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

List of Figures

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 solve 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.

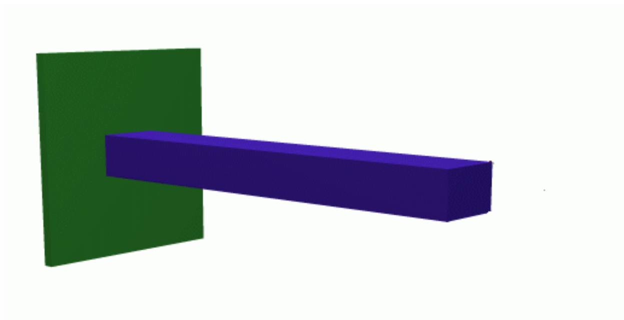

Figure 1: *A Cantilever Beam*

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 ξ, η and $\mu \ \epsilon(-1, 1)$
- As in the case of quadrilaterals, this particular choice of limits was made to facilitate the use of the standard Gauss integration formulas.

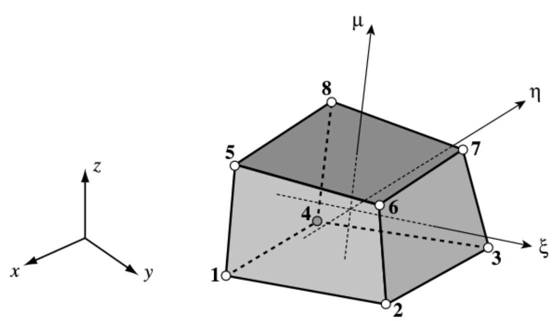

Figure 2: *A Hexahedron Element*

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 ξ , η and μ can be now be made more precise:

- ξ goes from -1 from (center of) face 1485 to +1 on face 2376
- η goes from -1 from (center of) on face 1265 to +1 on face 3487
- μ goes from -1 from (center of) on face 1234 to +1 on face 5678

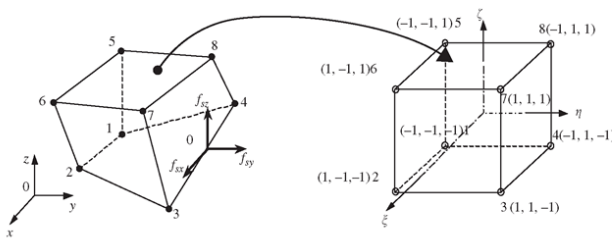

Figure 3: *Mapping of a hexahedral element to intrinsic space*

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,
$x(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu) x_i$

$y(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu) y_i$

$z(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu) z_i$

$u(\xi, \eta, \mu) = \sum_i N_i(\xi, \eta, \mu) u_i$

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

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

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,

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

$$J^{-1}$$

$$\implies J^{-1} = \frac{\partial(\xi, \eta, \mu)}{\partial(x, y, z)}$$

$$\implies J = \frac{\partial(x,y,z)}{\partial(\xi,\eta,\mu)} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{pmatrix}$$

Since, the isoparametric definition of the hexahedron element is:

$$x = x_i N_i^e,\ y = y_i N_i^e,\ z = z_i N_i^e$$

,

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

Strain Displacement Matrix

The matrix $\mathbf{B}$ is given by:

$$\mathbf{B} = \mathbf{D}\mathbf{\Phi} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial x} & 0 \\ 0 & 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \mathbf{q} & 0 & 0 \\ \mathbf{q} & \mathbf{q} & 0 \\ 0 & 0 & \mathbf{q} \end{pmatrix} = \begin{pmatrix} \mathbf{q_x} & 0 & 0 \\ 0 & \mathbf{q_y} & 0 \\ 0 & 0 & \mathbf{q_z} \\ \mathbf{q_y} & \mathbf{q_x} & 0 \\ 0 & \mathbf{q_z} & \mathbf{q_y} \\ \mathbf{q_z} & 0 & \mathbf{q_x} \end{pmatrix}$$

where,

$$\mathbf{q} = [\ N_1^e \quad ... \quad N_n^e\ ]$$

Constitutive Matrix:

Constitutive Matrix, $\mathbf{C}$ is given by:

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

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:

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

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 $\mathbf{C}$ matrix is constant,

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

where, $\mathbf{B}_{ijk}$ and J_{ijk} are abbreviations for,

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

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:

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

Where, U_{EX} is the exact potetial 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 t 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:

