ME 471
FEM Project

Mohamed Elkamash
me21@illinois.edu

December 4, 2024

Contents

1 Code

Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Material Properties

In [2]:
```
elastic_modulus = 1e9
v = 0.3
```

Quadrature Points

In [3]:
```
quad_points = np.array([[1/np.sqrt(3), 1/np.sqrt(3)], [-1/np.sqrt(3), 1/np.sqrt(3)]
                        [-1/np.sqrt(3), -1/np.sqrt(3)], [1/np.sqrt(3), -1/np.sqrt(3
w = np.array([1,1,1,1])
```

Shape Functions

In [4]:
```
def evaluate_N(xi, eta):
    n = np.zeros(4)
    n[0] = (1+xi)*(1+eta)/4
    n[1] = (1-xi)*(1+eta)/4
    n[2] = (1-xi)*(1-eta)/4
    n[3] = (1+xi)*(1-eta)/4
    return n

def evaluate_dN_d_xi(xi, eta):
    v = np.zeros(4)
    v[0] = (1+eta)/4
    v[1] = (-1-eta)/4
    v[2] = (-1+eta)/4
    v[3] = (1-eta)/4
    return v

def evaluate_dN_d_eta(xi, eta):
    v = np.zeros(4)
    v[0] = (1+xi)/4
    v[1] = (1-xi)/4
    v[2] = (-1+xi)/4
    v[3] = (-1-xi)/4
    return v
```

In [5]:
```
#evaluate shape functions and derivatives at quadrature points once and save them i
n_quad = len(quad_points)
N = np.zeros([n_quad, 4])
```

```
dN_d_xi = np.zeros([n_quad, 4])
dN_d_eta = np.zeros([n_quad, 4])
for i in range(n_quad):
    xi = quad_points[i,0]
    eta = quad_points[i,1]
    N[i] = evaluate_N(xi,eta)
    dN_d_xi[i] = evaluate_dN_d_xi(xi,eta)
    dN_d_eta[i] = evaluate_dN_d_eta(xi,eta)
```

Node Class

In [6]:
```
class Node:
    def __init__(self, id, x, y):
        self.id = id
        self.x  = x
        self.y  = y

    def __repr__(self):
        return f"Node({self.id}, x={self.x}, y={self.y})"
```

Element Class

In [7]:
```
class Element:
    def __init__(self, id, nodes):
        self.id = id
        self.nodes = nodes

    def __repr__(self):
        node_ids = [node.id for node in self.nodes]
        return f"Element(id={self.id}, nodes={node_ids})"

    def coordinates(self):
        return np.array([(node.x, node.y) for node in self.nodes])

    def compute_E(self):
        E = np.zeros([3,3])
        E[0,0] = 1-v
        E[0,1] = v
        E[0,2] = 0
        E[1,0] = v
        E[1,1] = 1-v
        E[0,2] = 0
        E[2,0] = 0
        E[2,1] = 0
        E[2,2] = (1-2*v)/2
        E = E * elastic_modulus / ((1+v)*(1-2*v))
        return E

    def compute_J(self, i_quad):
        #define jacobian matrix at quadrature point i, j
        j = np.zeros([2,2])
```

```
        #get coordinates of the nodes
        coords = self.coordinates()
        x = coords[:,0]
        y = coords[:,1]
        #compute jacobian matrix elements
        j[0,0] = np.dot(x, dN_d_xi[i_quad])
        j[0,1] = np.dot(y, dN_d_xi[i_quad])
        j[1,0] = np.dot(x, dN_d_eta[i_quad])
        j[1,1] = np.dot(y, dN_d_eta[i_quad])
        return j

    def compute_B(self, i_quad, J_inv):
        dN = np.zeros([4,2]) #derivative matrix, size = n_shape * n_dims
        for i in range(4): #loop over shape functions
            mat = np.array([dN_d_xi[i_quad,i], dN_d_eta[i_quad,i]])
            dN[i] = np.matmul(J_inv, mat)
        B = np.zeros([3,8])
        B[0,0] = dN[0,0]
        B[0,2] = dN[1,0]
        B[0,4] = dN[2,0]
        B[0,6] = dN[3,0]
        B[1,1] = dN[0,1]
        B[1,3] = dN[1,1]
        B[1,5] = dN[2,1]
        B[1,7] = dN[3,1]
        B[2,0] = dN[0,1]
        B[2,1] = dN[0,0]
        B[2,2] = dN[1,1]
        B[2,3] = dN[1,0]
        B[2,4] = dN[2,1]
        B[2,5] = dN[2,0]
        B[2,6] = dN[3,1]
        B[2,7] = dN[3,0]
        return B

    def localStiffnessMatrix(self):
        k = 0
        E = self.compute_E()
        for i in range(n_quad):
            J = self.compute_J(i)
            J_inv = np.linalg.inv(J)
            det_J = np.linalg.det(J)
            B = self.compute_B(i, J_inv)
            BT = np.transpose(B)
            k += BT @ E @ B * det_J * w[i]
        return k

    def compute_stresses(self, D):
        stress = np.zeros([n_quad, 3])
        # Loop over the quadrature points
        for i in range(n_quad):
            J = self.compute_J(i)
            J_inv = np.linalg.inv(J)
            B = self.compute_B(i, J_inv)
            # Get the displacements for the element (use D with the corresponding D
            d = np.array([item for pair in zip([D[2 * (node.id - 1)] for node in se
```

```
                            for item in pair])
            # Calculate the strain vector
            strain = B @ d
            # Compute the stress vector
            E = self.compute_E()
            stress[i,:] = E @ strain
        return stress
```

