Finite Element Analysis of a plate with a hole using Constant Strain triangle, four and eight noded isoparametric quadrilateral elements

Sagar Bhatt

Person Number: 50170651

Department of Mechanical and Aerospace Engineering,

University at Buffalo

Contents

List of Figures

Abstract

This project aims at studying the deformation of a thin plate with a central circular hole when the plate is loaded in tension with a constant load. Finite element codes were developed in MATLAB using constant strain triangle elements, four- and eight-noded isoparametric elements. A convergence study was performed based on the energy norm a for all three cases and the stress concentration factor around the circular whole was also investigated.The computed results were then compared with the results obtained using commercial code ABAQUS.

I. Introduction

A machine component is bound to have irregularities like a hole in its geometry for various design reasons. These irregularities can contribute immensely to the strength of the part. Configuring the structures with discontinuities is one of the most important topics in the construction of ships, aero-planes, cars etc. Examples of problems in which discontinuities play prominent role in the physical behavior of a system are numerous. From mathematical point of view, analytical solutions are possible only for a limited class of such problems. Many times an accurate solution was not possible due to the complexity of the discontinuity configuration. However, with the advent of Finite element method (FEM), theses analyses can now be performed with a degree of accuracy.[3]

A Constant Strain Triangle element, also referred to as a CST element or a T3 element, has constant shape functions which when applied to plane stress or plane strain conditions, yield approximate solutions for stress and strain fields that are constant throughout the domain of the element.

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 two quadrilateral isoparametric elements are being considered, 4-noded (also called Q4 element) and 8-noded (also called Q8 element).

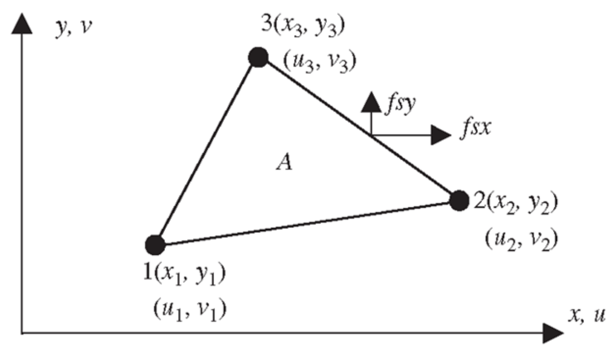

Figure 1: *A constant strain triangle element*

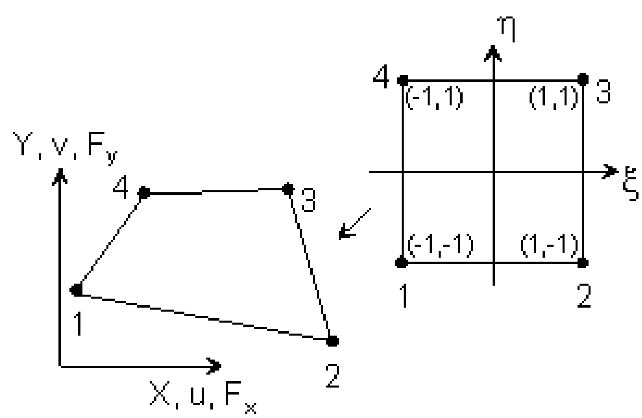

Figure 2: *A 4-noded quadrilateral element*

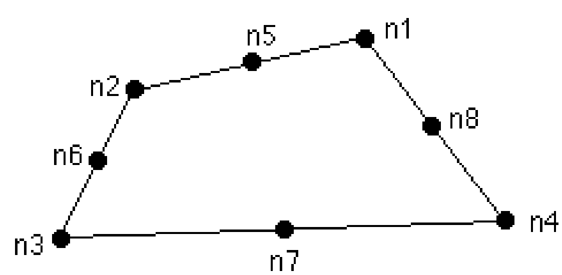

Figure 3: *An 8-noded quadrilateral element*

I. Problem Set Up

The problem we are considering consists of a finite plate of length 2L and width 2H with a central hole of radius R, depicted in fig. 4. The relationship between these dimensions is given by: $L/H = \alpha, R/H = \beta$ and these vales were determined from the my UB person number (50170651). Let *abcd ijkl* be the person number, the the pooisson's ratio, $\nu = j/20$. ans $\alpha = (k+1)/2, \beta = 1/(l+3)$. Young's modulus was arbitrarily assumed to be 2.5. Hence, in this case, $\alpha = L/H = 3, \beta = R/H = 0.25 \; and \; \nu = 0.3$. Since the problem was symmetrical, we broke the problem to a quarter plate and solved the problem for just one quadrant of the plate using these parameters.

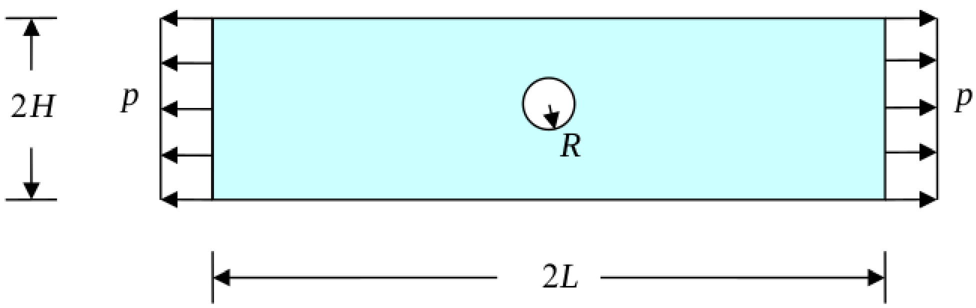

Figure 4: *The problem setup*

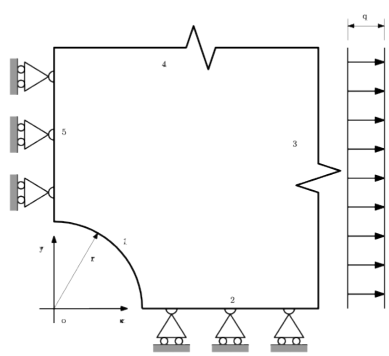

Figure 5: *The quadrant being used in this study considering symmetry*

II. Method of Solution

I. Governing Equations

A general approach for a displacement based finite element formulation is given by the following nine steps[1]. We will demonstrate this formulation for 2D CST. The iso parametric formulation follows the same basic guideline with additional steps involving coordinate transformation and Gauss Quadrature.

1. Choose the coordinate system and define the node numbering system, nodal displacements and body forces for the element.

