Development of Finite Element Code for Analysis of A Three-Dimensional Isotropic Elastostatic Body

SAGAR BHATT
Person Number: 50170651
Department of Mechanical and Aerospace Engineering,
University at Buffalo

CONTENTS

I Introduction 3
   I Problem Set Up .................................................. 3

II Method of Solution 4
   I 8-Noded Hexahedral Element .................................. 4
   II Governing Equations ........................................... 5
      II.1 Shape Functions ........................................... 5
      II.2 Jacobian .................................................. 6
      II.3 Strain Displacement Matrix ............................... 7
      II.4 Constitutive Matrix: .................................... 8
      II.5 Stiffness Matrix .......................................... 8
      II.6 Energy Norm: ........................................... 8
      II.7 Boundary Conditions: ................................... 9
      II.8 Data Required: .......................................... 9

III Results 9
   I Rectangular Beam ............................................. 9
   II I-Beam ..................................................... 11
   III Error in energy norm: .................................... 12

IV Summary: 13

V Appendix 14
   I FE Code: ................................................... 14
   II Code to plot the mesh: .................................... 26
List of Figures

1 A Cantilever Beam ............................................................... 4
2 A Hexahedron Element .......................................................... 4
3 Mapping of a hexahedral element to intrinsic space ....................... 5
4 Unloaded Beam ...................................................................... 9
5 Deformed beam under load ...................................................... 10
6 Displacement Contour plotted by ABAQUS ................................. 10
7 Unloaded Beam ...................................................................... 11
8 Deformed beam under load ...................................................... 11
9 Top view of the deformed beam under load ................................. 11
10 Displacement Contour plotted by ABAQUS ............................... 12
11 Error in energy norm .............................................................. 13
Abstract

The aim of this project is to develop a code to perform finite element analysis of a three-dimensional isotropic elastostatic body by developing an eight-node linear isoparametric hexahedral element. The code was developed using MATLAB and analysis was performed on a cantilever rectangular beam and an I-beam. The results were compared with the results obtained from ABAQUS to verify the accuracy of the solution. The deflections were plotted for both problems based on the results obtained from the code as well as ABAQUS. The code was also used to study the error in energy norm for the rectangular beam.

I. Introduction

A hexahedron is to a quadrilateral what a tetrahedron is to a triangle. A hexahedron is topologically equivalent to a cube. It has eight corners, twelve edges or sides, and six faces. Finite elements with this geometry are extensively used in modeling three-dimensional solids. Hexahedra also have been the motivating factor for the development of “Ahmad-Pawsey” shell elements through the use of the “degenerated solid” concept.[2]

Introduction of isoparametric element formulation in 1968 by Bruce Irons was one of the most important contributions to the field of Finite Elements because it gave us the tools to overcome the complexity of dealing with the consistency requirements for higher order elements with curved boundaries. The same shape functions are used to interpolate the nodal coordinates and displacements. The whole element is transformed into an ideal element (e.g., a square element) by mapping it into a different coordinate system. The shape functions are then defined for this idealized element. Here, this formulation is used in three dimensions to formulate the governing equations for the C3D8 element (8-noded brick element or Hexahedron).

I. Problem Set Up

Two problems were solved using the code. One involved a 3D rectangular cantilever beam and the other involved an I-beam. In both the cases, the load was applied on one edge of the beam whereas the opposite face was fixed. The rectangular beam is depicted in fig. 1.
II. Method of Solution

I. 8-Noded Hexahedral Element

- Topology Equivalent to a cube
- The isoparametric coordinates or natural coordinates for this geometry are called $\xi, \eta$ and $\mu \in (-1, 1)$
- As in the case of quadrilaterals, this particular choice of limits was made to facilitate the use of the standard Gauss integration formulas.

The node numbering is very important that allows us to guarantee a positive volume (or, more precisely, a positive Jacobian determinant at every point). The following rules can be followed for node numbering:

- Chose one starting corner, which is given number 1, and the other 3 corners as 2,3,4 traversing the initial face counterclockwise.
- Number the corners of the opposite face directly opposite 1,2,3,4 as 5,6,7,8, respectively.
The definition of \( \xi \), \( \eta \) and \( \mu \) can be now be made more precise:

- \( \xi \) goes from -1 from (center of) face 1485 to +1 on face 2376
- \( \eta \) goes from -1 from (center of) on face 1265 to +1 on face 3487
- \( \mu \) goes from -1 from (center of) on face 1234 to +1 on face 5678

**Figure 3:** Mapping of a hexahedral element to intrinsic space

II. Governing Equations

The iso parametric formulation of an 8-noded hexahedron element is presented in this section.

