

# ME 5310 - Finite Element Project

**Name:** Shuvo Chowdhury

Student ID: 1002238403

## 2D FE Code

### Algorithm

The finite element code is organized into four modular files to separate the global logic from the element-level mathematics.

1. **MainProject.m (Driver Script):** This is the primary execution file. It handles input definitions, mesh generation, the global assembly loop, boundary condition applications, the linear solver, and output plotting.
2. **Quad4Stiffness.m (Function):** A helper function called during the assembly phase. It accepts element coordinates and material properties, performs Gaussian quadrature, and returns the  $8 \times 8$  element stiffness matrix ( $\mathbf{k}^e$ ).
3. **CalcStress.m (Function):** A helper function called during post-processing. It accepts deformed element coordinates and nodal displacements to calculate the stress tensor ( $\boldsymbol{\sigma}$ ) and Von Mises stress.
4. **PlotMesh.m (Function):** A helper function to plot the deformed and undeformed configuration of the mesh.

### Step 1: Initialization and Input

1. Define material properties: Young's Modulus ( $E$ ), Poisson's Ratio ( $\nu$ ), and thickness ( $t$ ).
2. Define the mesh:
  - Load the **Nodes** list (coordinates  $x, y$ ).
  - Load the **Elements** connectivity list (node IDs for each element).
3. Initialize the global stiffness matrix  $\mathbf{K}$  (size  $2N \times 2N$ ) with zeros.
4. Initialize the global force vector  $\mathbf{F}$  (size  $2N \times 1$ ) with zeros.
5. Calculate the material constitutive matrix  $\mathbf{D}$  for Plane Stress conditions.

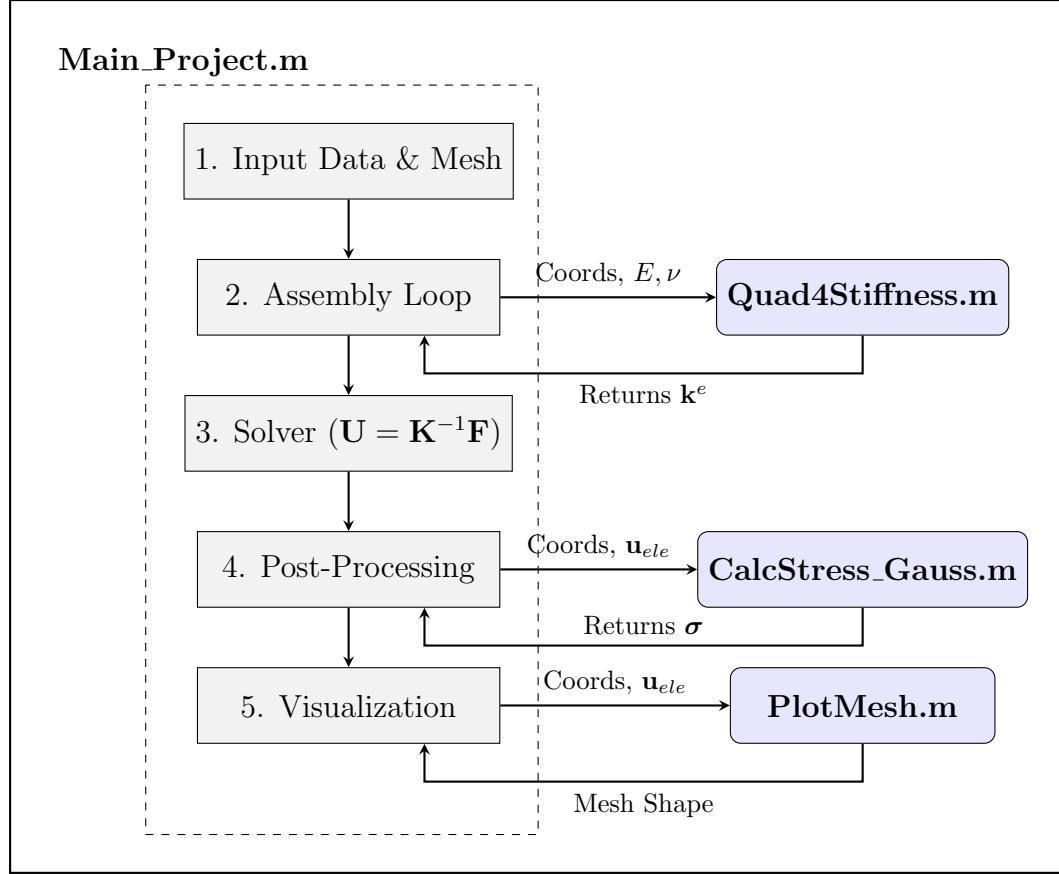


Figure 1: Data flow diagram showing the interaction between the main driver script and the helper functions.