Mesh class

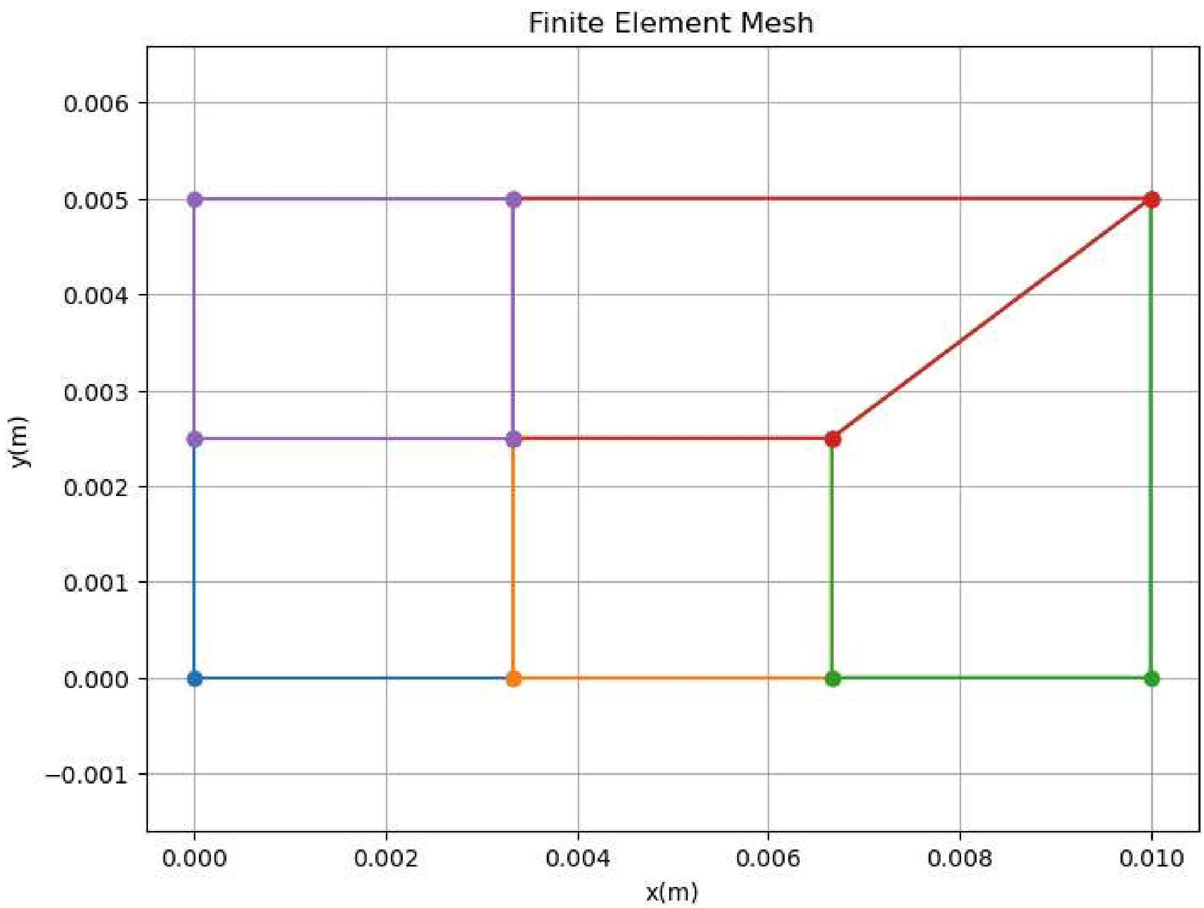
In [8]:
```
class Mesh:
    def read_node_list(self, file_path):
        nodes = []
        node_section = False  # Flag to check if we are in the *Node section
        with open(file_path, 'r') as file:
            for line in file:
                line = line.strip()
                # Check for the start of the *Node section
                if line.startswith('*Node'):
                    node_section = True
                    continue  # Skip the line that contains *Node header
                # Start of element section, stop reading node
                if node_section and line.startswith('*Element'):
                    node_section = False
                    break
                # Skip the line if it is not a node data line
                if not line or line.startswith('*') or not line[0].isdigit():
                    continue
                # If we are in the *Node section, process the node data
                node_data = line.split(',')
                node_id = int(node_data[0].strip())
                x = float(node_data[1].strip())
                y = float(node_data[2].strip())
                # Create Node object and append to the list
                node = Node(node_id, x, y)
                nodes.append(node)
        return nodes

    def read_element_list(self, file_path):
        elements = []
        element_section = False  # Flag to check if we are in the *Element section
        with open(file_path, 'r') as file:
            for line in file:
                line = line.strip()
                # Check for the start of the *Element section
                if line.startswith('*Element'):
                    element_section = True
                    continue  # Skip the line that contains *Element header
                # Stop reading elements
                if element_section and line.startswith('*Nset'):
                    element_section = False
                    break
                # Skip the line if it is not an element data line
                if not line or line.startswith('*') or not line[0].isdigit() or not
```

```
                    continue
                # If we are in the *Element section, process the element data
                element_data = line.split(',')
                element_id = int(element_data[0].strip())
                nodes = []
                for i in range(1,5):
                    node_id = int(element_data[i].strip())
                    node_index = node_id - 1 #zero indexing shift
                    nodes.append(self.nodes[node_index])
                # Create Element object and append to the list
                element = Element(element_id, nodes)
                elements.append(element)
        return elements

    def __init__(self, file_path):
        self.nodes = self.read_node_list(file_path)
        self.elements = self.read_element_list(file_path)

    def plot_mesh(mesh):
        plt.figure(figsize=(8, 6))
        # Plot each element
        for element in mesh.elements:
            # Get the coordinates of the nodes
            coords = np.array([(node.x, node.y) for node in element.nodes])
            # Close the polygon by appending the first node again
            coords = np.vstack([coords, coords[0]])
            plt.plot(coords[:, 0], coords[:, 1], marker='o')
        # Set labels and title
        plt.title('Finite Element Mesh')
        plt.xlabel('x(m)')
        plt.ylabel('y(m)')
        plt.axis('equal')
        plt.grid(True)
        plt.show()
```

