ME5204 Finite Element Analysis

Project 2D Plane Stress Problems

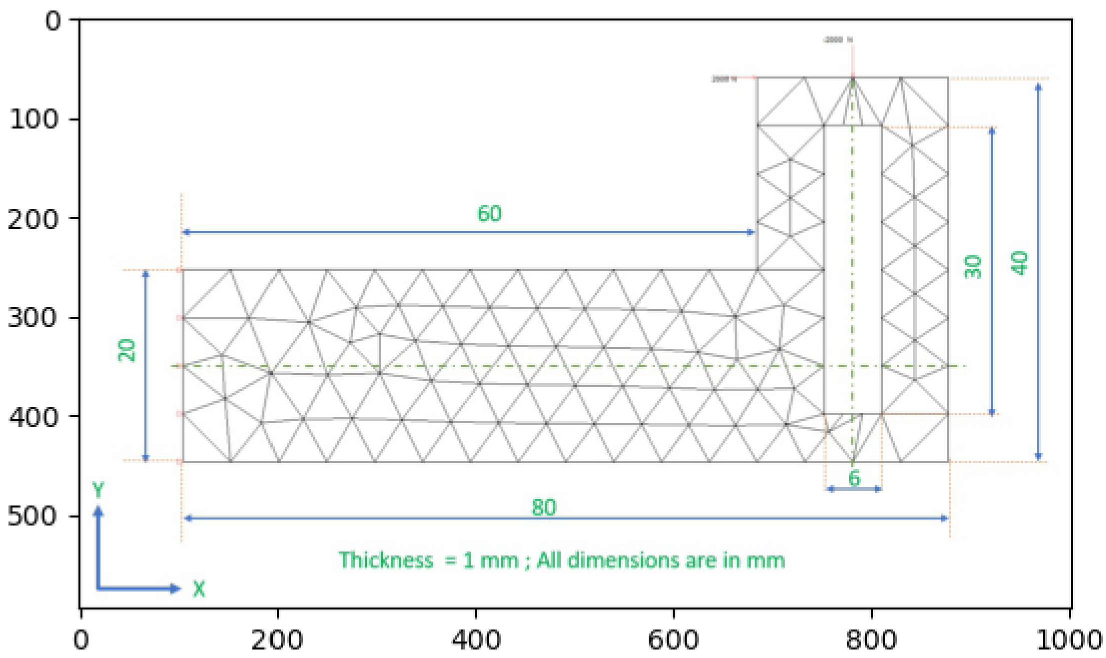
SENTHILKUMAR R (ME22M016)

In [1]:
```python
import math
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Problem Definition

In [2]:
```python
img = Image.open('problem_definition.png')
plt.imshow(img)
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x1fb8ea2c610>
```



Analysis Input

In [3]:
```python
Num_Nodes, Num_Elems, Num_Mats, Prob_Type, Thickness = np.loadtxt("./input.txt")

Num_Nodes, Num_Elems, Num_Mats, Prob_Type = int(Num_Nodes), int(Num_Elems), int(N
```

Nodal Co-ordinate and Array

In [4]:
```python
COORD = np.loadtxt("./COORD.txt").astype(np.float32)
```

In [5]:
```python
NCA = np.loadtxt("./NCA.txt").astype(np.int32)
#print(NCA)
```

Material Property

In [6]:
```
MAT = np.loadtxt("./MAT.txt").astype(np.float32)
#print(MAT)
```

Equivalent Nodal Forces

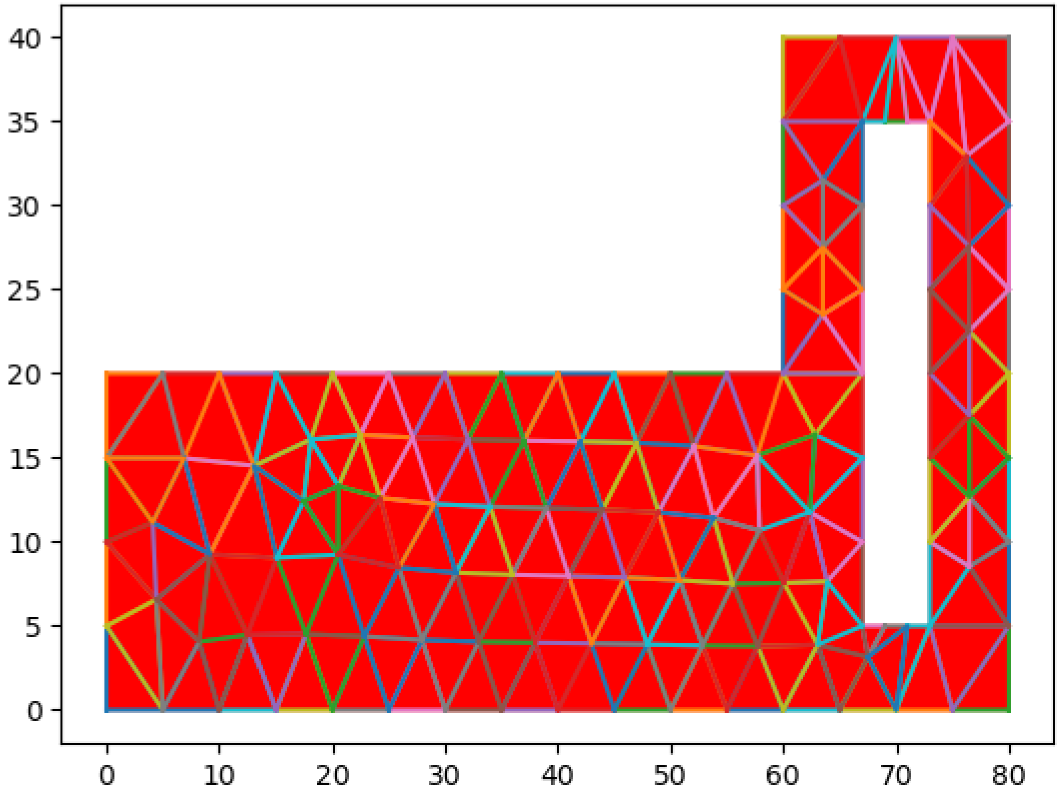
In [7]:
```
LOAD_BC = np.loadtxt("./LOAD_BC.txt").astype(np.float32)
```

Boundary Conditions

In [8]:
```
DISP_BC = np.loadtxt("./DISP_BC.txt").astype(np.float32)
```

Input Verification and Plot

In [9]:
```
X = np.zeros([Num_Elems, 4])
Y = np.zeros([Num_Elems, 4])
CGX = np.zeros([Num_Elems, 1])
CGY = np.zeros([Num_Elems, 1])

for elem in range(1, Num_Elems+1,1):
    N1 = NCA[elem,3]
    N2 = NCA[elem,2]
    N3 = NCA[elem,1]
    X1N1 = COORD[N1,1]