## Step 2: Stiffness Matrix Assembly

Loop over every element in the mesh:

1. Extract the coordinates of the 4 nodes belonging to the current element.

2. Initialize the element stiffness matrix  $\mathbf{k}^e$  (size  $8 \times 8$ ).

3. Numerical Integration (Gaussian Quadrature):

- (a) Loop through the 4 Gauss points ( $\xi = \pm 0.577, \eta = \pm 0.577$ ).
- (b) Calculate derivatives of shape functions with respect to natural coordinates  $(\xi, \eta)$ .
- (c) Compute the Jacobian matrix  $\mathbf{J}$  and its determinant  $|\mathbf{J}|$ .
- (d) Transform derivatives to physical coordinates  $(x, y)$  using  $\mathbf{J}^{-1}$ .
- (e) Construct the strain-displacement matrix  $\mathbf{B}$ .
- (f) Compute the local stiffness contribution:  $\mathbf{B}^T \mathbf{D} \mathbf{B} \cdot |\mathbf{J}| \cdot \text{weight} \cdot \text{thickness}$ .
- (g) Add this contribution to  $\mathbf{k}^e$ .

4. Assembly: Map the local indices of  $\mathbf{k}^e$  to the global indices in  $\mathbf{K}$  and add the values.

## Step 3: Boundary Conditions and Loading

1. Apply external forces to the global force vector  $\mathbf{F}$  at the specified node degrees of freedom (DOFs).

2. Identify the list of **Fixed DOFs** (where displacement is constrained to 0).
3. Identify the list of **Free DOFs** (the unknowns we need to solve for).

## Step 4: Solution

1. Extract the sub-matrix  $\mathbf{K}_{free}$  containing only the rows/columns of Free DOFs.
2. Extract the sub-vector  $\mathbf{F}_{free}$  containing only the forces at Free DOFs.
3. Solve the linear system of equations:

$$\mathbf{K}_{free} \cdot \mathbf{U}_{free} = \mathbf{F}_{free}$$

4. Reconstruct the full displacement vector  $\mathbf{U}$  by filling Fixed DOFs with zeros.

## Step 5: Post-Processing (Stress Calculation)

Loop over every element again:

1. Extract the computed displacements for the element's nodes.
2. Calculate the  $\mathbf{B}$  matrix at the element centroid (or integration points).
3. Compute Strain:  $\boldsymbol{\epsilon} = \mathbf{B} \cdot \mathbf{u}_{element}$ .
4. Compute Stress:  $\boldsymbol{\sigma} = \mathbf{D} \cdot \boldsymbol{\epsilon}$ .
5. Calculate Von Mises stress for yield criteria checking.

# Matlab Code

## Main\_Project.m

```
1 % =====
2 % ME 5310 - Finite Element Project
3 % 2D Plane Stress Elasticity Solver
4 % File: Main_Project.m (Driver Script)
5 % =====
6 clc; clear; close all;
7 %% 1. Input Data (Set up for test case A: Single Element Validation)
8 %-----
9 % Material Properties
10 E = 30e6;
11 nu = 0.3;
12 t = 1; %Thickness
13
14 % Mesh Definition
15 % Nodes List: [Node ID, x-coord, y-coord]
16 Nodes = [
17     1, 0.0, 0.0;
18     2, 1.0, 0.0;
19     3, 1.0, 1.0;
20     4, 0.0, 1.0
21 ];
22
23 % Element Connectivity: [Elem ID, Node 1, Node 2, Node 3, Node 4]
24 % Note: Nodes must be listed in counter-clockwise order.
25 Elements = [
26     1, 1, 2, 3, 4;
27 ];
28
29 % Problem Size Parameters
30 num_nodes = size(Nodes, 1);
31 num_elems = size(Elements, 1);
32 num_dofs = num_nodes * 2; % 2 DOFs per node (u, v)
33
34 %% 2. INITIALIZATION
35 % -----
36 % Initialize global stiffness matrix (Sparse is better for large problems
37 % )
37 K_global = sparse(num_dofs, num_dofs);
38 F_global = zeros(num_dofs, 1);
39 U_global = zeros(num_dofs, 1);
40
41 %% 3. ASSEMBLY PROCESS
42 % -----
43 disp('Assembling Global Stiffness Matrix...');

44
45 for e = 1: num_elems
46     % A. extract node IDs for this element
47     node_ids = Elements(e, 2:5);
```

```

49 % B. Extract coordinates for this node
50 ele_coords = Nodes(node_ids, 2:3);
51
52 % C. Compute element stiffness matrix
53 k_e = Quad4Stiffness (ele_coords, E, nu, t);
54
55 % D. Mapping to the global matrix by a scatter vector
56
57 scatter = zeros (1,8);
58
59 for n = 1:4
60     global_node= node_ids(n);
61     scatter(2*n-1) = 2*global_node - 1; % x-dof (Odd index)
62     scatter(2*n)    = 2*global_node;      % y-dof (Even index)
63 end
64
65 % E. Add to global matrix
66 K_global(scatter, scatter) = K_global(scatter, scatter) + k_e;
67
68 end
69
70
71 %% 4. APPLY BOUNDARY CONDITIONS (BCs) & LOADS
72 % -----
73
74 fixed_nodes_u = [1, 4]; % Nodes fixed in X
75 fixed_nodes_v = [1, 4]; % Nodes fixed in Y
76
77 % Convert Node IDs to DOF Indices
78 fixed_dofs = [ (2*fixed_nodes_u - 1), (2*fixed_nodes_v) ]; % [1, 2, 4, 7]
79 fixed_dofs = unique(fixed_dofs); % Remove duplicates % [1, 2, 4, 7]
80
81 % Free DOFs are all other DOFs
82 all_dofs = 1:num_dofs; % [1, 2, 3, 4, 5, 6,7,8]
83 free_dofs = setdiff(all_dofs, fixed_dofs); % [3, 5, 6,8]
84
85 % B. Apply External Forces (Point Loads)
86 % Test Case Setup: Apply Tension in Y-direction at top nodes (3 & 4)
87 % Load P = 1000 lbs split between top nodes.
88 P_load = 100;
89 load_nodes = [2,3];
90
91
92 for i = 1:length(load_nodes)
93     node = load_nodes(i);
94     dof_y = 2*node-1; % y-dof
95     F_global(dof_y) = F_global(dof_y) + P_load;
96 end
97
98 %% 5. SOLVER
99 % -----
100 disp('Solving system equations...');