**Shape Functions**

- \( N_1^e = \frac{1}{8}(1 - \xi)(1 - \eta)(1 - \mu) \)
- \( N_2^e = \frac{1}{8}(1 + \xi)(1 - \eta)(1 - \mu) \)
- \( N_3^e = \frac{1}{8}(1 + \xi)(1 + \eta)(1 - \mu) \)
- \( N_4^e = \frac{1}{8}(1 - \xi)(1 + \eta)(1 - \mu) \)
- \( N_5^e = \frac{1}{8}(1 - \xi)(1 - \eta)(1 + \mu) \)
- \( N_6^e = \frac{1}{8}(1 + \xi)(1 - \eta)(1 + \mu) \)
- \( N_7^e = \frac{1}{8}(1 + \xi)(1 + \eta)(1 + \mu) \)
- \( N_8^e = \frac{1}{8}(1 - \xi)(1 + \eta)(1 + \mu) \)

Also,\[ x(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)x_i \]
\[ y(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)y_i \]
\[ z(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)z_i \]
\[ u(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)w_i \]

\[ v(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)v_i \]

\[ w(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu)w_i \]

**Jacobian**

The derivatives of the shape functions are given by the following chain rule formulae:

\[
\frac{\partial N_i^e}{\partial x} = \frac{\partial N_i^e}{\partial \xi} \frac{\partial \xi}{\partial x} + \frac{\partial N_i^e}{\partial \eta} \frac{\partial \eta}{\partial x} + \frac{\partial N_i^e}{\partial \mu} \frac{\partial \mu}{\partial x} 
\]

\[
\frac{\partial N_i^e}{\partial y} = \frac{\partial N_i^e}{\partial \xi} \frac{\partial \xi}{\partial y} + \frac{\partial N_i^e}{\partial \eta} \frac{\partial \eta}{\partial y} + \frac{\partial N_i^e}{\partial \mu} \frac{\partial \mu}{\partial y} 
\]

\[
\frac{\partial N_i^e}{\partial z} = \frac{\partial N_i^e}{\partial \xi} \frac{\partial \xi}{\partial z} + \frac{\partial N_i^e}{\partial \eta} \frac{\partial \eta}{\partial z} + \frac{\partial N_i^e}{\partial \mu} \frac{\partial \mu}{\partial z} 
\]

Or,

\[
\begin{pmatrix}
\frac{\partial N_i^e}{\partial x} \\
\frac{\partial N_i^e}{\partial y} \\
\frac{\partial N_i^e}{\partial z}
\end{pmatrix} =
\begin{pmatrix}
\frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \mu}{\partial x} \\
\frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \mu}{\partial y} \\
\frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \mu}{\partial z}
\end{pmatrix}
\begin{pmatrix}
\frac{\partial N_i^e}{\partial \xi} \\
\frac{\partial N_i^e}{\partial \eta} \\
\frac{\partial N_i^e}{\partial \mu}
\end{pmatrix}
\]

\[ J^{-1} = \frac{\partial (\xi, \eta, \mu)}{\partial (x, y, z)} \]
\[ J = \frac{\partial(x, y, z)}{\partial(\xi, \eta, \mu)} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{pmatrix} \]

Since, the isoparametric definition of the hexahedron element is:

\[ x = x_i N_i^e, \quad y = y_i N_i^e, \quad z = z_i N_i^e \]

\[ J = \begin{pmatrix} x_i \frac{\partial N_i^e}{\partial \xi} & y_i \frac{\partial N_i^e}{\partial \xi} & z_i \frac{\partial N_i^e}{\partial \xi} \\ x_i \frac{\partial N_i^e}{\partial \eta} & y_i \frac{\partial N_i^e}{\partial \eta} & z_i \frac{\partial N_i^e}{\partial \eta} \\ x_i \frac{\partial N_i^e}{\partial \mu} & y_i \frac{\partial N_i^e}{\partial \mu} & z_i \frac{\partial N_i^e}{\partial \mu} \end{pmatrix} \]

