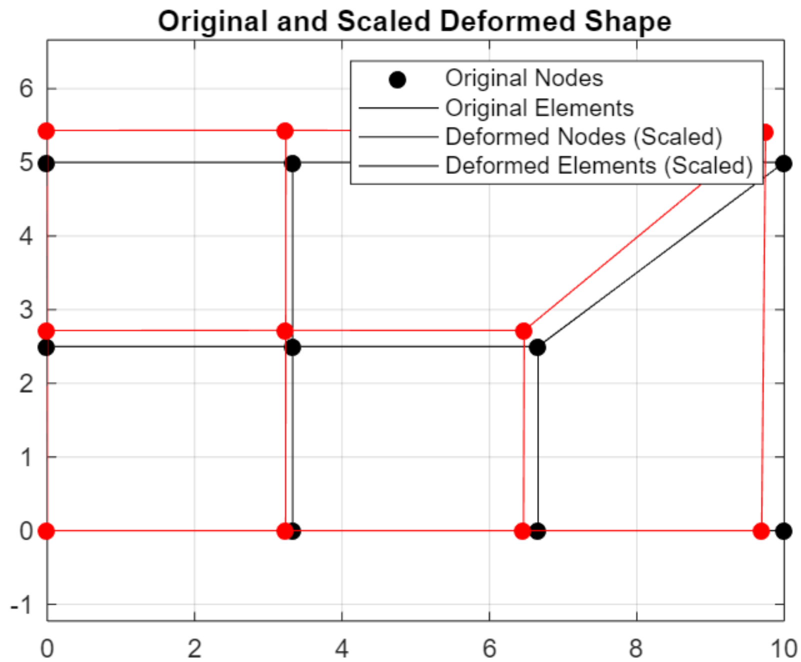
```
            B(1, colX) = dN.(sprintf('dN%d', p))(1); % dNn/dx in B(1,:)
            B(2, colY) = dN.(sprintf('dN%d', p))(2); % dNn/dy in B(2,:)
            B(3, colX) = dN.(sprintf('dN%d', p))(2); % dNn/dy in B(3,:)
            B(3, colY) = dN.(sprintf('dN%d', p))(1); % dNn/dx in B(3,:)
        end
    strainGPt= B*uElement; %strain calculated at each Guass Point
    stressGPt= E_Matrix* strainGPt; %stress calcualted at each Guass Point
    stressElement(G,:)= stressGPt'; %Matrix of all stresses at Guass points
    labels= [labels; z, n]; %labels of locations in terms of zeta and eta
end
for i = 1:guassQty
    fprintf('Zeta Location: %f, Eta Location: %f, stress Sigma11: %f, stress
Sigma22: %f, \n', ...
        labels(i, 1), labels(i, 2), stressElement(i, 1), stressElement(i, 2));
end
```

```
Zeta Location: 0.577350, Eta Location: 0.577350, stress Sigma11: 2.268124, stress Sigma22: 19.684456,
Zeta Location: -0.577350, Eta Location: 0.577350, stress Sigma11: 3.000869, stress Sigma22: 20.276807,
Zeta Location: -0.577350, Eta Location: -0.577350, stress Sigma11: 1.670427, stress Sigma22: 19.706618,
Zeta Location: 0.577350, Eta Location: -0.577350, stress Sigma11: 1.367124, stress Sigma22: 19.298313,
```

Plotting of Deformed Shape

```
% Define the scaling factor (adjust this value for desired magnification)
scale_factor = 5;  % Example: 10 times larger deformation for visualization

% Extract node coordinates
x_coords = Nodes(:,2);
y_coords = Nodes(:,3);

% Extract displacements
x_displacements = uGlobal(1:2:end); % X-direction displacements
y_displacements = uGlobal(2:2:end); % Y-direction displacements

% Apply the scaling factor to the displacements
x_deformed = x_coords + scale_factor * x_displacements;
y_deformed = y_coords + scale_factor * y_displacements;

% Create figure for the plot
figure;

% Plot the original shape (nodes and elements)
plot(x_coords, y_coords, 'ko', 'MarkerFaceColor', 'k'); % Original nodes as black
circles
hold on;

% Plot the elements for the original shape
for i = 1:size(Elements, 1)
    for j = 2:size(Elements, 2)
        node_start = Elements(i, j);
        if j > 1 && j < size(Elements, 2)
```

```
            node_stop = Elements(i, j + 1);
        end
        if j == size(Elements, 2)
            node_stop = Elements(i, 2);
        end
        % Original coordinates of nodes
        x_start = Nodes(node_start, 2);
        x_stop = Nodes(node_stop, 2);
        y_start = Nodes(node_start, 3);
        y_stop = Nodes(node_stop, 3);
        plot([x_start, x_stop], [y_start, y_stop], 'k'); % Plot original element
lines in black
    end
end

% Plot the deformed shape (nodes and elements)
% Deformed coordinates are already scaled by the factor
% Plot the deformed nodes as red circles
plot(x_deformed, y_deformed, 'ro', 'MarkerFaceColor', 'r'); % Deformed nodes in red

% Plot the deformed elements
for i = 1:size(Elements, 1)
    for j = 2:size(Elements, 2)
        node_start = Elements(i, j);
        if j > 1 && j < size(Elements, 2)
            node_stop = Elements(i, j + 1);
        end
        if j == size(Elements, 2)
            node_stop = Elements(i, 2);
        end
        % Deformed coordinates of nodes
        x_start_deformed = x_deformed(node_start);
        x_stop_deformed = x_deformed(node_stop);
        y_start_deformed = y_deformed(node_start);
        y_stop_deformed = y_deformed(node_stop);
        plot([x_start_deformed, x_stop_deformed], [y_start_deformed,
y_stop_deformed], 'r'); % Plot deformed elements in red
    end
end

% Final plot formatting
axis equal;
grid on;
title('Original and Scaled Deformed Shape');
legend('Original Nodes', 'Original Elements', 'Deformed Nodes (Scaled)', 'Deformed
Elements (Scaled)');
hold off;
```



TEST MATERIAL