```

```

102 % Partition the system (Extract only free DOFs)
103 K_ff = K_global(free_dofs, free_dofs);
104 F_f = F_global(free_dofs);
105
106 % Solve for unknown displacements [cite: 65]
107 U_f = K_ff \ F_f;
108
109 % Reconstruct the full displacement vector
110 U_global(free_dofs) = U_f;
111 U_global(fixed_dofs) = 0; % Enforce zero at fixed supports
112
113 disp('Nodal Displacements:');
114 disp(U_global);
115
116 % %% 6. POST-PROCESSING
117 % % -----
118 disp('Calculating Element Stresses at Integration Points...');

119
120 for e = 1:num_elems
121     node_ids = Elements(e, 2:5);
122     ele_coords = Nodes(node_ids, 2:3);

123
124     % Extract displacements
125     u_ele = zeros(8,1);
126     for n = 1:4
127         global_node = node_ids(n);
128         u_ele(2*n-1) = U_global(2*global_node - 1);
129         u_ele(2*n) = U_global(2*global_node);
130     end

131
132     % Calculate Stress at 4 Gauss Points
133     [stress_mat, vm_vec] = CalcStress_Gauss(ele_coords, u_ele, E, nu);

134
135     % Print Results
136     fprintf('\nElement %d Results:\n', e);
137     fprintf(' GP | Sig_xx | Sig_yy | Tau_xy | VonMises\n');
138     fprintf(' ---|-----|-----|-----|-----\n');
139     for gp = 1:4
140         fprintf(' %d | %8.2f | %8.2f | %8.2f | %8.2f\n', ...
141                 gp, stress_mat(gp,1), stress_mat(gp,2), stress_mat(gp,3),
142                 vm_vec(gp));
143     end
144 end

145 disp('Analysis Complete.');
146 disp('Plotting input mesh...');

147 PlotMesh(Nodes, Elements);

148
149 disp('Plotting deformed shape...');

150 % scale = 1000; % Exaggerate deformation to make it visible
151 % PlotMesh(Nodes, Elements, U_global, 1);
152 disp('Plotting Superimposed shape')

```

```

154 figure;
155 hold on;
156 % 1. Plot undeformed mesh
157 PlotMesh(Nodes, Elements);
158 scale = 10000; % Exaggerate deformation to make it visible
159 PlotMesh(Nodes, Elements, U_global, scale);
160
161 %title(['Superimposed Deformation (Scale: ', num2str(scale), ')']);
162 axis equal;
163 grid on;
164 hold off; % Release the hold

```

## Quad4Stiffness.m

```

1 % =====
2 % Function: Quad4Stiffness
3 % Purpose: Calculates the 8x8 Element Stiffness Matrix for a
4 %           4-Node Isoparametric Quadrilateral in Plane Stress.
5 %
6 % Inputs:
7 %   coords - 4x2 matrix of node coordinates [x1 y1; x2 y2; x3 y3; x4 y4]
8 %   E       - Young's Modulus
9 %   nu      - Poisson's Ratio
10 %   t       - Thickness
11 %
12 % Outputs:
13 %   ke      - 8x8 Element Stiffness Matrix
14 % =====
15
16 function [ke] = Quad4Stiffness(coords, E, nu, t)
17
18 % 1. CONSTITUTIVE MATRIX (D) - Plane Stress Setup
19 C = E/(1-nu^2);
20 D = C* [1 nu 0;
21          nu 1 0;
22          0 0 (1-nu)/2];
23 pt = 1 / sqrt(3); % Location of points +/- 0.57735
24 w = 1.0; % Weight is 1.0 for all points in 2x2
25 rule
26
27 % Gauss points (xi, eta)
28 GP_xi = [-pt, pt, pt, -pt];
29 GP_eta = [-pt, -pt, pt, pt];
30
31 % Initialize Element Stiffness Matrix
32 ke = zeros(8, 8);
33
34 % 3. Integration points
35
36 for i = 1:4
37     xi = GP_xi(i);
38     eta = GP_eta(i);