   $\vec{u}^e = \{u_1\ v_1\ u_2\ v_2\ u_3\ v_3\}^T$

   $\vec{f}^e = \{f_{x1}\ f_{y1}\ f_{x2}\ f_{y2}\ f_{x3}\ f_{y3}\}^T$

2. Choose a displacement function that can represent the fundamental deformation of the elements. According to Principle of Virtual Work (PVW):

   $\int_\Omega \sigma_{ij} d\Omega = \int_{\Gamma_t} t_i \delta u_i d\Gamma + \int_\Omega f_i \delta u_i d\Omega$

   Here, if the highest order derivative is n^{th} order, C^{n-1} continuity is required. In case of elastic bodies, the highest order derivative is 1^{st} order, hence we require C_0 continuity. So, Let:

   $u(x) = \alpha_1 + \alpha_2 x + \alpha_3 y$

   $v(x) = \alpha_4 + \alpha_5 x + \alpha_4 y$

   Then,
   $\vec{u}(\vec{x}) = \Phi(\vec{x})\vec{\alpha}$

   i.e.

$$\begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

   or,

   $\vec{u}^e = \mathbb{A}\vec{\alpha}$

$$\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} u(x_1, y_1) \\ v(x_1, y_1) \\ u(x_1, y_1) \\ v(x_1, y_1) \\ u(x_2, y_2) \\ v(x_3, y_3) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

or,

$$\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \overline{\mathbb{A}} & 0 \\ 0 & \overline{\mathbb{A}} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

where,

$$\overline{\mathbb{A}} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}$$

Then,

$$\mathbb{A}^{-1} = \begin{bmatrix} \overline{\mathbb{A}}^{-1} & 0 \\ 0 & \overline{\mathbb{A}}^{-1} \end{bmatrix}$$

$$\implies \vec{u}(\vec{x}) = \Phi(\vec{x})\mathbb{A}^{-1}\vec{u}^e$$

3.

$$\vec{\epsilon}(\vec{x}) = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

$$\epsilon_{xx} = \frac{\partial u}{\partial x}$$

$$\epsilon_{yy} = \frac{\partial v}{\partial y}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

$$\vec{\epsilon}(\vec{x}) = \partial \Phi \mathbb{A}^{-1} \vec{u}^e$$

$$\implies \vec{\epsilon}(\vec{x}) = \mathbb{B} \vec{u}^e$$

4.

$$\vec{\sigma}(\vec{x}) = \mathbb{C}(\vec{x}) \vec{\epsilon}(\vec{x})$$

Where,

$$\mathbb{C}(\vec{x}) = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{2\nu}{2(1-\nu)} \end{bmatrix}$$

5.

$\delta \vec{u}(\vec{x}) = (N)(\vec{x}) \delta \vec{u}^e$
$\partial \vec{\epsilon}(\vec{x}) = \mathbb{B} \partial \vec{u}^e$

6. Invoke PVW and develop elemental stiffness matrix:
   $\mathbb{K}^e = \int_\Omega \mathbb{B}^T \mathbb{C} \mathbb{B} d\Omega$

   Now,
   $\mathbb{K}^e \vec{u}^e = \vec{f}^e$

7. Assemble and enforce boundary conditions
   $\mathbb{K} \vec{u} = \vec{f}$

8. Solve for $\vec{u}$

9. Post-Processing

For Isoparametric Elements:

Shape Functions:

- 4-noded element:
  - $N_1(\zeta,\eta) = \frac{1}{4}(1-\zeta)(1-\eta)$
  - $N_2(\zeta,\eta) = \frac{1}{4}(1+\zeta)(1-\eta)$
  - $N_3(\zeta,\eta) = \frac{1}{4}(1+\zeta)(1+\eta)$
  - $N_4(\zeta,\eta) = \frac{1}{4}(1-\zeta)(1+\eta)$
- 8-noded element:
  - $N_1(\zeta,\eta) = -\frac{1}{4}(1-\zeta)(1-\eta)(\zeta+\eta+1)$
  - $N_1(\zeta,\eta) = \frac{1}{4}(1+\zeta)(1-\eta)(xi-\eta-1)$
  - $N_1(\zeta,\eta) = \frac{1}{4}(1+\zeta)(1+\eta)(\zeta+\eta-1)$
  - $N_1(\zeta,\eta) = -\frac{1}{4}(1-\zeta)(1+\eta)(\zeta-\eta+1)$
  - $N_1(\zeta,\eta) = \frac{1}{2}(1-\zeta^2)(1-\eta)$
  - $N_1(\zeta,\eta) = \frac{1}{2}(1-\eta^2)(1+\zeta)$
  - $N_1(\zeta,\eta) = \frac{1}{2}(1-\zeta^2)(1+\eta)$
  - $N_1(\zeta,\eta) = \frac{1}{2}(1-\eta^2)(1-\zeta)$

Also,
$x(\zeta,\eta) = \sum_i N_i(\zeta,\eta)x_i$

$y(\zeta,\eta) = \sum_i N_i(\zeta,\eta)y_i$

$u(\zeta,\eta) = \sum_i N_i(\zeta,\eta)u_i$

$v(\zeta,\eta) = \sum_i N_i(\zeta,\eta)v_i$

Jacobian: $\mathbb{J} = \begin{bmatrix} \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}$

Strain:

$$\vec{\epsilon}(\vec{x}) = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix}$$

$$\epsilon_{xx} = \frac{\partial u}{\partial x} = \sum_i (\hat{J}_{11}\frac{\partial N_i}{\partial \zeta} + \hat{J}_{12}\frac{\partial N_i}{\partial \eta})u_i$$

$$\epsilon_{yy} = \frac{\partial v}{\partial y} = \sum_i (\hat{J}_{21}\frac{\partial N_i}{\partial \zeta} + \hat{J}_{22}\frac{\partial N_i}{\partial \eta})v_i$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

Stress Concentration Factor:

The stress concentration factor was computed base on the following formula:
$SCF = \frac{\sigma_{max}}{\sigma_{nom}},\ where,$
$\sigma_{nom} = \frac{load}{minimum\ cross\ section}$

Energy Norm:

The energy norm was computed using: $\frac{|U_{FE} - U_{EX}|}{|U_{EX}|}$

Boundary Conditions:

Since we are considering only one quadrant of the plate, fig.5, the following boundary conditions were imposed on this geometry:

- Left edge : No displacement in x-direction i.e. 1^{st} degree of freedom set to 0.
- Bottom edge : No displacement in y-direction i.e. 2^{nd} degree of freedom set to 0.

III. Results

I. Constant Strain Triangle Element

Number of Elements	Stress Concentration Factor	Maximum Displacement
2190	1.3363	1.2619
594	1.3580	1.2589
394	1.1950	1.2567
282	1.3594	1.2557
156	1.0475	1.2531

Deformation:

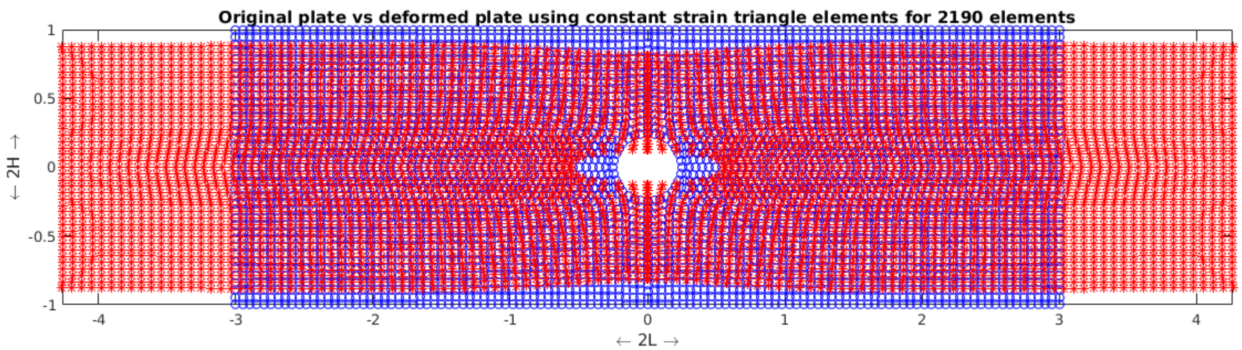

Figure 6: *Deformation of the plate due to the load for CST elements*

Convergence Based on Energy Norm:

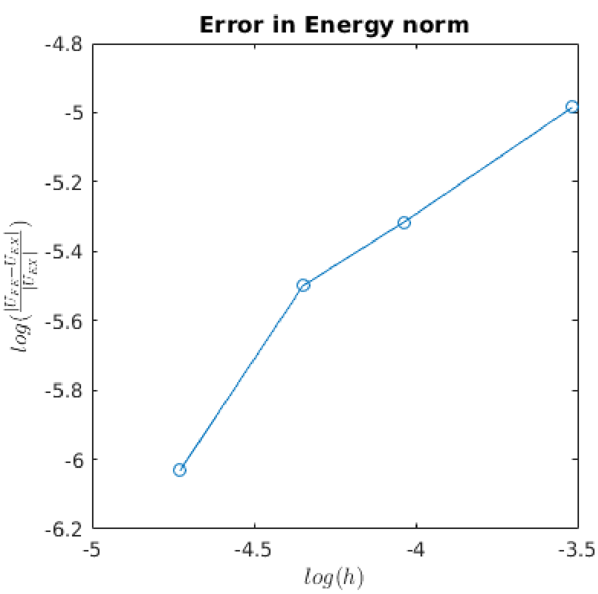

Figure 7: *Error in energy norm for CST elements*

II. Four Noded Quadrilateral Isoparametric Element

Number of Elements	Stress Concentration Factor	Maximum Displacement
1095	1.2679	1.2627
300	1.7638	1.2616
197	1.6378	1.2598
137	1.5582	1.2588
78	1.4677	1.2582

DEFORMATION:

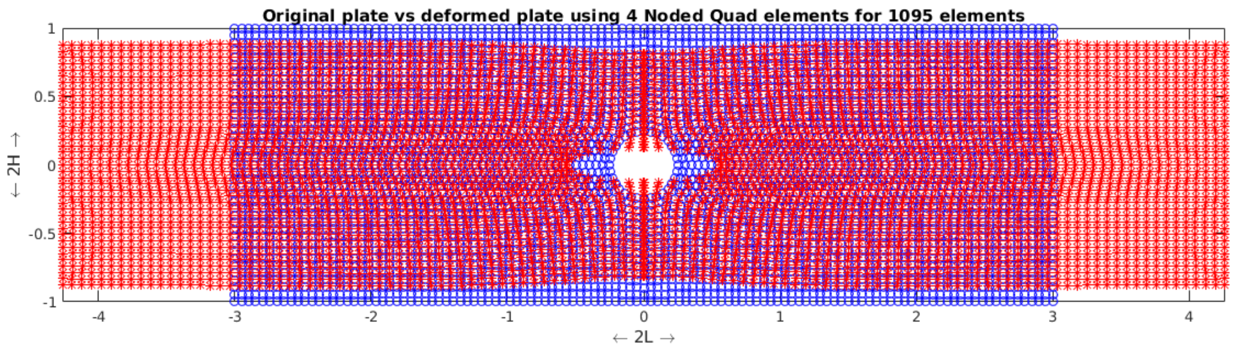

Figure 8: *Deformation of the plate due to the load for Q4 elements*

CONVERGENCE BASED ON ENERGY NORM:

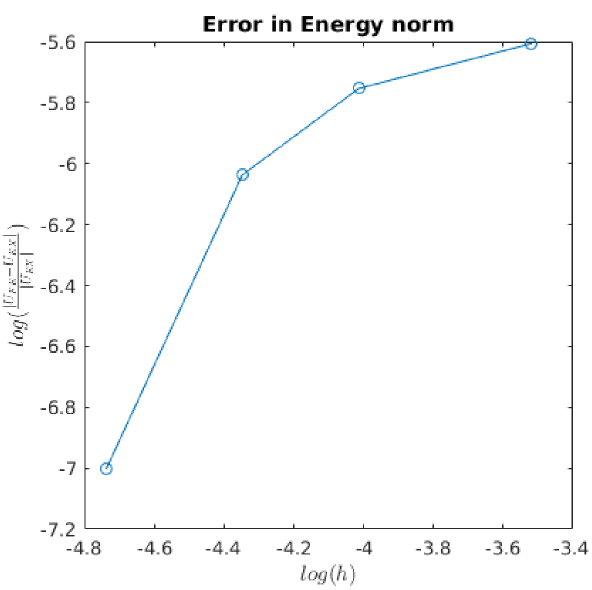

Figure 9: *Error in energy norm for Q4 elements*

III. Eight Noded Quadrilateral Isoparametric Element

Number of Elements	Stress Concentration Factor	Maximum Displacement
300	1.7540	1.3005
297	1.7378	1.3005
197	1.6175	1.3070
137	1.5389	1.3093
78	1.4529	1.3224

DEFORMATION:

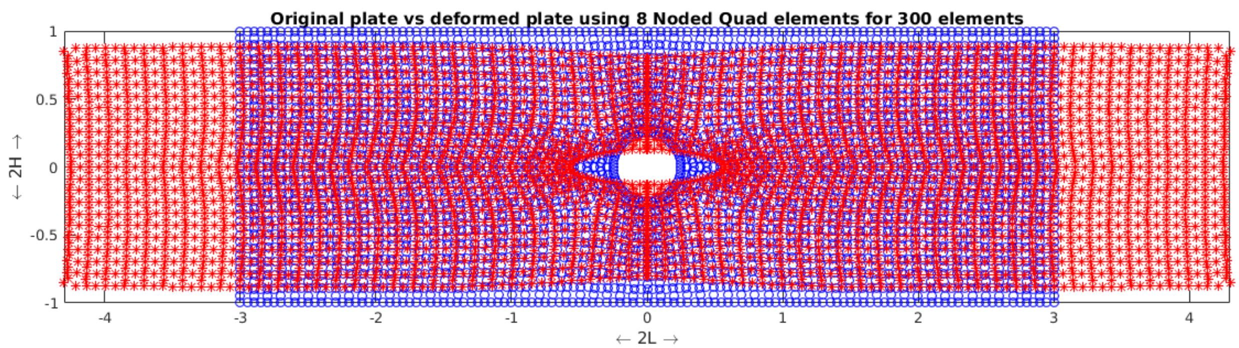

Figure 10: *Deformation of the plate due to the loading for Q8 elements*

CONVERGENCE BASED ON ENERGY NORM:

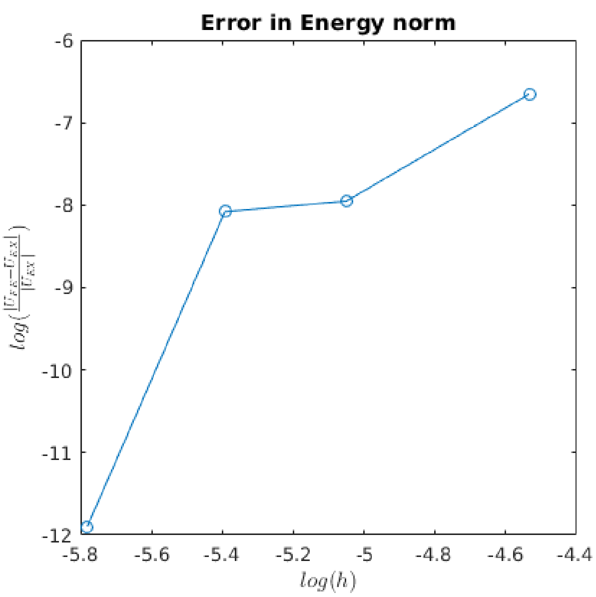

Figure 11: *Error in energy norm for Q8 elements*

IV. Results from ABAQUS

The problem was solved using ABAQUS as well. The maximum displacement obtained through the FE code was compared with the maximum displacement obtained through ABAQUS:

U_{max}, Abaqus: 1.26394
U_{max}, CST: 1.2619
U_{max}, Q4: 1.2627
U_{max}, Q8: 1.3005

As we can see from the above comparison, the closest result to ABAQUS's solution is that of Q4 elements. This is because the mesh chosen for the study on ABAQUS was equiped with Q4 elements. Hence, a comparative study between these two results is more suitable. However, the results obtained through any other elements should not be very different. The proximity of the results through all the methods corroborates the accuracy of the computation. Figure 12 demonstrates the deformation of a quadrant of the plate under the load.

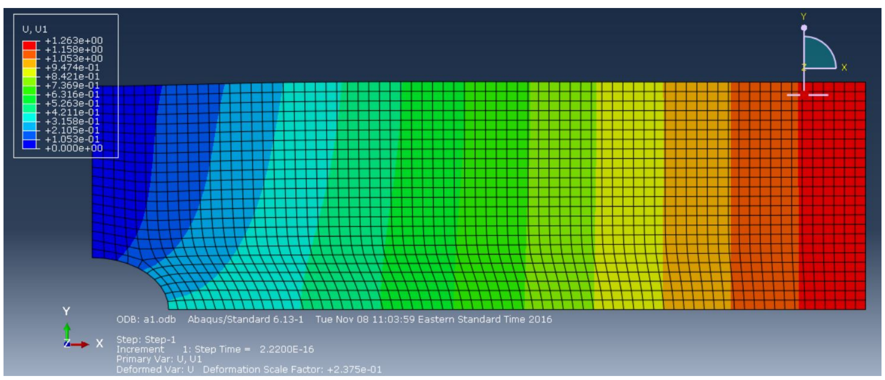

Figure 12: *Displacement plotted by ABAQUS*

IV. Summary:

Finite element codes were developed in MATLAB using constant strain triangle elements, four- and eight-noded isoparametric elements. A convergence study was performed based on the energy norm a for all three cases and the stress concentration factor around the circular whole was also investigated. The computed results were then compared with the results obtained using commercial code ABAQUS. The computed maximum displacement was ≈ 1.263 (for Q4 elements) where as the result from the result from ABAQUS showed this to be $= 1.264$. The results are in very good agreement with each other. The stress concentration factor was also computed. The deformation was plotted using the MATLAB code as well as ABAQUS. The convergence study based on the error in energy norm was also computed and is presented here in graphical form. We can see that the this error increases as 'h' increases or as the number of elements decreases. The results of deformation show that the maximum displacement corresponds to the node at the center of each loaded side. Due to symmetric loading, the central nodes do not move and the circle gets deformed to an ellipse. Another observation to be made here is as the number of nodes per element increases, we can see the computed deformation increases and the minor axis of now deformed hole is much smaller resulting in a much sharper ellipse.

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] E Hart and V Hudramovich. Projection-iterative schemes for the realization of the finite-element method in problems of deformation of plates with holes and inclusions. *Journal of Mathematical Sciences*, 203(1), 2014.

[3] Sharaban Thohura and Md Shahidul Islam. Study of the effect of finite element mesh quality on stress concentration factor of plates with holes. *Proceedings of the 15th Annual Paper Meet*, 7:08, 2014.

V. Appendix

I. Code to Plot Deformation:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
%                      PLOTTING THE DEFORMATION                           %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%  PLotting the initial position nodes

plot(Nodes(:,2),Nodes(:,3),'ob'); axis equal; axis tight; hold on;
plot(-Nodes(:,2),-Nodes(:,3),'ob'); axis equal; axis tight; hold on;
plot(-Nodes(:,2),Nodes(:,3),'ob'); axis equal; axis tight; hold on;
plot(Nodes(:,2),-Nodes(:,3),'ob'); axis equal; axis tight; hold on;

% Finding the final position of the nodes
j=1;
for i=1:2:size(U)
    n_disp(j,1)=U(i);
```

```
    n_disp(j,2)=U(i+1);
    j=j+1;
end
n_final(:,1)=Nodes(:,2)+n_disp(:,1);
n_final(:,2)=Nodes(:,3)+n_disp(:,2);