**Strain Displacement Matrix**

The matrix \( B \) is given by:

\[ B = D\Phi = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \end{pmatrix} \left( \begin{pmatrix} q \\ q \\ q \end{pmatrix} \right) = \begin{pmatrix} q_x & 0 & 0 \\ 0 & q_y & 0 \\ 0 & 0 & q_z \end{pmatrix} \]

where,

\[ q = [ N_1^e \ldots N_n^e ] \]
Constitutive Matrix:

Constitutive Matrix, $C$ is given by:

$$C = \begin{pmatrix}
\lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\
\lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\
\lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\
0 & 0 & 0 & \mu & 0 & 0 \\
0 & 0 & 0 & 0 & \mu & 0 \\
0 & 0 & 0 & 0 & 0 & \mu 
\end{pmatrix}$$

where,

$$\lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)}$$

$$\mu = \frac{E}{2(1 + \nu)}$$

Stiffness Matrix

The elemental stiffness matrix is given by:

$$K^e = \int_{V^e} B^T C B dV^e$$

As in the two-dimensional case, this is replaced by a numerical integration formula which now involves a triple loop over conventional Gauss quadrature rules. Assuming that $C$ matrix is constant,

$$K^e = \sum_{i=1}^{p_1} \sum_{j=1}^{p_2} \sum_{k=1}^{p_3} w_i w_j w_k B^T_{ijk} C B_{ijk} J_{ijk}$$

where, $B_{ijk}$ and $J_{ijk}$ are abbreviations for,

$$B_{ijk} = B(\xi_i, \eta_j, \mu_k) \text{ and } J_{ijk} = \det J(\xi_i, \eta_j, \mu_k)$$

And, $p_1, p_2$ and $p_3$ denote number of Gauss point in $\xi, \eta$ and $\mu$ directions respectively, which is generally the same in all directions i.e. $p = p_1, p_2$ and $p_3 = 2$ in case of 8-noded brick element.

Energy Norm:

The energy norm was computed using:

$$\frac{|U_{FE} - U_{EX}|}{|U_{EX}|}$$

8
Where, $U_{EX}$ is the exact potential energy and $U_{FE}$ is the computed potential energy.

**Boundary Conditions:**

All degrees of freedom on the face opposite to the loaded edge was set to zero in both the problems.

**Data Required:**

- Node info: Node ID, X-Coordinate, Y-Coordinate, Z-Coordinate
- Element info: Element ID, Material ID, Element connectivity matrix
- Material info: Material ID, Elastic Modulus, Poisson’s Ratio
- Boundary Conditions:
  - Dirichlet BC: Node ID, DOF, Value
  - Neumann BC: Element ID, Nodes, DOF, Value

**III. Results**

**I. Rectangular Beam**

**Unloaded Beam:**

![Unloaded body](image)

**Figure 4:** Unloaded Beam

**Deformation:**

9
Result from ABAQUS:

Comparison:

<table>
<thead>
<tr>
<th></th>
<th>Maximum Displacement</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE Code</td>
<td>6.0304</td>
<td>Z</td>
</tr>
<tr>
<td>ABAQUS</td>
<td>5.97456</td>
<td>Z</td>
</tr>
</tbody>
</table>
II. I-Beam

Unloaded Beam:

Deformation:

Figure 7: Unloaded Beam

Figure 8: Deformed beam under load

Figure 9: Top view of the deformed beam under load

Result from ABAQUS:
As we can see from the above tables and figures, the results from the FE code are corroborated by the results obtained from ABAQUS. Interesting observations can be made from the results. As seen in fig. 9, the loaded flange of the I-beam tends to bend inside from the corners while the center does not deform as much due to the support provided by the web. The unloaded flange does not bend. The beam however, as a whole does bend. This observation can also be made from the results obtained from ABAQUS, thus implying that the FE code can capture the physical nature of the structures accurately.