```

```

38
39 % N1 = 0.25*(1-xi)*(1-eta)
40 % N2 = 0.25*(1+xi)*(1-eta)
41 % N3 = 0.25*(1+xi)*(1+eta)
42 % N4 = 0.25*(1-xi)*(1+eta)

43
44 % dN_dxi (1x4 vector)
45 dN_dxi = 0.25 * [ -(1-eta), (1-eta), (1+eta), -(1+eta) ];

46
47 % dN_deta (1x4 vector)
48 dN_deta = 0.25 * [ -(1-xi), -(1+xi), (1+xi), (1-xi) ];

49
50 % B. JACOBIAN MATRIX (J)
51 % J = [ dx/dxi dy/dxi ]
52 % [ dx/deta dy/deta]
53 % computed as (dN_dnatural * coords)

54
55 J = [dN_dxi; dN_deta] * coords;

56
57 detJ = det(J); % Determinant of Jacobian (Area scaling factor)

58
59 % Check for distorted elements (Area must be positive)
60 if detJ <= 0
61     error('Element Jacobian is non-positive. Check node numbering
62             (Counter-Clockwise).');
63 end

64
65 % C. DERIVATIVES wrt PHYSICAL COORDS (x, y)
66 % [dN_dx] = inv(J) * [dN_dxi]
67 % [dN_dy]           [dN_deta]

68
69 %invJ = inv(J);
70 dN_dxy = J\ [dN_dxi; dN_deta]; % invJ * [dN_dxi; dN_deta]; % 2x4
71 % Matrix

72
73 dN_dx = dN_dxy(1, :); % Top row is dN/dx
74 dN_dy = dN_dxy(2, :); % Bottom row is dN/dy

75
76 % D. ASSEMBLE B-MATRIX (Strain-Displacement Matrix) [cite: 62]
77 % B is 3x8. Each node "j" fills 2 columns.
78 % Format:
79 % [ N1,x   0       N2,x   0       ... ]
80 % [ 0       N1,y   0       N2,y   ... ]
81 % [ N1,y   N1,x   N2,y   N2,x   ... ]

82 B = zeros(3, 8);
83 for n = 1:4
84     col_u = 2*n - 1; % Column for u-displacement
85     col_v = 2*n;       % Column for v-displacement

86     B(1, col_u) = dN_dx(n);
87     B(1, col_v) = 0;

```

```

89         B(2, col_u) = 0;
90         B(2, col_v) = dN_dy(n);
91
92         B(3, col_u) = dN_dy(n);
93         B(3, col_v) = dN_dx(n);
94     end
95
96     % E. STIFFNESS ACCUMULATION
97     % ke = Sum ( B' * D * B * detJ * weight * thickness )
98     ke = ke + (B' * D * B) * detJ * w * w * t;
99 end
100 end

```

## CalStress\_Gauss.m

```

1 function [stress_matrix, vm_vec] = CalcStress_Gauss(coords, u_ele, E, nu)
2 % =====
3 % Function: CalcStress_Gauss
4 % Purpose: Calculates stress at ALL 4 Integration Points (Gauss Points)
5 %
6 % Inputs:
7 % coords - 4x2 matrix of node coordinates
8 % u_ele - 8x1 vector of element displacements
9 % E, nu - Material properties
10 %
11 % Outputs:
12 % stress_matrix - 4x3 matrix containing stress at each Gauss point.
13 %                   Rows: Gauss Point 1 to 4
14 %                   Cols: [Sig_xx, Sig_yy, Tau_xy]
15 % vm_vec - 4x1 vector of Von Mises stress at each Gauss point
16 % =====
17
18 % 1. CONSTITUTIVE MATRIX (D)
19 C = E / (1 - nu^2);
20 D = C * [ 1   nu   0 ;
21           nu   1   0 ;
22           0   0   (1-nu)/2 ];
23
24 % 2. GAUSS POINTS (Same as Stiffness Matrix)
25 pt = 1 / sqrt(3);
26 GP_xi = [-pt, pt, pt, -pt];
27 GP_eta = [-pt, -pt, pt, pt];
28
29 % Initialize Outputs
30 stress_matrix = zeros(4, 3);
31 vm_vec = zeros(4, 1);
32
33 % 3. LOOP OVER GAUSS POINTS
34 for i = 1:4
35     xi = GP_xi(i);
36     eta = GP_eta(i);