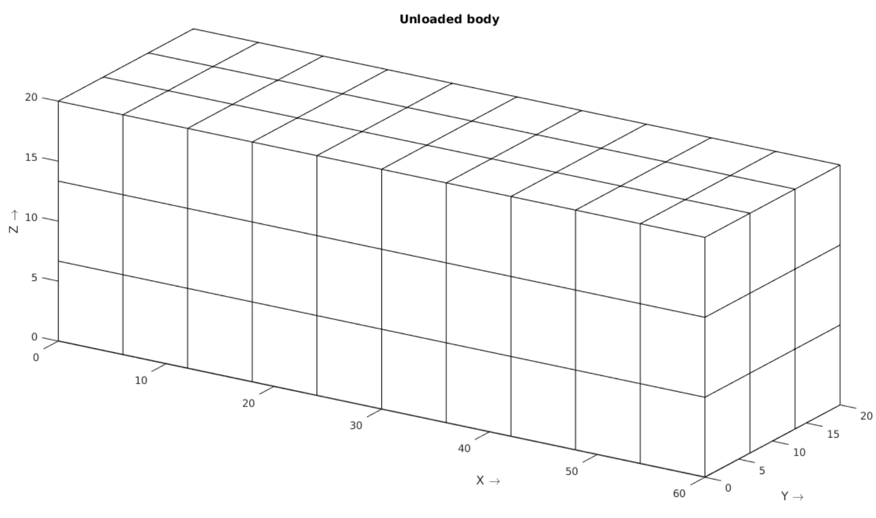

Figure 4: *Unloaded Beam*

Deformation:

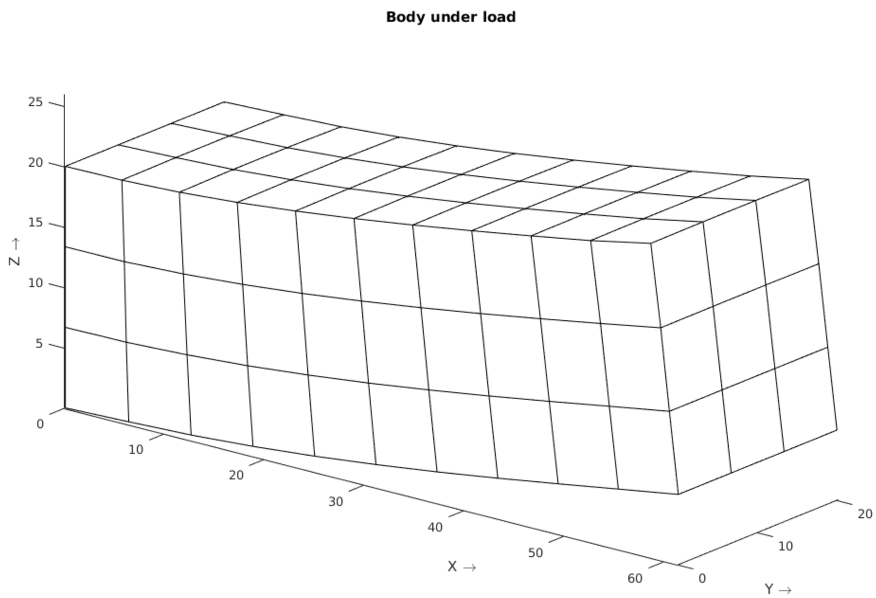

Figure 5: *Deformed beam under load*

Result from ABAQUS:

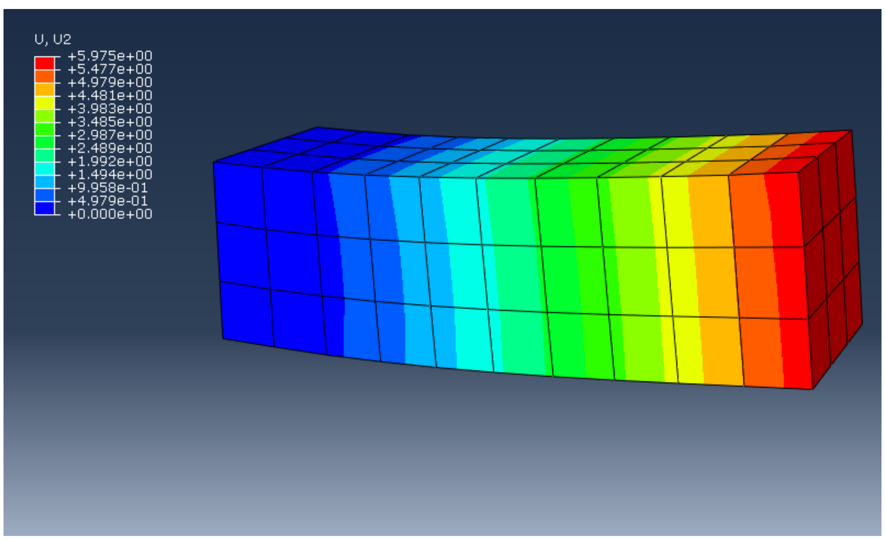

Figure 6: *Displacement Contour plotted by ABAQUS*

Comparison:

	Maximum Displacement	Direction
FE Code	6.0304	Z
ABAQUS	5.97456	Z

II. I-Beam

Unloaded Beam:

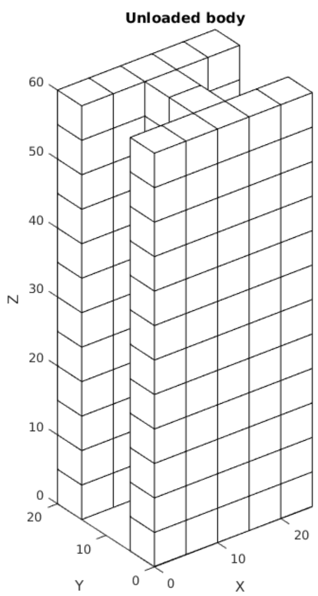

Figure 7: *Unloaded Beam*

Deformation:

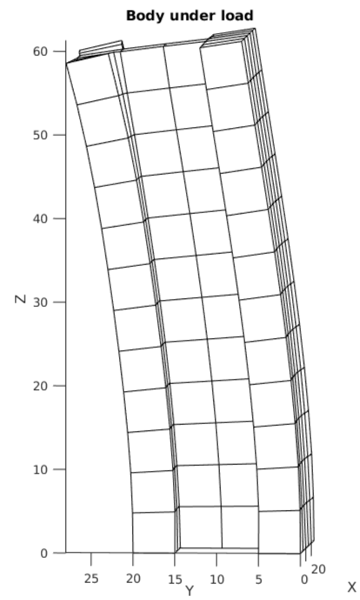

Figure 8: *Deformed beam under load*

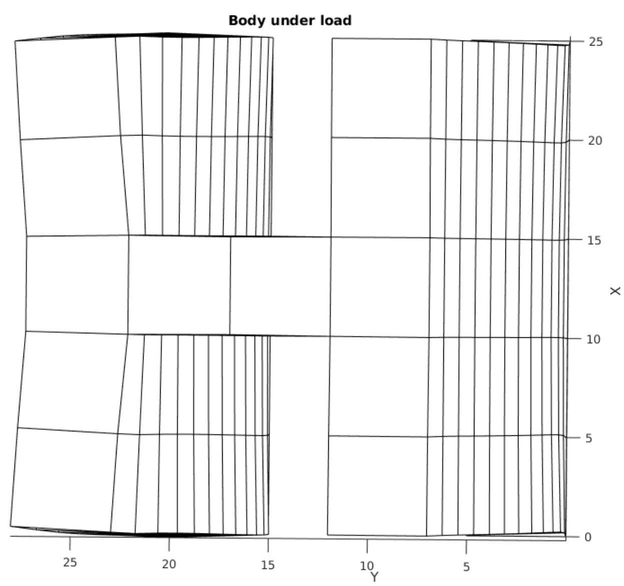

Figure 9: *Top view of the deformed beam under load*

Result from ABAQUS:

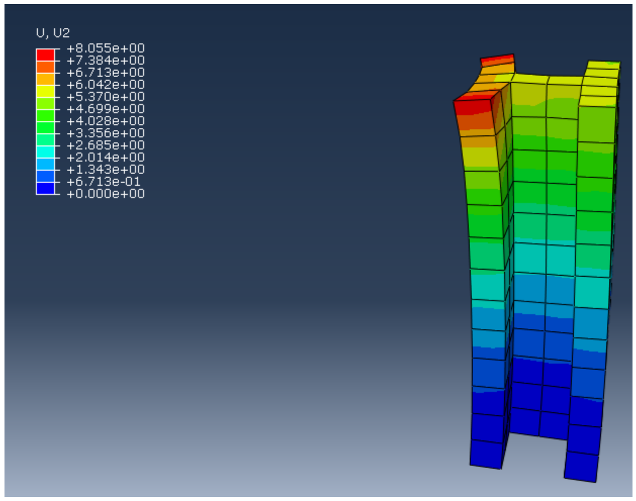

Figure 10: *Displacement Contour plotted by ABAQUS*

COMPARISON:

	Maximum Displacement	Direction
FE Code	8.0164	Y
ABAQUS	8.05540	Y

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 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.