    X2N1 = COORD[N1,2]
    X1N2 = COORD[N2,1]
    X2N2 = COORD[N2,2]
    X1N3 = COORD[N3,1]
    X2N3 = COORD[N3,2]
    Mat_Num = NCA[elem,4]
    X = [X1N1,X1N2,X1N3,X1N1]
    Y = [X2N1,X2N2,X2N3,X2N1]
    CGX[elem - 1] = (X1N1+X1N2+X1N3)/3.0
    CGY[elem - 1] = (X2N1+X2N2+X2N3)/3.0
    plt.plot(X, Y)
#     plt.scatter(X, Y)
    #plt.text(CGX[elem-1],CGY[elem-1],str(elem),bbox = dict(facecolor = 'white',
    match NCA[elem , 4]:
        case 1:
            plt.fill(X, Y, color = 'red')
        case 2:
            plt.fill(X, Y, color = 'green')
```



Element Stiffness matrix and Assembly of Global Stiffness matrix

In [10]:
```
DOF_PN = 2
Total_DOF = Num_Nodes*DOF_PN
GSTIFF = np.zeros((Total_DOF,Total_DOF))
F = np.zeros((Total_DOF,1))

for elem in range(1, Num_Elems+1,1):
    N1 = NCA[elem,3]
    N2 = NCA[elem,2]
    N3 = NCA[elem,1]
    X1N1 = COORD[N1,1]
    X2N1 = COORD[N1,2]
    X1N2 = COORD[N2,1]
    X2N2 = COORD[N2,2]
    X1N3 = COORD[N3,1]
    X2N3 = COORD[N3,2]
    Two_Delta_matrix= np.array([[1, X1N1, X2N1],
                                [1, X1N2, X2N2],
                                [1, X1N3, X2N3]])
    Two_Delta = np.linalg.det(Two_Delta_matrix)
    Num_Nodes_PE = 3
    B = np.zeros((Num_Nodes_PE,Num_Nodes_PE*DOF_PN))
    B1 = (X2N2 - X2N3)
    B2 = (X2N3 - X2N1)
    B3 = (X2N1 - X2N2)
    G1 = (X1N3 - X1N2)
    G2 = (X1N1 - X1N3)
    G3 = (X1N2 - X1N1)