Mesh Generation

In [9]:

```
file_path = 'input.txt'
mesh = Mesh(file_path)
mesh.plot_mesh()
```



Assembly

In [10]:

```
n_nodes = len(mesh.nodes)
#Global stiffness matrix and global load vector
K_global = np.zeros((2*n_nodes, 2*n_nodes))
N_local_nodes = 4
#loop over all elements
for element in mesh.elements:
    k_local = element.localStiffnessMatrix()
    local_indices = [node.id - 1 for node in element.nodes] #adjust for zero indexi
    local_indices = np.multiply(2,local_indices)
    local_indices = np.concatenate([np.arange(x, x+2) for x in local_indices]) #exp
    for i in range(2*N_local_nodes):
        for j in range(2*N_local_nodes):
            K_global[local_indices[i], local_indices[j]] += k_local[i,j]
```

Load

In [11]:

```
R_global = np.zeros(2*n_nodes)
R_global[6] = 18
R_global[8] = 42
R_global[9] = 333.0
```

```
R_global[17] = 166.5
R_global[19] = 499.5
```

Boundary Conditions

In [12]:
```
i_zero_displacement = [0, 1, 3, 5, 7, 14, 16]
ids = np.arange(0,20,1)
mask = np.setdiff1d(ids, i_zero_displacement)
```

Solve

In [13]:
```
K_reduced = np.delete(K_global, i_zero_displacement, 0)
K_reduced = np.delete(K_reduced, i_zero_displacement, 1)
R_reduced = np.delete(R_global, i_zero_displacement)
D_reduced = np.linalg.solve(K_reduced,R_reduced)
D = np.zeros(2*n_nodes) #full displacement vector
D[mask] = D_reduced
R = K_global @ D - R_global #reaction forces
R[abs(R)<1.0e-10] = 0
thickness = 0.005
D = D/thickness
```

Output

In [14]:
```
#Displacements
print(f"{'Node ID':<10} {'u_magnitude (m)':<20} {'u_x (m)':<15} {'u_y (m)':<15}")
print("-" * 60)
for i in range(n_nodes):
    j = 2*i
    k = 2*i+1
    u_x = f"{D[j]:.3g}"
    u_y = f"{D[k]:.3g}"
    u_m = np.sqrt(D[j]**2 + D[k]**2)
    u_m_str = f"{u_m:.3g}"
    print(f"{i+1:<10} {u_m_str:<20} {u_x:<15} {u_y:<15}")
print('-' * 60)
```

```
Node ID    u_magnitude (m)      u_x (m)         u_y (m)
------------------------------------------------------------
1          0                    0               0
2          1.93e-05             -1.93e-05       0
3          3.96e-05             -3.96e-05       0
4          6.13e-05             -6.13e-05       0
5          9.91e-05             -5.09e-05       8.5e-05
6          5.77e-05             -3.78e-05       4.37e-05
7          4.78e-05             -1.88e-05       4.39e-05
8          4.34e-05             0               4.34e-05
9          8.67e-05             0               8.67e-05
10         8.9e-05              -1.85e-05       8.71e-05
------------------------------------------------------------
```

In [15]:
```python
#Reaction forces
print(f"{'Node ID':<10} {'R_Magnitude (N)':<20} {'R_x (N)':<15} {'R_y (N)':<15}")
print("-" * 60)
for i in range(n_nodes):
    j = 2*i
    k = 2*i+1
    R_x = f"{R[j]:.3g}"
    R_y = f"{R[k]:.3g}"
    R_m = np.sqrt(R[j]**2 + R[k]**2)
    R_m_str = f"{R_m:.3g}"
    print(f"{i+1:<10} {R_m_str:<20} {R_x:<15} {R_y:<15}")
print("-" * 60)
```

```
Node ID    R_Magnitude (N)      R_x (N)         R_y (N)
------------------------------------------------------------
1          169                  -15.1           -168
2          338                  0               -338
3          337                  0               -337
4          156                  0               -156
5          0                    0               0
6          0                    0               0
7          0                    0               0
8          30.2                 -30.2           0
9          14.7                 -14.7           0
10         0                    0               0
------------------------------------------------------------
```