III. Error in energy norm:

The error in energy norm was computed by using the potential energy of the finest mesh possible as $U_{EX}$ and then comparing the subsequent coarser meshes with it. Figure 10 shows the plot of the error in energy norm.
IV. SUMMARY:

Finite element codes were developed in MATLAB using 8-noded brick elements. The computed results were then compared with the results obtained using commercial code ABAQUS. A convergence study was performed based on the energy norm. We can see that the this error increases as ‘h’ increases or as the number of elements decreases. The results are in good agreement with the results obtained from ABAQUS. The plots generated by the code show that the code can accurately capture the physical nature of the structures and give accurate estimate of the deformations caused due to loading.

REFERENCES


V. Appendix

I. FE Code:

```matlab
% Elastic C3D8 Brick Elements
clc
clear
close all

% Read nodes and coords
nod1 = csvread('Nodes1.csv');
nod2 = csvread('Nodes_5.csv');
nod3 = csvread('Nodes_4.csv');
nod4 = csvread('Nodes_3.csv');
nod5 = csvread('Nodes_2.csv');

Elm1 = csvread('Elements1.csv');
Elm2 = csvread('Elements_5.csv');
Elm3 = csvread('Elements_4.csv');
Elm4 = csvread('Elements_3.csv');
Elm5 = csvread('Elements_2.csv');

for no=1:1
    % Read nodes and coords
    if no==1
        Nodes = nod1;
    end
    if no==2
        Nodes = nod2;
    end
    if no==3
        Nodes = nod3;
    end
    if no==4
        Nodes = nod4;
    end
    if no==5
        Nodes = nod5;
    end
    [N,1] = size(Nodes);
end
```

% Read element material id, thickness and nodal connectivity
if no==1
Elems = Elm1;

if no==2
    Elems = Elm2;
end

if no==3
    Elems = Elm3;
end

if no==4
    Elems = Elm4;
end

if no==5
    Elems = Elm5;
end

[E, l] = size(Elems);
j.dbc=1;
nbc=1;

% Number of nodes per element
NE = 8;

% Read material info
Mats = load('Materials.txt');
[M, l] = size(Mats);

% Identify out-of-plane conditions
ipstrn = 1 Plane strain
ipstrn = 2 Plane stress
ipstrn = 2;
nstrn = 3;

% Determine Derichlet BC
for (i=1:N)
    if (Nodes(i,4)==0)
        DBC(j.dbc,1)=Nodes(i,1);
        DBC(j.dbc,2)=1;
        DBC(j.dbc,3)=0;
        j.dbc=j.dbc+1;
    end
end

[P, l] = size(DBC);

% Determine Neummann BC
for (i=1:N)
    if (Nodes(i,4)==60 & Nodes(i,3)==20)
        right(j.nbc,1)=Nodes(i,1);
        j.nbc=j.nbc+1;
    end
end

j.nbc=1;
for i=1:E
for j=1:size(right(:,1))
    for k=3:10
        if Elems(i,k)==right(j,1)
            el_list(j,nbc,1)=Elems(i,1);
            el_list(j,nbc,2)=right(j,1);
            j_nbc=j_nbc+1;
            break
        end
    end
end
NBC(:,1)=unique(el_list(:,1));
j_nbc=1;
for i=1:2:size(el_list(:,1))
    for j=3:10
        if (el_list(i,2)==Elems(el_list(i,1),j))
            NBC(j_nbc,2)=el_list(i,2);
            NBC(j_nbc,3)=el_list(i+1,2);
            j_nbc=j_nbc+1;
        end
    end
end
NBC(:,4)=2;
NBC(:,5)=1;
[Q,1] = size(NBC);

% Determine total number of degrees-of-freedom
udof = 3;  % Degrees-of-freedom per node
NDOF = N*udof;

% Initialize global matrix and vectors
K = zeros(NDOF,NDOF);  % Stiffness matrix
U = zeros(NDOF,1);  % Displacement vector
F = zeros(NDOF,1);  % Force vector

% Set penalty for displacement constraints
Klarge = 10^8;

% Set Gauss point locations and weights
NG = 8;
[XG,WG] = C3D8_E1_Gauss_Points(NG);

% Loop over C3D8 elements
for e = 1:E
    % Establish element connectivity and coordinates
Nnums = Elems(e,3:2+NE);
xyz = Nodes(Nnums(:,2:4));

% Extract element thickness for plane stress
h = Elems(e,3);