% Plotting the final positions of the nodes

plot(n_final(:,1),n_final(:,2),'*r'); axis equal; axis tight; hold on;
plot(-n_final(:,1),n_final(:,2),'*r'); axis equal; axis tight; hold on;
plot(n_final(:,1),-n_final(:,2),'*r'); axis equal; axis tight; hold on;
plot(-n_final(:,1),-n_final(:,2),'*r'); axis equal; axis tight; hold on;set(
    gca,'color','black')
```

II. Code for CST Element:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
% Elastic Constant Strain Triangular Elements                             %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear workspace
clc
clear all

nod1= csvread('Nodes_1.csv');
nod2= csvread('Nodes_2.csv');
nod3= csvread('Nodes_3.csv');
nod4= csvread('Nodes_4.csv');
nod5= csvread('Nodes_5.csv');

Elm1=csvread('Elements_1.csv');
Elm2=csvread('Elements_2.csv');
Elm3=csvread('Elements_3.csv');
Elm4=csvread('Elements_4.csv');
Elm5=csvread('Elements_5.csv');

for no=1:5

    % 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;

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

    %Determine Derichlet BC
    for (i=1:N)
        if (Nodes(i,2)==0)
            DBC(j_dbc,1)=Nodes(i,1);
            DBC(j_dbc,2)=1;
            DBC(j_dbc,3)=0;
            j_dbc=j_dbc+1;
        end
        if (Nodes(i,3)==0)
            DBC(j_dbc,1)=Nodes(i,1);
            DBC(j_dbc,2)=2;
            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,2)==3)
            right(j_nbc,1)=Nodes(i,1);
            right(j_nbc,2)=1;
            right(j_nbc,3)=0;
            j_nbc=j_nbc+1;
        end
    end
    j_nbc=1;


    comb=combnk(right(:,1),2);

    for i=1:E
        for j=1:size(comb(:,1))
            if comb(j,:) == Elems(i,4:5) | comb(j,:) == Elems(i,5:6) | comb(j,:) == Elems(i,[4 6])| comb(j,:) == Elems(i,[6 4]) | comb(j,:) == Elems(i,[5 4]) | comb(j,:) == Elems(i,[6 5])
                NBC(j_nbc,1)=Elems(i,1);
                NBC(j_nbc,2:3)=comb(j,:);
                NBC(j_nbc,4)=1;
                NBC(j_nbc,5)=1;
                j_nbc=j_nbc+1;
            end
        end
    end
    [Q,l] = size(NBC);

    % Determining the hole nodes
    i_hol=1;
    for i=1:N
        if (Nodes(i,2)<=0.25 && Nodes(i,3)<=0.25)
            hole(i_hol)=Nodes(i,1);
            i_hol=i_hol+1;
        end
    end

    % Determining the hole elements
    i_hol=1;
    for i=1:E
        if (hole(i_hol)==Elems(i,4)||hole(i_hol)==Elems(i,5)||hole(i_hol)==Elems(i,6))
            hol_el(i_hol)=Elems(i,1);
            i_hol=i_hol+1;
        end
    end
```

```
    hol_el=unique(hol_el);

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

   % Determine total number of degrees-of-freedom
    udof = 2;      % 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 = 10^10;

   % Loop over CST element
    for e = 1:E

       % Establish element connectivity and coordinates
       Nnums = Elems(e,4:6);
       xy = Nodes(Nnums(:),2:3);

       % 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] = CST_El_Stiff(ipstrn,xy,h,Y,nu);

       % Assemble element stiffness matrix into global stiffness matrix
       ig = udof*(Nnums(:)-1);
       for ni = 1:3
           i0 = udof*(ni-1);
           for nj = 1:3
               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 q = 1:Q

        % Determine loaded edge
        e = NBC(q,1);
        in1 = NBC(q,2);
        in2 = NBC(q,3);
        idof = NBC(q,4);
        tval = NBC(q,5);
        h = Elems(e,3);

        % Establish edge length
        xlen2 = (Nodes(in2,2)-Nodes(in1,2))^2;
        ylen2 = (Nodes(in2,3)-Nodes(in1,3))^2;
        elen = sqrt(xlen2+ylen2);
        fval = tval*elen*h/2;
        iloc1 = udof*(in1-1)+idof;
        iloc2 = udof*(in2-1)+idof;
        F(iloc1) = F(iloc1) + fval;
        F(iloc2) = F(iloc2) + fval;
        F;

    end

    for p = 1:P
        inode = DBC(p,1);
        idof = DBC(p,2);
        idiag = udof*(inode-1) + idof;
        K(idiag,idiag) = Klarge;
        F(idiag) = Klarge*DBC(p,3);
    end
    % K
    % F

    % Solve system to determine displacements
    U = K\F;

    % Recover internal element displacement, strains and stresses
    Disp = zeros(E,6);
    Eps = zeros(E,3);
    Sig = zeros(E,3);

    for e = 1:E

        % Establish element connectivity and coordinates
```

```
        Nnums = Elems(e,4:6);
        xy = Nodes(Nnums(:),2:3);

        % 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
        inode1 = Nnums(1);
        inode2 = Nnums(2);
        inode3 = Nnums(3);
        Disp(e,1) = U(udof*(inode1-1)+1);
        Disp(e,2) = U(udof*inode1);
        Disp(e,3) = U(udof*(inode2-1)+1);
        Disp(e,4) = U(udof*inode2);
        Disp(e,5) = U(udof*(inode3-1)+1);
        Disp(e,6) = U(udof*inode3);

        u = Disp(e,:)';
        [eps,sig] = CST_El_Str(ipstrn,xy,u,h,Y,nu);

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

        % Store element stresses
        Sig(e,:) = sig;

    end

    % Computing Strain concentration factor

    sig_nom= 1/0.75;

    for i=1:size(hol_el)
        sig_max=max(Sig(hol_el(i),:,:));
    end
    SCF(no)=mean(sig_max)/sig_nom;


    PE(no,1)=0.5*U'*K*U;

    PE(no,2)=3/N;
%     if (no==1)
%         str=sprintf('Original plate vs deformed plate using constant strain triangle elements for %d elements',E);
%         figure;
```

```
%           Plot_deformation;
%           title(str);
%           xlabel('\leftarrow 2L \rightarrow');
%           ylabel('\leftarrow 2H \rightarrow');
          max(U)
%       end
%       E
     clearvars -except nod1 nod2 nod3 nod4 nod5 Elm1 Elm2 Elm3 Elm4 Elm5 PE SCF
    ;
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;
%
% Disp
% Eps
% Sig
```

```
function Ke = CST_El_Stiff(ipstrn,xy,h,Y,nu)

ndof = 6;
Ke = zeros(ndof,ndof);

Abar = [ 1 xy(1,1) xy(1,2); 1 xy(2,1) xy(2,2); 1 xy(3,1) xy(3,2) ];
A = det(Abar)/2;

B = (1/A/2)*[ xy(2,2)-xy(3,2) 0 xy(3,2)-xy(1,2) 0 xy(1,2)-xy(2,2) 0;
              0 xy(3,1)-xy(2,1) 0 xy(1,1)-xy(3,1) 0 xy(2,1)-xy(1,1);
              xy(3,1)-xy(2,1) xy(2,2)-xy(3,2) xy(1,1)-xy(3,1) ...
              xy(3,2)-xy(1,2) xy(2,1)-xy(1,1) xy(1,2)-xy(2,2) ];

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

```

```
Ke = h*A*B'*C*B;
```

```
function [eps,str] = CST_El_Str(ipstrn,xy,u,h,Y,nu)

ndof = 6;

Abar = [ 1 xy(1,1) xy(1,2); 1 xy(2,1) xy(2,2); 1 xy(3,1) xy(3,2) ];
A = det(Abar)/2;

B = (1/A/2)*[ xy(2,2)-xy(3,2) 0 xy(3,2)-xy(1,2) 0 xy(1,2)-xy(2,2) 0;
              0 xy(3,1)-xy(2,1) 0 xy(1,1)-xy(3,1) 0 xy(2,1)-xy(1,1);
              xy(3,1)-xy(2,1) xy(2,2)-xy(3,2) xy(1,1)-xy(3,1) ...
              xy(3,2)-xy(1,2) xy(2,1)-xy(1,1) xy(1,2)-xy(2,2) ];

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 = B*u;
str = C*eps;
```

III. Code for 4-noded quadrilateral isoparametric Element:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
% Elastic 4-node Quadralateral Elements                               %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear workspace
clc
clear