In [16]:
```python
#Stresses
print(f"{'Element ID':<15} {'Quad Point':<15} {'S11 (Pa)':<14} {'S22 (Pa)':<15}")
print("-" * 60)
# Loop over each element
for element in mesh.elements:
    stress = element.compute_stresses(D)
    # Loop over each quadrature point
    for quad_id in range(stress.shape[0]):
        S11 = stress[quad_id, 0]
        S22 = stress[quad_id, 1]
        print(f"{element.id:<15} {quad_id+1:<15} {S11:.2e} {'':<5} {S22:.2e}")
print("-" * 60)
```

Element ID	Quad Point	S11 (Pa)	S22 (Pa)
1	1	2.47e+06	2.03e+07
1	2	2.39e+06	2.02e+07
1	3	2.29e+06	2.01e+07
1	4	2.36e+06	2.03e+07
2	1	2.32e+06	2.02e+07
2	2	2.35e+06	2.03e+07
2	3	2.04e+06	2.02e+07
2	4	2.00e+06	2.01e+07
3	1	2.27e+06	1.97e+07
3	2	3.00e+06	2.03e+07
3	3	1.67e+06	1.97e+07
3	4	1.37e+06	1.93e+07
4	1	3.13e+06	2.00e+07
4	2	3.24e+06	2.03e+07
4	3	2.62e+06	2.00e+07
4	4	2.45e+06	1.96e+07
5	1	2.44e+06	2.00e+07
5	2	2.47e+06	2.01e+07
5	3	2.40e+06	2.01e+07
5	4	2.37e+06	2.00e+07

2 Abaqus

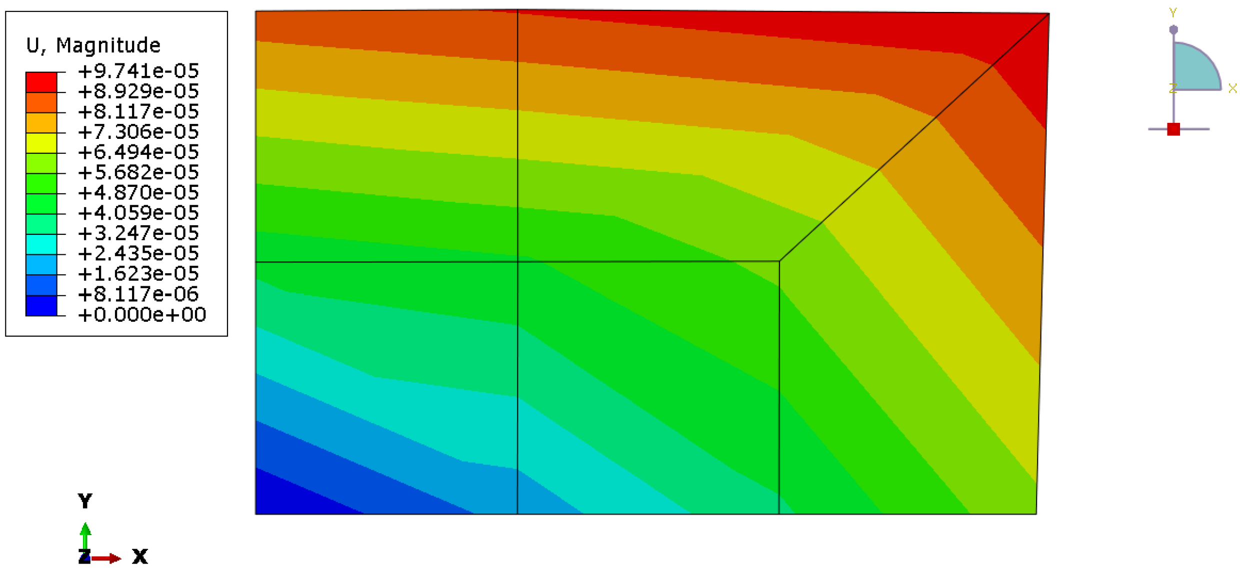


Figure 1: Magnitude of the displacements at the nodes in m.