```

```

38 % A. Derivatives at this Gauss Point
39 dN_dxi = 0.25 * [ -(1-eta), (1-eta), (1+eta), -(1+eta) ];
40 dN_deta = 0.25 * [ -(1-xi), -(1+xi), (1+xi), (1-xi) ];
41
42 % B. Jacobian
43 J = [dN_dxi; dN_deta] * coords;
44 invJ = inv(J);
45
46 % C. Global Derivatives
47 dN_dxy = invJ * [dN_dxi; dN_deta];
48 dN_dx = dN_dxy(1, :);
49 dN_dy = dN_dxy(2, :);
50
51 % D. B-Matrix Construction
52 B = zeros(3, 8);
53 for n = 1:4
54     col_u = 2*n - 1;
55     col_v = 2*n;
56     B(1, col_u) = dN_dx(n);
57     B(1, col_v) = 0;
58     B(2, col_u) = 0;
59     B(2, col_v) = dN_dy(n);
60     B(3, col_u) = dN_dy(n);
61     B(3, col_v) = dN_dx(n);
62 end
63
64 % E. Calculate Stress
65 epsilon = B * u_ele;
66 sigma = D * epsilon;
67
68 % Store Sig_xx, Sig_yy, Tau_xy
69 stress_matrix(i, :) = sigma';
70
71 % Calculate Von Mises for this point
72 sig_x = sigma(1);
73 sig_y = sigma(2);
74 tau_xy = sigma(3);
75 vm_vec(i) = sqrt(sig_x^2 + sig_y^2 - sig_x*sig_y + 3*tau_xy^2);
76 end
77 end

```

## PlotMesh.m

```

1 function PlotMesh(Nodes, Elements, U_global, scale_factor)
2 % =====
3 % Function: PlotMesh
4 % Purpose: Plots the finite element mesh. Can show deformed shape.
5 %
6 % Inputs:
7 %   Nodes      - N x 3 matrix [ID, x, y]
8 %   Elements   - M x 5 matrix [ID, n1, n2, n3, n4]

```

```

9 %     U_global      - (Optional) Displacement vector. Pass [] if not needed
10 %
11 %     scale_factor - (Optional) Scale factor for deformation. Default = 0.
12 % =====
13
14 % Handle optional arguments
15 if nargin < 3
16     U_global = [];
17     scale_factor = 0;
18 elseif nargin < 4
19     scale_factor = 1.0;
20 end
21
22 % 1. Prepare Vertices (Coordinates)
23 % Extract X and Y columns (Cols 2 and 3)
24 coords = Nodes(:, 2:3);
25
26 % If displacements are provided, add them to coordinates
27 if ~isempty(U_global) && scale_factor ~= 0
28     % Reshape U into (N_nodes x 2) to match coords
29     % U_global is [u1; v1; u2; v2...]
30
31     disp_x = U_global(1:2:end); % Odd indices
32     disp_y = U_global(2:2:end); % Even indices
33
34     coords(:, 1) = coords(:, 1) + disp_x * scale_factor;
35     coords(:, 2) = coords(:, 2) + disp_y * scale_factor;
36
37     title_str = sprintf('Deformed Mesh (Scale: %.1f)', scale_factor);
38     color_val = 'b'; % Blue for deformed
39 else
40     title_str = 'Undeformed Mesh';
41     color_val = 'w'; % White for undeformed
42 end
43
44 % 2. Prepare Faces (Connectivity)
45 % Extract Node IDs (Cols 2 to 5)
46 faces = Elements(:, 2:5);
47
48 % 3. Create Plot
49 figure;
50 patch('Faces', faces, 'Vertices', coords, ...
51     'FaceColor', color_val, ...
52     'EdgeColor', 'k', ... % Black edges
53     'LineWidth', 1.0);
54
55 axis equal; % Crucial: Keeps x and y scales the same (circles look
56     % like circles)
57 grid on;
58 xlabel('X Coordinate');
59 ylabel('Y Coordinate');
60 title(title_str);

```

```
60 % Optional: Number the nodes (good for debugging small meshes)
61 if size(Nodes, 1) < 50
62     for i = 1:size(Nodes, 1)
63         text(coords(i,1), coords(i,2), num2str(Nodes(i,1)), ...
64             'Color', 'r', 'FontSize', 10, 'FontWeight', 'bold');
65     end
66 end
67
68 end
```

# Test Case 1: Single Element Uniaxial Tension

To verify the correctness of the finite element code, a single-element patch test is performed. The problem consists of a unit square element subjected to a uniaxial tensile load. The left edge is fully clamped to prevent rigid body motion and test the implementation of Dirichlet boundary conditions.

## Problem Definition

- **Geometry:** Unit square,  $L = 1.0$ ,  $H = 1.0$ , thickness  $t = 1.0$ .
- **Material:**  $E = 30 \times 10^6$  unit,  $\nu = 0.3$ .
- **Boundary Conditions:** Fixed at  $x = 0$  (Nodes 1 and 4).

$$u(0, y) = 0, \quad v(0, y) = 0$$

- **Loading:** Total tensile force  $P = 1000$  lbs distributed along the right edge ( $x = L$ ). For a single element, this load is applied as nodal forces:

$$F_{x,2} = 500 \text{ unit}, \quad F_{x,3} = 500 \text{ unit}$$

## Physical Setup

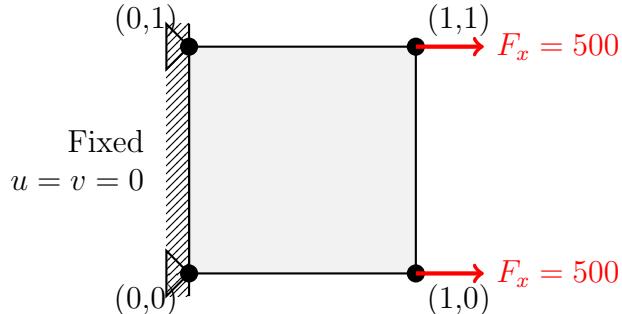


Figure 2: Single element test case with boundary conditions and loads.