% Extract element elastic Young's modulus and Poisson's ratio
Y = Mats(Elems(e,2),2);
nu = Mats(Elems(e,2),3);

% Construct element stiffness matrix
[Ke] = C3D8_E1_Stiff(ipstrn,xyz,Y,nu,udof,NE,NG,XG,WG);

% Assemble element stiffness matrix into global stiffness matrix
ig = udof*(Nnums(:,1)-1);
for ni = 1:NE
  i0 = udof*(ni-1);
  for nj = 1:NE
    j0 = udof*(nj-1);
    for i = 1:udof
      for j = 1:udof
        K(ig(ni)+i,ig(nj)+j) = K(ig(ni)+i,ig(nj)+j) + Ke(i0+i,j0+j);
      end
    end
  end
end

% Construct global force vector for loaded edges with constant traction
NES = 2;
% Set Gauss point locations and weights for traction integration
NGS = 2;
[XGS,WGS] = C3D8_E1_Gauss_Points_Surf(NGS);

for q = 1:Q
  in = zeros(NES);
  tval = zeros(NES,1);
  fval = zeros(NES,1);

  % Determine loaded nodes
e = NBC(q,1);
in1 = NBC(q,2);
in2 = NBC(q,3);
ido = NBC(q,4);
tval(:,1) = NBC(q,5);
for i = 1:NGS
    % Evaluate force contributions at Gauss points
    xi = XGS(i);
    wgt = WGS(i);

    [NshapeS] = C3D8_E1_Shape_Surf(NES, xi);
    [DNshapeS] = C3D8_E1_DShape_Surf(NES, xi);

    xyS(1,1) = Nodes(in1,2);
    xyS(1,2) = Nodes(in1,3);
    xyS(1,3) = Nodes(in1,4);
    xyS(2,1) = Nodes(in2,2);
    xyS(2,2) = Nodes(in2,3);
    xyS(2,3) = Nodes(in2,4);

    [detJS] = C3D8_E1_Jacobian_Surf(NES, xi, xyS, DNshapeS);

    fval = fval + wgt*NshapeS'*NshapeS*tval*detJS;

end

% fval

iloc1 = udof*(in1-1)+idof;
iloc2 = udof*(in2-1)+idof;
F(iloc1) = F(iloc1) + fval(1);
F(iloc2) = F(iloc2) + fval(2);
%

end

% Impose Dirichlet boundary conditions
for p = 1:P
    inode = DBC(p,1);
    idof1 = 1;
    idof2 = 2;
    idof3 = 3;
    idiag1 = udof*(inode-1) + idof1;
    idiag2 = udof*(inode-1) + idof2;
    idiag3 = udof*(inode-1) + idof3;
    K(idiag1,idiag1) = Klarge;
    K(idiag2,idiag2) = Klarge;
    K(idiag3,idiag3) = Klarge;
    F(idiag1) = Klarge*DBC(p,3);
    F(idiag2) = Klarge*DBC(p,3);
    F(idiag3) = Klarge*DBC(p,3);
end

F = F/sum(F);
%
%K
% Solve system to determine displacements
U = inv(K)*F;

% Recover internal element displacements, strains and stresses
nedof = udof*NE;
Disp = zeros(E,nedof);
Eps = zeros(E,nstrn,NG);
Sig = zeros(E,nstrn,NG);

for e = 1:E

% Establish element connectivity and coordinates
Nnums = Elems(e,3:2+NE);
xyz = Nodes(Nnums(:,2:4));

% Extract element thickness for plane stress
h = Elems(e,3);

% Extract element elastic Young’s modulus and Poisson’s ratio
Y = Mats(Elems(e,2),2);
nu = Mats(Elems(e,2),3);

% Extract element nodal displacements
for i = 1:NE
    inode = Nnums(i);
    iglb1 = udof*(inode-1)+1;
    iglb2 = udof*inode;
    iloc1 = udof*(i-1)+1;
    iloc2 = udof*i;
    Disp(e,iloc1) = U(iglb1);
    Disp(e,iloc2) = U(iglb2);
end

% Store element strains
Eps(e,:) = eps(:,:);

% Store element stresses
Sig(e,:) = sig(:,:);
end
PE(no,1) = 0.5*U'*K*U;