% Read nodes and coords
nod1= csvread('Nodes_1.csv');
nod2= csvread('Nodes_2.csv');
nod3= csvread('Nodes_3.csv');
nod4= csvread('Nodes_4.csv');
nod5= csvread('Nodes_5.csv');

Elm1=csvread('Elements_1.csv');
Elm2=csvread('Elements_2.csv');
Elm3=csvread('Elements_3.csv');
Elm4=csvread('Elements_4.csv');
Elm5=csvread('Elements_5.csv');

```

```
for no=1:5

    % 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 = l-3;

    % 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,2)==0)
        DBC(j_dbc,1)=Nodes(i,1);
        DBC(j_dbc,2)=1;
        DBC(j_dbc,3)=0;
        j_dbc=j_dbc+1;
    end
    if (Nodes(i,3)==0)
        DBC(j_dbc,1)=Nodes(i,1);
        DBC(j_dbc,2)=2;
        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,2)==3)
        right(j_nbc,1)=Nodes(i,1);
        right(j_nbc,2)=1;
        right(j_nbc,3)=0;
        j_nbc=j_nbc+1;
    end
end
j_nbc=1;


for i=1:E
    for j=1:size(right(:,1))
        for k=4:7

            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));
```

```matlab
    for i=1:2:size(el_list(:,1))
        for j=1:size(NBC(:,1))

            k=0;
            if (NBC(j,1)==el_list(i,1))
                NBC(j,2:3)=[el_list(i,2) el_list(i+1,2)];
            end
        end
    end
    NBC(:,4)=1;
    NBC(:,5)=1;
    [Q,l] = size(NBC);

    % Determining the hole nodes
    i_hol=1;
    for i=1:N
        if (Nodes(i,2)<=0.25 && Nodes(i,3)<=0.25)
            hole(i_hol)=Nodes(i,1);
            i_hol=i_hol+1;
        end
    end

    % Determining the hole elements
    i_hol=1;
    for i=1:E
        if (hole(i_hol)==Elems(i,4)||hole(i_hol)==Elems(i,5)||hole(i_hol)==Elems(i,6)||hole(i_hol)==Elems(i,7))
            hol_el(i_hol)=Elems(i,1);
            i_hol=i_hol+1;
        end
    end
    hol_el=unique(hol_el);
    % Determine total number of degrees-of-freedom
    udof = 2;      % 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 = 4;
    [XG,WG] = Q4_El_Gauss_Points(NG);

```

```
    % Loop over Q4 elements
    for e = 1:E

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

        % 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] = Q4_El_Stiff(ipstrn,xy,h,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 point locations and weights for traction integration
    NGS = 2;
    [XGS,WGS] = Q4_El_Gauss_Points_Surf(NGS);

    for q = 1:Q

        in   = zeros(NES);
        tval = zeros(NES,1);
        fval = zeros(NES,1);

        % Determine loaded edge
        e = NBC(q,1);
        in1 = NBC(q,2);
```

```
        in2 = NBC(q,3);
        idof = NBC(q,4);
        tval(:,1) = NBC(q,4:5);
        h = Elems(e,3);

        for i=1:NGS

            % Evaluate force contributions at Gauss points
            xi  = XGS(i);
            wgt = WGS(i);

            [NshapeS] = Q4_El_Shape_Surf(NES,xi);
            [DNshapeS] = Q4_El_DShape_Surf(NES,xi);

            xyS(1,1) = Nodes(in1,2);
            xyS(1,2) = Nodes(in1,3);
            xyS(2,1) = Nodes(in2,2);
            xyS(2,2) = Nodes(in2,3);
            [detJS] = Q4_El_Jacobian_Surf(NES,xi,xyS,DNshapeS);

            fval = fval + h*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);
        idof = DBC(p,2);
        idiag = udof*(inode-1) + idof;
        K(idiag,idiag) = Klarge;
        F(idiag) = Klarge*DBC(p,3);
    end
    %K
    %F

    % Solve system to determine displacements
    U = 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,4:3+NE);
        xy = Nodes(Nnums(:),2:3);

        % 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] = Q4_El_Str(ipstrn,xy,u,h,Y,nu,udof,NE,NG,XG);
        %eps
        %sig

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

        % Store element stresses
        Sig(e,:,:) = sig(:,:);

    end

   % Computing Strain concentration factor

    sig_nom= 1/0.75;

    for i=1:size(hol_el)
        sig_max=max(Sig(hol_el(i),:,:));
    end
```

```
    SCF(no)=mean(sig_max)/sig_nom;

    PE(no,1)=0.5*U'*K*U;

    PE(no,2)=3/N;
%       if(no==1)
%           str=sprintf('Original plate vs deformed plate using 4 Noded Quad elements for %d elements',E);
%           figure;
%           Plot_deformation;
%           title(str);
%           xlabel('\leftarrow 2L \rightarrow');
%           ylabel('\leftarrow 2H \rightarrow');
            max(U)
%       end
    E
    clearvars -except nod1 nod2 nod3 nod4 nod5 Elm1 Elm2 Elm3 Elm4 Elm5 PE SCF;
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;
% Disp;
% Eps;
% Sig;
```

```
function [DNshape] = Q4_El_DShape(NE,xi,eta)

DNshape(1,1) = -(1-eta)/4;
DNshape(2,1) = +(1-eta)/4;
DNshape(3,1) = +(1+eta)/4;
DNshape(4,1) = -(1+eta)/4;

DNshape(1,2) = -(1-xi)/4;
DNshape(2,2) = -(1+xi)/4;
DNshape(3,2) = +(1+xi)/4;
DNshape(4,2) = +(1-xi)/4;
```

```
function [DNshapeS] = Q4_El_DShape_Surf(NES,xi)
```

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

```
function [XG,WG] = Q4_El_Gauss_Points(NG)

if (NG == 4)

    alf = sqrt(1/3);

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

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

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

else

    alf = sqrt(3/5);

    XG(1,1) = -alf;
    XG(2,1) = 0;
    XG(3,1) = +alf;
    XG(4,1) = -alf;
    XG(5,1) = 0;
    XG(6,1) = +alf;
    XG(7,1) = -alf;
    XG(8,1) = 0;
    XG(9,1) = +alf;

    XG(1,2) = -alf;
    XG(2,2) = -alf;
    XG(3,2) = -alf;
    XG(4,2) = 0;
    XG(5,2) = 0;
    XG(6,2) = 0;
    XG(7,2) = +alf;
    XG(8,2) = +alf;
    XG(9,2) = +alf;

    WG(1) = 25/81;
    WG(2) = 40/81;
```

```
    WG(3) = 25/81;
    WG(4) = 40/81;
    WG(5) = 64/81;
    WG(6) = 40/81;
    WG(7) = 25/81;
    WG(8) = 40/81;
    WG(9) = 25/81;

end
```

```
function [XGS,WGS] = Q4_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;

else

     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;

end
```

```
function [Jac,detJ,Jhat] = Q4_El_Jacobian(NE,xi,eta,xy,DNshape)