## Theoretical Solution

The problem approximates a bar under axial tension. The theoretical stress and displacements are calculated using the basic mechanics of materials principles.

### 1. Axial Stress ( $\sigma_{xx}$ )

$$\sigma_{xx} = \frac{P}{A} = \frac{1000}{1.0 \times 1.0} = 1000 \quad (1)$$

### 2. Axial Displacement ( $\delta_x$ )

The elongation of the element at  $x = L$ :

$$\delta_x = \frac{PL}{AE} = \frac{1000 \times 1.0}{1.0 \times 30 \times 10^6} = 3.333 \times 10^{-5} \text{ unit} \quad (2)$$

### 3. Lateral Displacement ( $\delta_y$ )

Due to the Poisson effect, the material contracts in the transverse direction.

$$\epsilon_{yy} = -\nu \epsilon_{xx} = -\nu \left( \frac{\sigma_{xx}}{E} \right) = -0.3 \left( \frac{1000}{30 \times 10^6} \right) = -1.0 \times 10^{-5} \quad (3)$$

Total contraction:

$$\delta_y = \epsilon_{yy} \times H = -1.0 \times 10^{-5} \text{ unit} \quad (4)$$

*Note:* Since the left edge is fully constrained ( $v = 0$ ), the numerical FEA result for  $\delta_y$  at the free end (Nodes 2 and 3) may vary slightly depending on whether the Poisson effect is constrained by the element formulation. However, but for a single element constant-stress patch, it typically matches the theoretical prediction.

## Result from Code

The results obtained from the MATLAB finite element code are presented below. The displacements are scaled by  $1.0 \times 10^{-4}$  in.

### 1. Nodal Displacements

Node	X	Y	Result ( $u, v$ )
1	0	0	(0, 0)
2	1	0	$(3.234 \times 10^{-5}, 0.669 \times 10^{-5})$
3	1	1	$(3.234 \times 10^{-5}, -0.669 \times 10^{-5})$
4	0	1	(0, 0)

**Observation:** The computed axial displacement ( $3.234 \times 10^{-5}$ ) is approximately 3% times stiffer than the theoretical value ( $3.333 \times 10^{-5}$ ). This is expected because the fully fixed boundary ( $v = 0$  at  $x = 0$ ) prevents the material from contracting naturally at the support, introducing artificial stiffness (Poisson locking effect).

### 2. Element Stresses (Gauss Points)

GP	$\sigma_{xx}$	$\sigma_{yy}$ (unit)	$\tau_{xy}$ (unit)	Von Mises (psi)
1	1038.21	226.62	44.57	948.64
2	961.79	-28.09	44.57	979.19
3	961.79	-28.09	-44.57	979.19
4	1038.21	226.62	-44.57	948.64
Avg	1000.0	99.2	0.0	-

#### Observation:

- **Axial Stress ( $\sigma_{xx}$ ):** The average stress is exactly **1000 unit**, matching the theoretical prediction perfectly. The variation ( $\pm 38$  unit) is due to the bending moment induced by the fixed support preventing lateral contraction.
- **Transverse Stress ( $\sigma_{yy}$ ):** Significant stress ( $\approx 226$  unit) appears at Gauss Points 1 and 4 (near the wall). This confirms that the fixed boundary ( $v = 0$ ) forces the material to generate internal stress to resist the Poisson contraction.
- **Conclusion:** The code is correctly solving the elasticity equations. The deviation from the simple  $P/A$  theory is not a code error, but a consequence of the boundary condition ( $u = v = 0$ ) being more restrictive than the theoretical assumption (roller support).