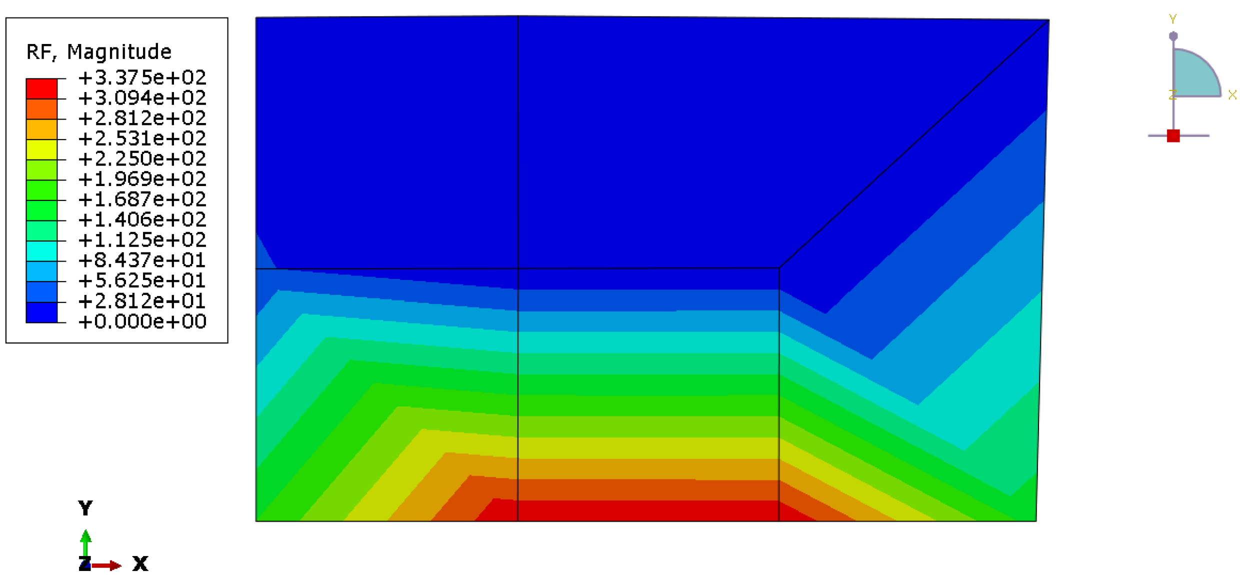


Figure 2: Magnitude of the reaction forces at the nodes in N.