Jac = zeros(2,2);

for i=1:NE
     Jac(1,1) = Jac(1,1) + DNshape(i,1)*xy(i,1);
     Jac(1,2) = Jac(1,2) + DNshape(i,1)*xy(i,2);
     Jac(2,1) = Jac(2,1) + DNshape(i,2)*xy(i,1);
     Jac(2,2) = Jac(2,2) + DNshape(i,2)*xy(i,2);
end

detJ = det(Jac);
Jhat = inv(Jac);
```

```
function [detJS] = Q4_El_Jacobian_Surf(NES, xi, xyS, DNshapeS)

dxdxi = 0;
dydxi = 0;

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

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

```
function [Nshape] = Q4_El_Shape(NE, xi, eta)

Nshape(1) = (1-xi)*(1-eta)/4;
Nshape(2) = (1+xi)*(1-eta)/4;
Nshape(3) = (1+xi)*(1+eta)/4;
Nshape(4) = (1-xi)*(1+eta)/4;
```

```
function [NshapeS] = Q4_El_Shape_Surf(NES, xi)

NshapeS(1) = (1-xi)/2;
NshapeS(2) = (1+xi)/2;

%NshapeS = NshapeS';
```

```
function Ke = CST_El_Stiff(ipstrn, xy, h, Y, nu, udof, NE, NG, XG, WG)

ndof = NE*udof;
nstrn = 3;
Ke = zeros(ndof, ndof);

for i=1:NG

    xi  = XG(i,1);
    eta = XG(i,2);
    wgt = WG(i);

    %[Nshape] = Q4_El_Shape(NE, xi, eta);
    [DNshape] = Q4_El_DShape(NE, xi, eta);
    [Jac, detJ, Jhat] = Q4_El_Jacobian(NE, xi, eta, xy, 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

    Ke = Ke + h*wgt*B'*C*B*detJ;

end
```

```
function [eps,sig] = Q4_El_Str(ipstrn,xy,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);

    [DNshape] = Q4_El_DShape(NE,xi,eta);
    [Jac,detJ,Jhat] = Q4_El_Jacobian(NE,xi,eta,xy,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
```

IV. Code for 4-noded quadrilateral isoparametric Element:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
% Elastic 8-node Quadralateral Elements                                        %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear workspace
clc
clear
close all
% Read nodes and coords
nod1= csvread('Nodes_1.csv');
nod2= csvread('Nodes_2.csv');
nod3= csvread('Nodes_3.csv');
nod4= csvread('Nodes_4.csv');
nod5= csvread('Nodes_5.csv');

Elm1=csvread('Elements_1.csv');
Elm2=csvread('Elements_2.csv');
Elm3=csvread('Elements_3.csv');
Elm4=csvread('Elements_4.csv');
Elm5=csvread('Elements_5.csv');

for no=1:5

    % 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 = l-3;

    % 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,2)==0)
            DBC(j_dbc,1)=Nodes(i,1);
            DBC(j_dbc,2)=1;
            DBC(j_dbc,3)=0;
            j_dbc=j_dbc+1;
        end
        if (Nodes(i,3)==0)
            DBC(j_dbc,1)=Nodes(i,1);
            DBC(j_dbc,2)=2;
```

```
                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,2)==3)
                right(j_nbc,1)=Nodes(i,1);
                right(j_nbc,2)=1;
                right(j_nbc,3)=0;
                j_nbc=j_nbc+1;
            end
        end
        j_nbc=1;
        for i=1:E
            for j=1:size(right(:,1))
                for k=4:11
                    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=1:3:size(el_list(:,1))
            for j=4:7
                if (el_list(i,2)==Elems(el_list(i,1),j) || el_list(i+1,2)==Elems(el_list(i,1),j))
                    if (el_list(i,2)==Elems(el_list(i,1),j))
                        NBC(j_nbc,2)=el_list(i,2);
                        NBC(j_nbc,4)=el_list(i+1,2);
                        NBC(j_nbc,3)=el_list(i+2,2);
                        j_nbc=j_nbc+1;
                        break;
                    else
                        NBC(j_nbc,2)=el_list(i+1,2);
                        NBC(j_nbc,4)=el_list(i,2);
                        NBC(j_nbc,3)=el_list(i+2,2);
                        j_nbc=j_nbc+1;
                        break;
                    end
                end
            end
        end
```

```
    NBC(:,5)=1;
    NBC(:,6)=1;
    [Q,l] = size(NBC);


    % Determining the hole nodes
    i_hol=1;
    for i=1:N
        if (Nodes(i,2)<=0.25 && Nodes(i,3)<=0.25)
            hole(i_hol)=Nodes(i,1);
            i_hol=i_hol+1;
        end
    end

    % Determining the hole elements
    i_hol=1;
    for i=1:E
        if (hole(i_hol)==Elems(i,4)||hole(i_hol)==Elems(i,5)||hole(i_hol)==Elems(i,6)||hole(i_hol)==Elems(i,7)||hole(i_hol)==Elems(i,8)||hole(i_hol)==Elems(i,9)||hole(i_hol)==Elems(i,10)||hole(i_hol)==Elems(i,11))
            hol_el(i_hol)=Elems(i,1);
            i_hol=i_hol+1;
        end
    end
    hol_el=unique(hol_el);


    % Determine total number of degrees-of-freedom
    udof = 2;      % 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 = 4;
    [XG,WG] = Q8_El_Gauss_Points(NG);

    % Loop over Q8 elements
    for e = 1:E

        % Establish element connectivity and coordinates
```

```
        Nnums = Elems(e,4:3+NE);
        xy = Nodes(Nnums(:),2:3);

        % 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] = Q8_El_Stiff(ipstrn,xy,h,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 = 3;
    % Set Gauss pint locations and weights for traction integration
    NGS = 3;
    [XGS,WGS] = Q8_El_Gauss_Points_Surf(NGS);

    for q = 1:Q

        in   = zeros(NES);
        tval = zeros(NES,1);
        fval = zeros(NES,1);

        % Determine loaded edge
        e = NBC(q,1);
        in1 = NBC(q,2);
        in2 = NBC(q,3);
        in3 = NBC(q,4);
        idof = NBC(q,5);
        tval(:,1) = NBC(q,6);
```

```
        h = Elems(e,3);

        for i=1:NGS

            % Evaluate force contributions at Gauss points
            xi  = XGS(i);
            wgt = WGS(i);

            [NshapeS] = Q8_El_Shape_Surf(NES,xi);
            [DNshapeS] = Q8_El_DShape_Surf(NES,xi);

            xyS(1,1) = Nodes(in1,2);
            xyS(1,2) = Nodes(in1,3);
            xyS(2,1) = Nodes(in2,2);
            xyS(2,2) = Nodes(in2,3);
            xyS(3,1) = Nodes(in3,2);
            xyS(3,2) = Nodes(in3,3);
            [detJS] = Q8_El_Jacobian_Surf(NES,xi,xyS,DNshapeS);