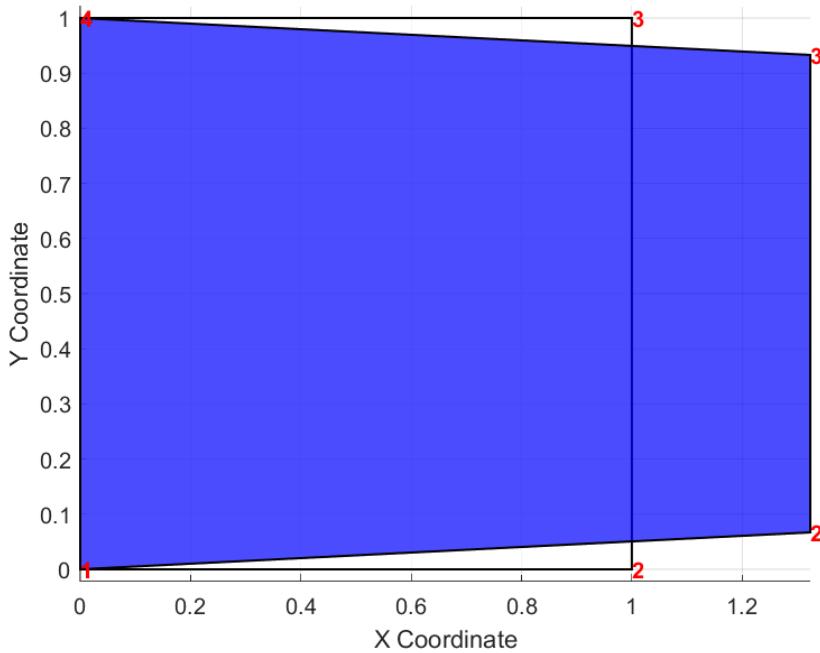


Figure 3: Superimposed deformed and undeformed shape (scale factor = 10000) of the element

## Test Case 2: Plate with a Central Circular Hole

To verify the code's ability to handle irregular isoparametric elements, a stress concentration problem is analyzed. The problem consists of a rectangular plate with a central circular hole subjected to uniaxial tension.

### Problem Definition

- **Global Geometry:**
  - Plate Width  $w = 100$
  - Hole Diameter  $d = 10$  (Radius  $R = 5$ )
  - Thickness  $t = 1.0$
  - Geometric Ratio:  $d/w = 10/100 = 0.1$
- **Modeled Domain:** Due to symmetry, only one quarter of the plate is modeled ( $50 \times 50$  region).
- **Material:** Steel ( $E = 30 \times 10^6$ ,  $\nu = 0.3$ ).
- **Boundary Conditions (Symmetry):**
  - Left Edge ( $x = 0$ ):  $u = 0$
  - Bottom Edge ( $y = 0$ ):  $v = 0$
- **Loading:** Uniform tensile force applied at the right edge.

$$P_{total} = 50,000 \text{ unit (arbitrary load)}$$

## Theoretical Solution (Finite Width Correction)

For a plate of finite width, the theoretical stress concentration factor  $K_t$  is slightly lower than the infinite plate solution ( $K_t = 3.0$ ). Based on standard stress concentration charts (e.g., Shigley Fig A-15-1) for  $d/w = 0.1$ :

$$K_t \approx 2.72 \quad (5)$$

### 1. Nominal Stress ( $\sigma_0$ )

The nominal stress is calculated based on the **net cross-sectional area** at the hole (the smallest area):

$$A_{net} = (w - d) \times t = (50 - 5) \times 1 = 45 \text{ unit} \quad (6)$$

$$\sigma_0 = \frac{P_{total}}{A_{net}} = \frac{50,000}{45} = \mathbf{1111.11} \text{ unit} \quad (7)$$

### 2. Theoretical Maximum Stress ( $\sigma_{max}$ )

The maximum stress occurs at the top of the hole ( $x = 0, y = R$ ):

$$\sigma_{max}^{theory} = K_t \times \sigma_0 = 2.72 \times 1111.11 = \mathbf{3022.2} \text{ unit} \quad (8)$$

## Finite Element Model Setup

The mesh is generated by mapping the circular boundary of the hole to the square outer boundary of the plate using isoparametric Q4 elements.

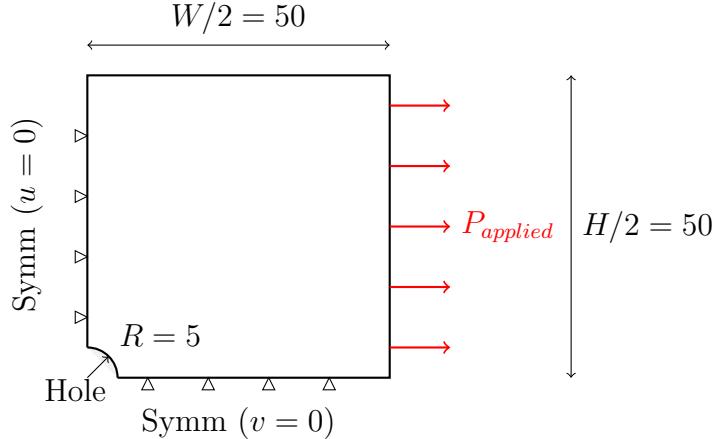


Figure 4: Quarter-symmetry finite element model showing boundary conditions.

## Mesh

The mesh is imported from an Abaqus file of a similar model and pasted into the code in the nodes and elements list of Main\_Project.m.