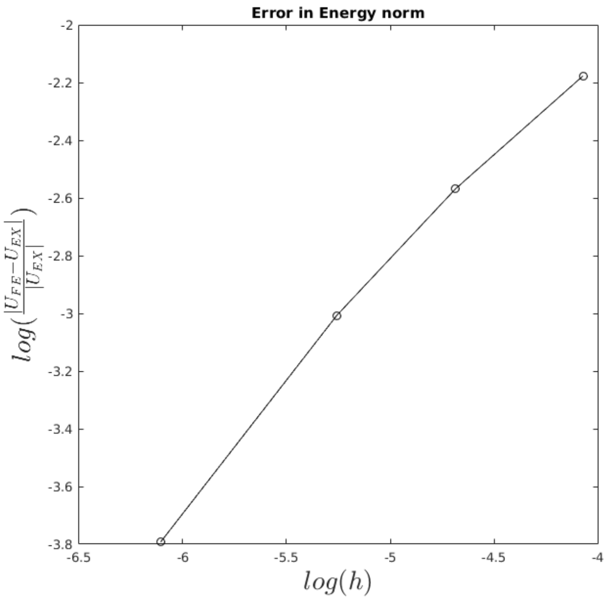

Figure 11: *Error in energy norm*

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

[1] Gary F. Dargush. Lecture notes in finite element analysis(mae529). *Department of Mechanical and Aerospace Engineering, University at Buffalo, The State University of New York*, Fall 2016.

[2] Carlos A. Felippa. Advanced finite element methods (asen 6367), department of aerospace engineering sciences university of colorado at boulder, 2013.

V. Appendix

I. FE Code:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
% 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,l] = size(Nodes);

%     Read element material id, thickness and nodal connectivity
    if no==1
```

```
        Elems = Elm1;
    end
    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;
    j_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 Neumann 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
    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,l] = 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_El_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_El_Stiff(ipstrn,xyz,Y,nu,udof,NE,NG,XG,WG);

        % Assemble element stiffness matrix into global stiffness matrix
        ig = udof*(Nnums(:)-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
    end
    %K

    % Construct global force vector for loaded edges with constant traction
    NES = 2;
    % Set Gauss pint locations and weights for traction integration
    NGS = 2;
    [XGS,WGS] = C3D8_El_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);
        idof = 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_El_Shape_Surf(NES,xi);
            [DNshapeS] = C3D8_El_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_El_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);
        %F

    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
```

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

    % Recover internal element displacement, 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
        %Disp

        u = Disp(e,:)';
        [eps,sig] = C3D8_El_Str(ipstrn,xyz,u,h,Y,nu,udof,NE,NG,XG);
        %eps
        %sig

        % 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;
end
%
% 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);
%
% % figure;
% plot(log(PE(:,2)),log(PE(:,1)), '-o');
% title('Error in Energy norm');
% xlabel('$log(h)$','Interpreter','latex');
% ylabel('$log(\frac{|U_{FE}-U_{EX}|}{|U_{EX}|})$','Interpreter','latex'); axis square;


% 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_El_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);

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

```
function [DNshapeS] = Q8_El_DShape_Surf(NES,xi)

DNshapeS(1) = -1/2;
DNshapeS(2) = +1/2;
```

```
function [XG,WG] = C3D8_El_Gauss_Points(NG)



alf = sqrt(1/3);

XG(1,1) = -alf;
XG(2,1) = +alf;
XG(3,1) = +alf;
XG(4,1) = -alf;
XG(5,1) = -alf;
XG(6,1) = +alf;
XG(7,1) = +alf;
XG(8,1) = -alf;

XG(1,2) = -alf;
XG(2,2) = -alf;
XG(3,2) = +alf;
XG(4,2) = +alf;
```

```
XG(5,2) = -alf;
XG(6,2) = -alf;
XG(7,2) = +alf;
XG(8,2) = +alf;