PE(no,2) = 3/N;

clearvars -except nod1 nod2 nod3 nod4 nod5 Elm1 Elm2 Elm3 Elm4 Elm5 PE;

do  

for i = 1:5
  if (PE(i,2) == min(PE(:,2)))
    PE_ex = PE(i,1);
  end
end

PE(:,1) = abs(PE(:,1) - PE_ex)/abs(PE_ex);

% Plotting the deformed vs the original shape

Plot_mesh(Nodes(:,2:4), Elems(:,3:10));
title('Unloaded body');
xlabel('X');
ylabel('Y');
zlabel('Z');
j = 1;
for i = 1:3: size(U)
  n_disp(j,1) = U(i);
  n_disp(j,2) = U(i+1);
  n_disp(j,3) = U(i+2);
  j = j+1;
end
n_final(:,1) = Nodes(:,2) + n_disp(:,1);
n_final(:,2) = Nodes(:,3) + n_disp(:,2);
n_final(:,3) = Nodes(:,4) + n_disp(:,3);
figure;
Plot_mesh(n_final, Elems(:,3:10));
title('Body under load');
xlabel('X');
ylabel('Y');
zlabel('Z');

function [DNshape] = C3D8_E1_DShape(NE, xi, eta, mu)

DNshape(1,1) = -(eta - 1)*(mu - 1))/8;
DNshape(2,1) = ((eta - 1)*(mu - 1))/8;
DNshape(3,1) = -((eta + 1)*(mu - 1))/8;
DNshape(4,1) = ((eta + 1)*(mu - 1))/8;
DNshape(5,1) = ((eta - 1)*(mu + 1))/8;
DNshape(6,1) = -((eta - 1)*(mu + 1))/8;
DNshape(7,1) = ((eta + 1)*(mu + 1))/8;
DNshape(8,1) = -((eta + 1)*(mu + 1))/8;

DNshape(1,2) = -(xi/8 - 1/8)*(mu - 1);
DNshape(2,2) = (xi/8 + 1/8)*(mu - 1);
DNshape(3,2) = -(xi/8 + 1/8)*(mu - 1);
DNshape(4,2) = (xi/8 - 1/8)*(mu - 1);
DNshape(5,2) = (xi/8 - 1/8)*(mu + 1);
DNshape(6,2) = -(xi/8 + 1/8)*(mu + 1);
DNshape(7,2) = (xi/8 + 1/8)*(mu + 1);
DNshape(8,2) = -(xi/8 - 1/8)*(mu + 1);

function [DNshapeS] = Q8_E1_DShape_Surf(NES, xi)
DNshapeS(1) = -1/2;
DNshapeS(2) = +1/2;

function [XG,WG] = C3D8_E1_Gauss_Points(NG)
alfl = sqrt(1/3);
XG(1,1) = -alfl;
XG(2,1) = +alfl;
XG(3,1) = +alfl;
XG(4,1) = -alfl;
XG(5,1) = -alfl;
XG(6,1) = +alfl;
XG(7,1) = +alfl;
XG(8,1) = -alfl;
XG(1,2) = -alfl;
XG(2,2) = -alfl;
XG(3,2) = +alfl;
XG(4,2) = +alfl;
function \[ \text{XGS, WGS] = C3D8_E1_Gauss_Points_Surf(NGS) \]

if (NGS == 2)
    alf = sqrt(1/3);
    XGS(1,1) = -alf;
    XGS(2,1) = +alf;
    WGS(1) = 1;
    WGS(2) = 1;
elseif (NGS == 3)
    alf = sqrt(3/5);
    XGS(1,1) = -alf;
    XGS(2,1) = 0;
    XGS(3,1) = +alf;
    WGS(1) = 5/9;
    WGS(2) = 8/9;
    WGS(3) = 5/9;
elseif (NGS == 4)
    alf = 0.8611363115940526;
    bet = 0.3399810435848563;
    XGS(1,1) = -alf;
    XGS(2,1) = -bet;
    XGS(3,1) = bet;
XGS(4,1) = alf;
WGS(1) = 0.3478548451374538;
WGS(2) = 0.6521451548625461;
WGS(3) = 0.6521451548625461;
WGS(4) = 0.3478548451374538;
end