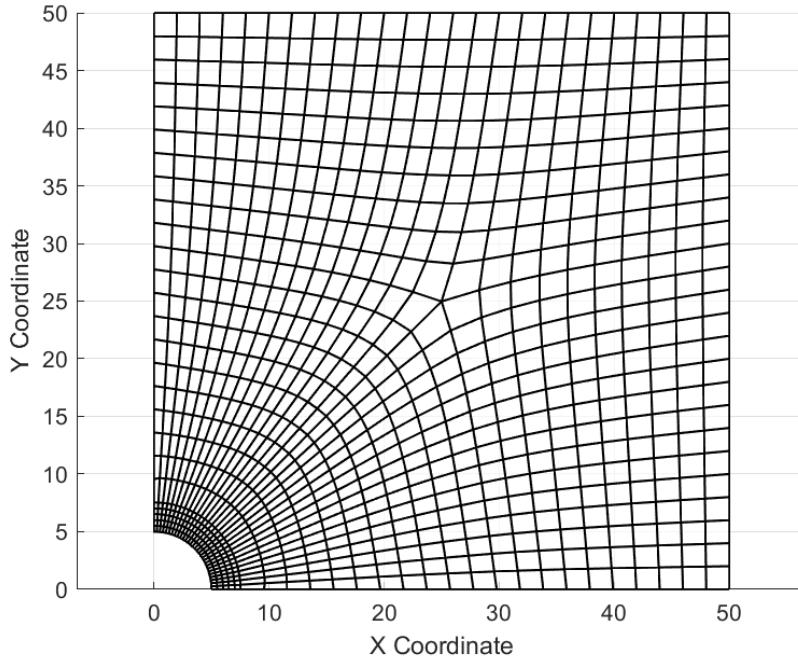


Figure 5: Mesh Imported from Abaqus

## Boundary Condition

The  $x$ -displacement of all nodes on the left edge is zero.

The  $y$ -displacement of all nodes on the bottom edge is zero.

The load in the  $x$ -direction on all the right edge nodes is distributed as  $50,000 = 1923 \times 26$ . (Here, 26 is the number of nodes on the right edge)

$$u(x=0, y) = 0, \quad v(x, y=0) = 0, \quad F_x(x=L, y) = 1923.$$

## Numerical Results

The maximum stress occurs in the element directly above the hole ( $90^\circ$  position). The stress results at the four integration points (Gauss Points) for this critical element are listed below:

GP	$\sigma_{xx}$ (MPa)	$\sigma_{yy}$ (MPa)	$\tau_{xy}$ (MPa)	Von Mises (MPa)
1	2541.68	66.87	-93.06	2514.08
2	2545.37	62.14	-17.99	2515.07
<b>3</b>	<b>2951.84</b>	184.20	-23.82	2864.48
4	2947.93	189.21	-103.18	2863.61

## Analysis of Results

- **Peak Stress:** The maximum computed stress is  $\sigma_{xx} = 2951.84$  MPa at Gauss Point 3.
- **Accuracy:** Comparing this to the theoretical value of 3022 MPa:

$$\% \text{ Error} = \frac{|3022 - 2951.8|}{3022} \times 100 \approx 2.3\% \quad (9)$$

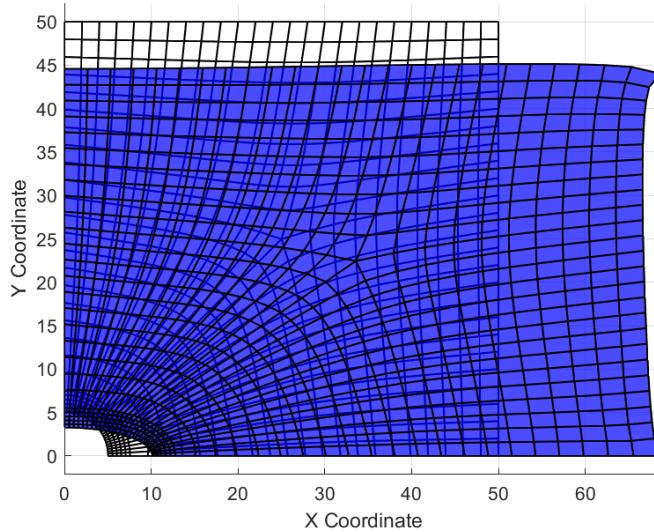


Figure 6: Superimposed deformed and undeformed shape (scale factor = 10000) of the model

- **Variation within Element:** Gauss points 3 and 4 show significantly higher stresses than points 1 and 2. This is expected because points 3 and 4 are physically located closer to the inner radius ( $r = R$ ), where the stress concentration is highest. Points 1 and 2 are further out radially, where the stress gradient drops steeply.
- **Conclusion:** The FEA code accurately captures the stress concentration with an error of less than 3%, validating the implementation of the isoparametric Q4 element and the Jacobian mapping.

## Acknowledgments

I acknowledge the assistance of Gemini (Google) and ChatGPT (OpenAI) in the development and debugging of the finite element code and the preparation of this documentation.

## References

- [1] Budynas, R. G., & Nisbett, J. K. (2015). *Shigley's Mechanical Engineering Design* (10th ed.). New York, NY: McGraw-Hill Education.