    B[0,0] = B1/Two_Delta
    B[0,2] = B2/Two_Delta
    B[0,4] = B3/Two_Delta
    B[1,1] = G1/Two_Delta
    B[1,3] = G2/Two_Delta
    B[1,5] = G3/Two_Delta
    B[2,0] = G1/Two_Delta
    B[2,1] = B1/Two_Delta
    B[2,2] = G2/Two_Delta
    B[2,3] = B2/Two_Delta
```

```
    B[2,4] = G3/Two_Delta
    B[2,5] = B3/Two_Delta
    Two_Delta_2 = np.linalg.det(Two_Delta_matrix)
    Mat_Num = NCA[elem,4]
    match Mat_Num:
        case 1:
            E = MAT[Mat_Num,1]
            PR = MAT[Mat_Num,2]
        case 2:
            E = MAT[Mat_Num,1]
            PR = MAT[Mat_Num,2]
    D = np.zeros((Num_Nodes_PE,Num_Nodes_PE))
    match Prob_Type:
        case 21:
            CONST = E/(1-PR**2)
            D[0,0] = 1*CONST
            D[0,1] = PR*CONST
            D[0,2] = 0*CONST
            D[1,0] = PR*CONST
            D[1,1] = 1*CONST
            D[1,2] = 0*CONST
            D[2,0] = 0*CONST
            D[2,1] = 0*CONST
            D[2,2] = 0.5*(1-PR)*CONST
        case 22:
            CONST = E/((1+PR)*(1-2*PR))
            D[0,0] = (1-PR)*CONST
            D[0,1] = PR*CONST
            D[0,2] = 0*CONST
            D[1,0] = PR*CONST
            D[1,1] = (1-PR)*CONST
            D[1,2] = 0*CONST
            D[2,0] = 0*CONST
            D[2,1] = 0*CONST
            D[2,2] = 0.5*(1-2*PR)*CONST
    ESTIFF = B.transpose()@D@B*Thickness*0.5*Two_Delta