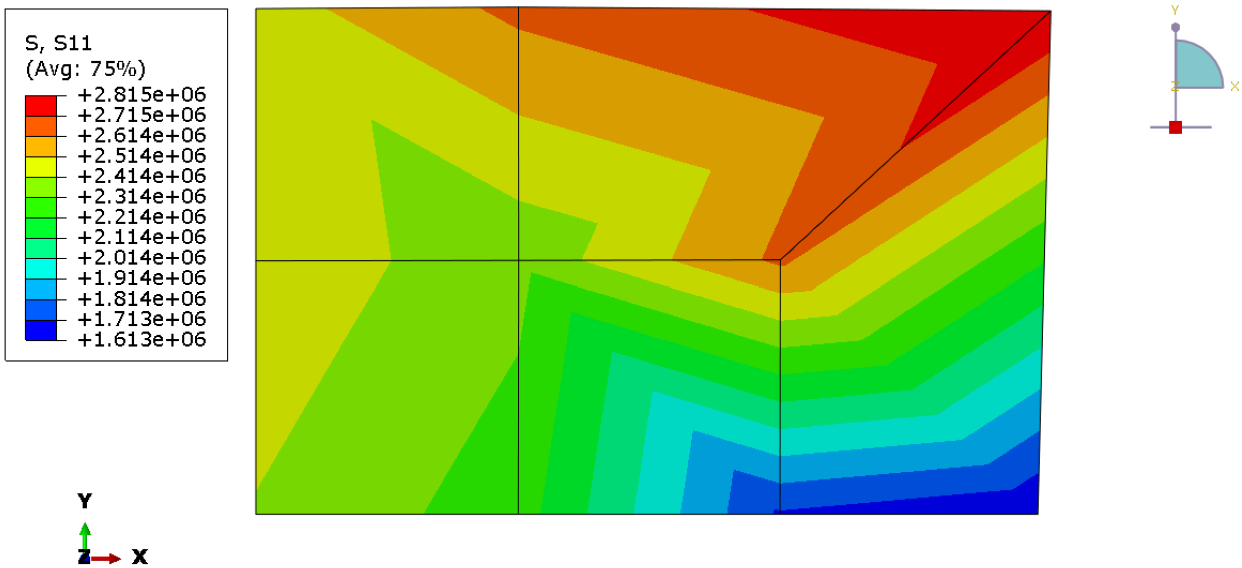


Figure 3: S11 stress at the integration points in MPa.

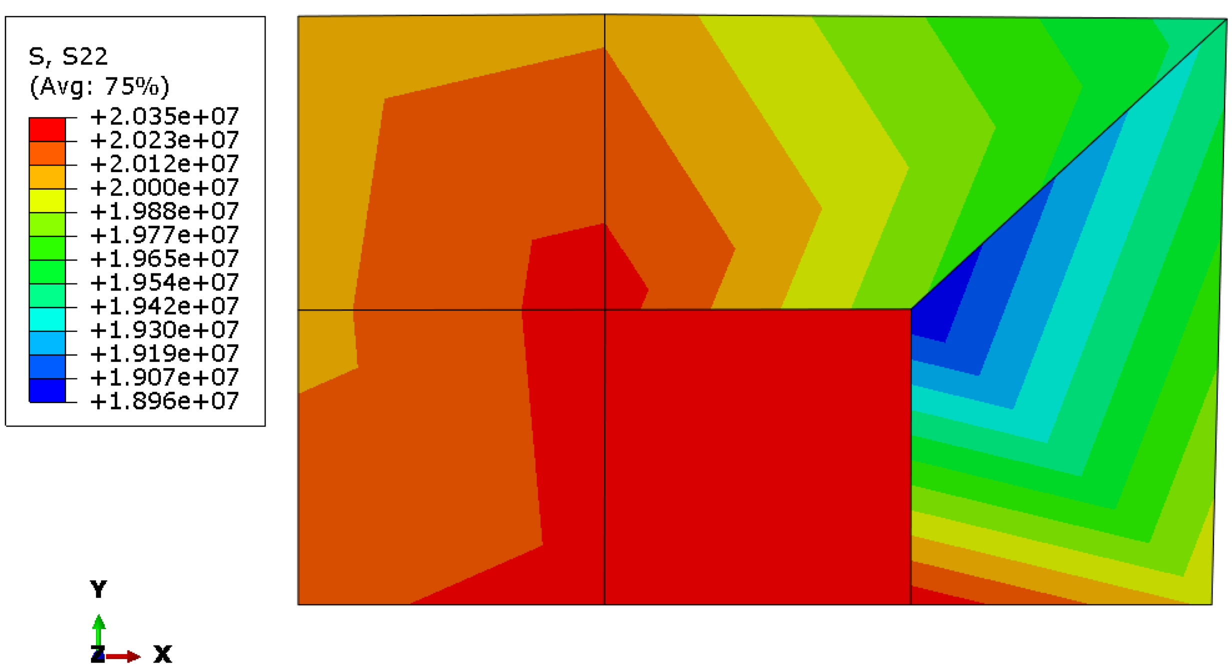


Figure 4: S22 stress at the integration points in MPa.

Node Label	U.Magnitude @Loc 1	U.U1 @Loc 1	U.U2 @Loc 1
1	0.	15.4068E-36	168.556E-36
2	19.2431E-06	-19.2431E-06	337.496E-36
3	39.0383E-06	-39.0383E-06	336.862E-36
4	62.8584E-06	-62.8584E-06	156.087E-36
5	97.4070E-06	-49.3493E-06	83.9807E-06
6	58.8104E-06	-38.6489E-06	44.3275E-06
7	47.6924E-06	-18.6977E-06	43.8744E-06
8	43.4134E-06	30.5210E-36	43.4134E-06
9	86.2559E-06	14.0723E-36	86.2559E-06
10	89.9377E-06	-19.0325E-06	87.9009E-06
Minimum	0.	-62.8584E-06	156.087E-36
At Node	1	4	4
Maximum	97.4070E-06	30.5210E-36	87.9009E-06
At Node	5	8	10
Total	544.657E-06	-246.868E-06	389.753E-06

Figure 5: Magnitude of the displacements at the nodes in m.