XG(1,3) = -alf;
XG(2,3) = -alf;
XG(3,3) = -alf;
XG(4,3) = -alf;
XG(5,3) = +alf;
XG(6,3) = +alf;
XG(7,3) = +alf;
XG(8,3) = +alf;

for i=1:NG
    WG(i) = 1;
end

end
```

```
function [XGS,WGS] = C3D8_El_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_El_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_El_Jacobian_Surf(NES,xi,xyS,DNshapeS)

dxdxi = 0;
dydxi = 0;
dzdxi = 0;
for i=1:NES
    dxdxi = dxdxi + DNshapeS(i)*xyS(i,1);
    dydxi = dydxi + DNshapeS(i)*xyS(i,2);
    dzdxi = dzdxi + DNshapeS(i)*xyS(i,3);
end

detJS = sqrt( dxdxi*dxdxi + dydxi*dydxi + dzdxi*dzdxi);
```

```
function [Nshape] = C3D8_El_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)*(1-xi)*(1+eta)*(1+mu);
```

```
function [NshapeS] = C3D8_El_Shape_Surf(NES, xi)

NshapeS(1) = (1-xi)/2;
NshapeS(2) = (1+xi)/2;
%NshapeS = NshapeS';
```

```
function [Ke] = C3D8_El_Stiff(ipstrn,xyz,Y,nu,udof,NE,NG,XG,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_El_Shape(NE,xi,eta,mu);
    [DNshape] = C3D8_El_DShape(NE,xi,eta,mu);
    [Jac,detJ,Jhat] = C3D8_El_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;
```

```
          B(6,i)=qz;
          B(6,i+1)=0;
          B(6,i+2)=qx;
           i=i+3;
      end

      lambda=nu*Y/((1+nu)*(1-2*nu));
      c=Y/(2*(1+nu));
      C = [lambda+2*nu lambda lambda 0 0 0; lambda lambda+2*nu lambda 0 0 0; lambda lambda lambda+2*nu 0 0 0; 0 0 0 nu 0 0; 0 0 0 0 nu 0; 0 0 0 0 0 nu];

      Ke = Ke + wgt*transpose(B)*C*B*detJ;

end
```

```
function [eps,sig] = C3D8_El_Str(ipstrn,xyz,u,h,Y,nu,udof,NE,NG,XG);

ndof = NE*udof;
nstrn = 3;
eps = zeros(nstrn,NG);
sig = zeros(nstrn,NG);

for i=1:NG

    xi  = XG(i,1);
    eta = XG(i,2);
    mu = XG(i,3);

    [DNshape] = C3D8_El_DShape(NE,xi,eta,mu);
    [Jac,detJ,Jhat] = C3D8_El_Jacobian(NE,xi,eta,mu,xyz,DNshape);

    B = zeros(nstrn,ndof);
    for j=1:NE
        jloc1 = 2*(j-1)+1;
        jloc2 = jloc1 + 1;
        B(1,jloc1) = B(1,jloc1) + Jhat(1,1)*DNshape(j,1) ...
            + Jhat(1,2)*DNshape(j,2);
        B(2,jloc2) = B(2,jloc2) + Jhat(2,1)*DNshape(j,1) ...
            + Jhat(2,2)*DNshape(j,2);
        B(3,jloc1) = B(3,jloc1) + Jhat(2,1)*DNshape(j,1) ...
            + Jhat(2,2)*DNshape(j,2);
        B(3,jloc2) = B(3,jloc2) + Jhat(1,1)*DNshape(j,1) ...
            + Jhat(1,2)*DNshape(j,2);
    end

    if (ipstrn == 1)
        c = Y*(1-nu)/(1-2*nu)/(1+nu);
        C = c*[ 1 nu/(1-nu) 0; nu/(1-nu) 1 0; 0 0 (1-2*nu)/(1-nu)/2 ];
    else
        c = Y/(1-nu)/(1+nu);
```

```
        C = c*[ 1 nu 0; nu 1 0; 0 0 (1-nu)/2 ];
    end

    eps(:,i) = B*u;
    sig(:,i) = C*eps(:,i);

end
```

II. Code to plot the mesh:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                           PLOTTING THE MESH                                  %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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,repmat({'Vertices'},1,n_el),XYZ,repmat({'Faces'},1,n_el),repmat
    ({node_face},1,n_el),repmat({'FaceColor'},1,n_el),repmat({'w'},1,n_el));
        view(3)
end
```