    CN = [2*N1-2, 2*N1-1, 2*N2-2, 2*N2-1,2*N3-2, 2*N3-1]
    CN_IDX = np.array(6*CN).reshape(6,6)
    RO_IDX = CN_IDX.transpose()
    GSTIFF[RO_IDX,CN_IDX] = GSTIFF[RO_IDX,CN_IDX]+ESTIFF
```

Force Matrix

In [11]:
```
Num_load = 2
F = np.zeros((Total_DOF,1))

for i in range(1, Num_load+1,1):
    LOAD_TYPE = int(LOAD_BC[i,2])
    match LOAD_TYPE:
        case 1:
            N = int(LOAD_BC[i,1])
            F[2*N-2,0] = F[2*N-2,0]+ LOAD_BC[i,3]
        case 2:
            N = int(LOAD_BC[i,1])
            F[2*N-1,0] = F[2*N-1,0]+ LOAD_BC[i,4]
        case 12:
            N = int(LOAD_BC[i,1])
            F[2*N-2,0] = F[2*N-2,0]+ LOAD_BC[i,3]
            F[2*N-1,0] = F[2*N-1,0]+ LOAD_BC[i,4]
```

Solve [K] {u} = {F} and find Displacement {u}

In [12]:
```
GSTIFFCOPY = GSTIFF.copy()
Num_Disp_BC = 5
```

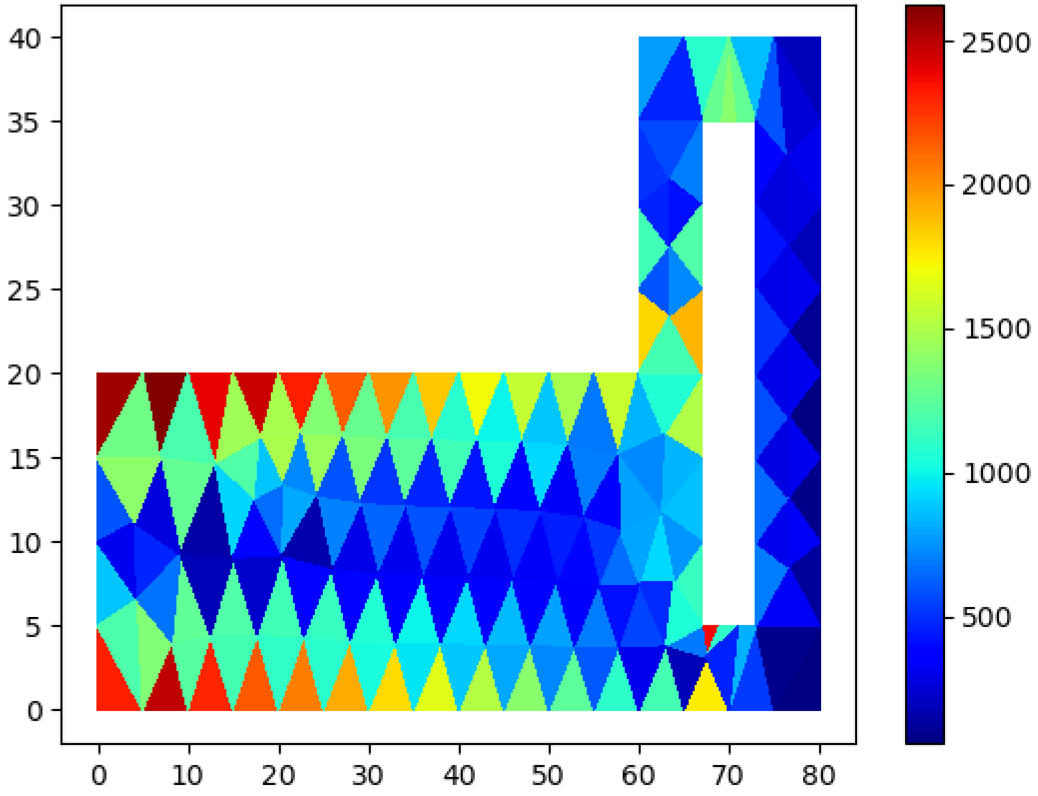
```
for i in range(1,Num_Disp_BC+1,1):
    DISP_TYPE = int(DISP_BC[i,2])
    match DISP_TYPE:
        case 1:
            N = int(DISP_BC[i,1])
            F[2*N-2,0] = F[2*N-2,0]+ DISP_BC[i,3]*10**32
            GSTIFFCOPY[2*N-2,2*N-2] = GSTIFFCOPY[2*N-2,2*N-2]+10**32
        case 2:
            N = int(DISP_BC[i,1])
            F[2*N-1,0] = F[2*N-1,0]+ DISP_BC[i,4]*10**32
            GSTIFFCOPY[2*N-1,2*N-1] = GSTIFFCOPY[2*N-1,2*N-1]+10**32
        case 12:
            N = int(DISP_BC[i,1])
            F[2*N-2,0] = F[2*N-2,0]+ DISP_BC[i,3]*10**32