function [Jac, detJ, Jhat] = C3D8_E1_Jacobian(NE, xi, eta, mu, xyz, DNshape)
Jac = zeros(3);
for i = 1:NE
Jac(1,1) = Jac(1,1) + DNshape(i,1)*xyz(i,1);
Jac(1,2) = Jac(1,2) + DNshape(i,1)*xyz(i,2);
Jac(1,3) = Jac(1,3) + DNshape(i,1)*xyz(i,3);
Jac(2,1) = Jac(2,1) + DNshape(i,2)*xyz(i,1);
Jac(2,2) = Jac(2,2) + DNshape(i,2)*xyz(i,2);
Jac(2,3) = Jac(2,3) + DNshape(i,2)*xyz(i,3);
Jac(3,1) = Jac(3,1) + DNshape(i,3)*xyz(i,1);
Jac(3,2) = Jac(3,2) + DNshape(i,3)*xyz(i,2);
Jac(3,3) = Jac(3,3) + DNshape(i,3)*xyz(i,3);
end
detJ = det(Jac);
Jhat = inv(Jac);

function [detJS] = C3D8_E1_Jacobian_Surf(NES, xi, xyS, DNshapeS)
dx = 0;
dy = 0;
dz = 0;
for i = 1:NES
dx = dx + DNshapeS(i)*xyS(i,1);
dy = dy + DNshapeS(i)*xyS(i,2);
dz = dz + DNshapeS(i)*xyS(i,3);
end
detJS = sqrt(dx*dx + dy*dy + dz*dz);

function [Nshape] = C3D8_E1_Shape(NE, xi, eta, mu)
Nshape(1) = (1/8)*(1-xi)*(1-eta)*(1-mu);
Nshape(2) = (1/8)*(1+xi)*(1-eta)*(1-mu);
Nshape(3) = (1/8)*(1+xi)*(1+eta)*(1-mu);
Nshape(4) = (1/8)*(1-xi)*(1+eta)*(1-mu);
Nshape(5) = (1/8)*(1-xi)*(1-eta)*(1+mu);
Nshape(6) = (1/8)*(1+xi)*(1-eta)*(1+mu);
Nshape(7) = (1/8)*(1+xi)*(1+eta)*(1+mu);
Nshape(8) = \((1/8)\times(1-x_i)\times(1+\eta)\times(1+\mu)\);

**function** [NshapeS] = C3D8_E1_Shape_Surf(NES, xi)

NshapeS(1) = \((1-x_i)/2\);
NshapeS(2) = \((1+x_i)/2\);
\%
NshapeS = NshapeS';

**function** [Ke] = C3D8_E1_Stiff(ipstrn, xyz, Y, nu, udf, NE, NG, WG)

ndof = NE*udof;
nstrn = 6;
Ke = zeros(ndof, ndof);

for i=1:NG
    xi = XG(i,1);
    eta = XG(i,2);
    mu = XG(i,3);
    wgt = WG(i);
    \%
    Nshape = C3D8_E1_Shape(NE, xi, eta, mu);
    DNshape = C3D8_E1_DShape(NE, xi, eta, mu);
    [Jac, detJ, Jhat] = C3D8_E1_Jacobian(NE, xi, eta, mu, xyz, DNshape);
    B = zeros(nstrn, ndof);
    i =1;
    for j=1:NE
        qx=DNshape(j,1)*Jhat(1,1)+DNshape(j,2)*Jhat(1,2)+DNshape(j,3)*Jhat(1,3);
        qy=DNshape(j,1)*Jhat(2,1)+DNshape(j,2)*Jhat(2,2)+DNshape(j,3)*Jhat(2,3);
        qz=DNshape(j,1)*Jhat(3,1)+DNshape(j,2)*Jhat(3,2)+DNshape(j,3)*Jhat(3,3);
        B(1,i)=qx;
        B(1,i+1)=0;
        B(1,i+2)=0;
        B(2,i)=0;
        B(2,i+1)=qy;
        B(2,i+2)=0;
        B(3,i)=0;
        B(3,i+1)=0;
        B(3,i+2)=qz;
        B(4,i)=qy;
        B(4,i+1)=qx;
        B(4,i+2)=0;
        B(5,i)=0;
        B(5,i+1)=qz;
        B(5,i+2)=qy;
end
\[
B(6, i) = qz; \\
B(6, i+1) = 0; \\
B(6, i+2) = qx; \\
i = i + 3; \\
\]