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

        end
        %     fval

        iloc1 = udof*(in1-1)+idof;
        iloc2 = udof*(in2-1)+idof;
        iloc3 = udof*(in3-1)+idof;
        F(iloc1) = F(iloc1) + fval(1);
        F(iloc2) = F(iloc2) + fval(2);
        F(iloc3) = F(iloc3) + fval(3);
        %F

    end

    % Impose Dirichlet boundary conditions
    for p = 1:P
        inode = DBC(p,1);
        idof = DBC(p,2);
        idiag = udof*(inode-1) + idof;
        K(idiag,idiag) = Klarge;
        F(idiag) = Klarge*DBC(p,3);
    end
    %K
    %F

    % Solve system to determine displacements
    U = 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,4:3+NE);
        xy = Nodes(Nnums(:),2:3);

        % 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] = Q8_El_Str(ipstrn,xy,u,h,Y,nu,udof,NE,NG,XG);
        %eps
        %sig

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

        % Store element stresses
        Sig(e,:,:) = sig(:,:);

    end

    % Computing Strain concentration factor

    sig_nom= 1/0.75;

    for i=1:size(hol_el)
        sig_max=max(Sig(hol_el(i),:,:));
```

```
    end
    SCF(no)=mean(sig_max)/sig_nom;

    PE(no,1)=0.5*U'*K*U;

    PE(no,2)=3/N;
%     if (no==1)
%         str=sprintf('Original plate vs deformed plate using 8 Noded Quad elements for %d elements',E);
%         figure;
%         Plot_deformation;
%         title(str);
%         xlabel('\leftarrow 2L \rightarrow');
%         ylabel('\leftarrow 2H \rightarrow');
        max(U)
%     end
    E
    clearvars -except nod1 nod2 nod3 nod4 nod5 Elm1 Elm2 Elm3 Elm4 Elm5 PE SCF;
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;
% Disp
% Eps
% Sig
```

```
function [DNshape] = Q8_El_DShape(NE,xi,eta)


DNshape(:,1)=[- (xi/4 - 1/4)*(eta - 1) - ((eta - 1)*(eta + xi + 1))/4;
              ((eta - 1)*(eta - xi + 1))/4 - (xi/4 + 1/4)*(eta - 1);
              (xi/4 + 1/4)*(eta + 1) + ((eta + 1)*(eta + xi - 1))/4;
              (xi/4 - 1/4)*(eta + 1) + ((eta + 1)*(xi - eta + 1))/4;
                                                      xi*(eta - 1);
                                                    1/2 - eta^2/2;
                                                     -xi*(eta + 1);
                                                   eta^2/2 - 1/2]';

```

```
DNshape(:,2)=[- (xi/4 - 1/4)*(eta - 1) - (xi/4 - 1/4)*(eta + xi + 1);
              (xi/4 + 1/4)*(eta - xi + 1) + (xi/4 + 1/4)*(eta - 1);
              (xi/4 + 1/4)*(eta + 1) + (xi/4 + 1/4)*(eta + xi - 1);
              (xi/4 - 1/4)*(xi - eta + 1) - (xi/4 - 1/4)*(eta + 1);
                                                 xi^2/2 - 1/2;
                                                 -eta*(xi + 1);
                                                 1/2 - xi^2/2;
                                                 eta*(xi - 1)]';
```

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

DNshapeS(1) = xi - 1/2;
DNshapeS(2) = -2*xi;
DNshapeS(3) = -xi - 1/2;
```

```
function [XG,WG] = Q8_El_Gauss_Points(NG)

if (NG == 4)

    alf = sqrt(1/3);

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

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

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

else

    alf = sqrt(3/5);

    XG(1,1) = -alf;
    XG(2,1) = 0;
    XG(3,1) = +alf;
    XG(4,1) = -alf;
    XG(5,1) = 0;
    XG(6,1) = +alf;
    XG(7,1) = -alf;
    XG(8,1) = 0;
    XG(9,1) = +alf;

```

```
    XG(1,2) = -alf;
    XG(2,2) = -alf;
    XG(3,2) = -alf;
    XG(4,2) = 0;
    XG(5,2) = 0;
    XG(6,2) = 0;
    XG(7,2) = +alf;
    XG(8,2) = +alf;
    XG(9,2) = +alf;

    WG(1) = 25/81;
    WG(2) = 40/81;
    WG(3) = 25/81;
    WG(4) = 40/81;
    WG(5) = 64/81;
    WG(6) = 40/81;
    WG(7) = 25/81;
    WG(8) = 40/81;
    WG(9) = 25/81;

end
```

```
function [XGS,WGS] = Q8_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;

else

    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;

end
```

```
function [Jac,detJ,Jhat] = Q8_El_Jacobian(NE,xi,eta,xy,DNshape)

```

```
Jac = zeros(2,2);

for i=1:NE
    Jac(1,1) = Jac(1,1) + DNshape(i,1)*xy(i,1);
    Jac(1,2) = Jac(1,2) + DNshape(i,1)*xy(i,2);
    Jac(2,1) = Jac(2,1) + DNshape(i,2)*xy(i,1);
    Jac(2,2) = Jac(2,2) + DNshape(i,2)*xy(i,2);
end

detJ = det(Jac);
Jhat = inv(Jac);
```

```
function [detJS] = Q8_El_Jacobian_Surf(NES,xi,xyS,DNshapeS)

dxdxi = 0;
dydxi = 0;

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

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

```
function [Nshape] = Q8_El_Shape(NE,xi,eta)


Nshape = [-1/4*(1-xi)*(1-eta)*(xi+eta+1);
           1/4*(1+xi)*(1-eta)*(xi-eta-1);
           1/4*(1+xi)*(1+eta)*(xi+eta-1);
          -1/4*(1-xi)*(1+eta)*(xi-eta+1);
                1/2*(1-xi^2)*(1-eta);
                1/2*(1-eta^2)*(1+xi);
                1/2*(1-xi^2)*(1+eta);
               1/2*(1-eta^2)*(1-xi)]';
```

```
function [NshapeS] = Q8_El_Shape_Surf(NES,xi)

NshapeS(1) = ((xi-0)*(xi-1))/((-1-0)*(-1-1));
NshapeS(2) = ((xi+1)*(xi-1))/((0+1)*(0-1));
NshapeS(3) = ((xi+1)*(xi-0))/((1+1)*(0-1));

%NshapeS = NshapeS';
```

```
function [Ke] = CST_El_Stiff(ipstrn,xy,h,Y,nu,udof,NE,NG,XG,WG)

ndof = NE*udof;
nstrn = 3;
Ke = zeros(ndof,ndof);

```

```
for i=1:NG

    xi  = XG(i,1);
    eta = XG(i,2);
    wgt = WG(i);

   %[Nshape] = Q8_El_Shape(NE,xi,eta);
    [DNshape] = Q8_El_DShape(NE,xi,eta);
    [Jac,detJ,Jhat] = Q8_El_Jacobian(NE,xi,eta,xy,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

    Ke = Ke + h*wgt*B'*C*B*detJ;

end
```

```
function [eps,sig] = Q8_El_Str(ipstrn,xy,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);

    [DNshape] = Q8_El_DShape(NE,xi,eta);
    [Jac,detJ,Jhat] = Q8_El_Jacobian(NE,xi,eta,xy,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
```