Node Label	RF.Magnitude @Loc 1	RF.RF1 @Loc 1	RF.RF2 @Loc 1
1	169.258	-15.4068	-168.556
2	337.496	0.	-337.496
3	336.862	0.	-336.862
4	156.087	0.	-156.087
5	0.	0.	0.
6	0.	0.	0.
7	0.	0.	0.
8	30.5210	-30.5210	0.
9	14.0723	-14.0723	0.
10	0.	0.	0.
Minimum	0.	-30.5210	-337.496
At Node	10	8	2
Maximum	337.496	0.	0.
At Node	2	10	10
Total	1.04430E+03	-60.	-999.

Figure 6: Magnitude of the reaction forces at the nodes in N.

Element Label	Int Pt	S.S11 @Loc 1	S.S22 @Loc 1
1	1	2.40515E+06	20.2116E+06
1	2	2.36420E+06	20.2526E+06
1	3	2.44153E+06	20.1753E+06
1	4	2.40058E+06	20.2162E+06
2	1	2.16873E+06	20.2783E+06
2	2	2.12848E+06	20.3186E+06
2	3	2.15833E+06	20.2887E+06
2	4	2.11808E+06	20.3290E+06
3	1	1.60813E+06	19.9859E+06
3	2	1.71696E+06	19.8770E+06
3	3	2.30893E+06	19.2851E+06
3	4	2.19155E+06	19.4024E+06
4	1	3.21409E+06	19.7383E+06
4	2	2.97442E+06	19.9780E+06
4	3	3.07512E+06	19.8773E+06
4	4	2.72122E+06	20.2312E+06
5	1	2.35572E+06	20.1609E+06
5	2	2.33340E+06	20.1832E+06
5	3	2.46088E+06	20.0557E+06
5	4	2.43855E+06	20.0780E+06
Minimum		1.60813E+06	19.2851E+06
At Element		3	3
Int Pt		1	3
Maximum		3.21409E+06	20.3290E+06
At Element		4	2
Int Pt		1	4
Total		47.5840E+06	400.923E+06

Figure 7: S11 and S22 stresses at the integration points in MPa.

3 Discussion

Table 1 shows a comparison between the results of the code and Abaqus. The maximum relative error in the displacements computed by the code was found to be 2.5% and the maximum relative error in the reaction forces is 4.3 %. The errors can be attributed to the difference in the way that Abaqus and the code formulate the local stiffness matrix, the method used in solving the linear system and the truncation errors due to float computations. The stresses from the code and Abaqus are reported in section 1 and section 2 respectively.

Table 1: Comparison of Results between Code and Abaqus

Node id	U_code (m)	U_Abaqus (m)	U_rel_error	RF_code (N)	RF_Abaqus (N)	RF_rel_error
1	0.00E+00	0.00E+00	0.0E+00	169	169	0.0E+00
2	1.93E-05	1.92E-05	3.0E-03	338	337	3.0E-03
3	3.96E-05	3.90E-05	1.4E-02	337	337	0.0E+00
4	6.13E-05	6.29E-05	2.5E-02	156	156	0.0E+00
5	9.91E-05	9.74E-05	1.7E-02	0	0	0.0E+00
6	5.77E-05	5.88E-05	1.9E-02	0	0	0.0E+00
7	4.78E-05	4.77E-05	2.3E-03	0	0	0.0E+00
8	4.34E-05	4.34E-05	3.1E-04	30.2	30.5	9.8E-03
9	8.67E-05	8.63E-05	5.1E-03	14.7	14.1	4.3E-02
10	8.90E-05	8.99E-05	1.0E-02	0	0	0.0E+00