lambda = nu*Y/((1+nu)*(1-2*nu));
\[
c = Y/(2*(1+nu)); \\
C = \begin{bmatrix}
\lambda + 2*nu & \lambda & 0 & 0 & 0 \\
\lambda & \lambda + 2*nu & 0 & 0 & 0 \\
\lambda & \lambda & \lambda + 2*nu & 0 & 0 \\
0 & 0 & 0 & \nu & 0 \\
0 & 0 & 0 & 0 & \nu 
\end{bmatrix}; \\
Ke = Ke + wgt*\text{transpose}(B)*C*B*detJ; \\
\]

1 function [eps, sig] = C3D8_El_Str(ipstrn, xyz, u, h, Y, nu, udof, NE, NG, XG) 
2 nstrn = 3; 
3 eps = zeros(nstrn, NG); 
4 sig = zeros(nstrn, NG); 
5 for i = 1:NG 
6 \[
\begin{align*}
x_i &= XG(i, 1); \\
\eta &= XG(i, 2); \\
\mu &= XG(i, 3); \\
\end{align*}
\]
\[
[D\text{Nshape}] = C3D8\_EL\_DShape(NE, x_i, \eta, \mu); \\
[Jac, detJ, Jhat] = C3D8\_EL\_Jacobian(NE, x_i, \eta, \mu, xyz, D\text{Nshape}); \\
B = zeros(nstrn, ndof); 
6 for j = 1:NE 
7 \[
\begin{align*}
jloc1 &= 2*(j-1)+1; \\
\text{jloc2} &= \text{jloc1} + 1; \\
B(1, \text{jloc1}) &= B(1, jloc1) + Jhat(1, 1)*D\text{Nshape}(j, 1) ... \\
&+ Jhat(1, 2)*D\text{Nshape}(j, 2); \\
B(2, jloc2) &= B(2, jloc2) + Jhat(2, 1)*D\text{Nshape}(j, 1) ... \\
&+ Jhat(2, 2)*D\text{Nshape}(j, 2); \\
B(3, jloc1) &= B(3, jloc1) + Jhat(2, 1)*D\text{Nshape}(j, 1) ... \\
&+ Jhat(2, 2)*D\text{Nshape}(j, 2); \\
B(3, jloc2) &= B(3, jloc2) + Jhat(1, 1)*D\text{Nshape}(j, 1) ... \\
&+ Jhat(1, 2)*D\text{Nshape}(j, 2); \\
\end{align*}
\]
7
8 \end{for} 
9 \text{if (ipstrn == 1)} 
10 \[
c = Y*(1-nu)/(1-2*nu)/(1+nu); \\
C = c*\begin{bmatrix}
1 & nu/(1-nu) & 0; \\
u/(1-nu) & 1 & 0; \\
0 & 0 & (1-2*nu)/(1-nu)/2 
\end{bmatrix}; \\
\text{else} \\
c = Y/(1-nu)/(1+nu); \\
\text{end} 
11 \]
\[ C = \mathbf{c} \begin{bmatrix} 1 & \nu & 0; & \nu & 1 & 0; & 0 & 0; & (1-\nu)/2 \end{bmatrix}; \]

\[
\text{end}
\]

\[
\text{eps}(;i) = \mathbf{B} \mathbf{u};
\]
\[
\text{sig}(;i) = \mathbf{C} \text{eps}(;i);
\]

\[
\text{end}
\]

II. Code to plot the mesh:

```matlab
function Plot mesh(node_coord, elements)

n_el = length(elements); % number of elements
node_face = [1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 1 2 3 4; 5 6 7 8]; % Nodes on faces
XYZ = cell(1, n_el); %

for e = 1:n_el
    nd = elements(e,:);
    XYZ{e} = [node_coord(nd,1) node_coord(nd,2) node_coord(nd,3)];
end

% Plot
axis equal;
axis tight;
cellfun(@(patch, remat({'Vertices'},1,n_el), XYZ, remat({'Faces'},1,n_el), remat({'node_face'},1,n_el), remat({'FaceColor'},1,n_el), remat({'w'},1,n_el));
view(3)
end
```