            GSTIFFCOPY[2*N-2,2*N-2] = GSTIFFCOPY[2*N-2,2*N-2]+10**32
            F[2*N-1,0] = F[2*N-1,0]+ DISP_BC[i,4]*10**32
            GSTIFFCOPY[2*N-1,2*N-1] = GSTIFFCOPY[2*N-1,2*N-1]+10**32
```

In [13]:
```
DISP = np.linalg.solve(GSTIFFCOPY,F)
DISP_XY = DISP.reshape(-1, 2)
DISP_XY
np.savetxt('./DISPOUT',DISP_XY)
```

Equivalent Stress (Von-Mises Stress) Plot

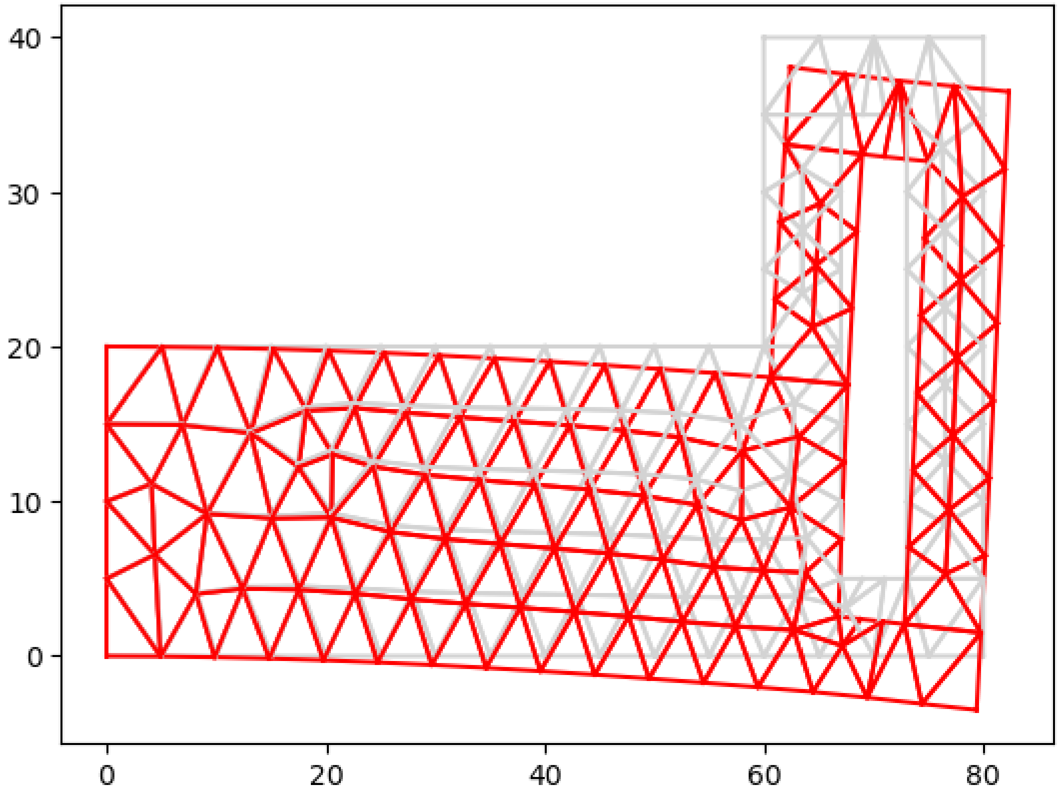
In [14]:
```
EQ_STRESS = np.zeros(Num_Elems+1)
for elem in range(1, Num_Elems+1,1):
    N1 = NCA[elem,3]
    N2 = NCA[elem,2]
    N3 = NCA[elem,1]
    X1N1 = COORD[N1,1]
    X2N1 = COORD[N1,2]
    X1N2 = COORD[N2,1]
    X2N2 = COORD[N2,2]
    X1N3 = COORD[N3,1]
    X2N3 = COORD[N3,2]
    Two_Delta_matrix= np.array([[1, X1N1, X2N1],
                               [1, X1N2, X2N2],
                               [1, X1N3, X2N3]])
    Two_Delta = np.linalg.det(Two_Delta_matrix)
    Num_Nodes_PE = 3
    B = np.zeros((Num_Nodes_PE,Num_Nodes_PE*DOF_PN))
    B1 = (X2N2 - X2N3)
    B2 = (X2N3 - X2N1)
    B3 = (X2N1 - X2N2)
    G1 = (X1N3 - X1N2)
    G2 = (X1N1 - X1N3)
    G3 = (X1N2 - X1N1)
    B[0,0] = B1/Two_Delta
    B[0,2] = B2/Two_Delta
    B[0,4] = B3/Two_Delta
    B[1,1] = G1/Two_Delta
    B[1,3] = G2/Two_Delta
    B[1,5] = G3/Two_Delta
    B[2,0] = G1/Two_Delta
    B[2,1] = B1/Two_Delta
    B[2,2] = G2/Two_Delta
    B[2,3] = B2/Two_Delta
    B[2,4] = G3/Two_Delta
    B[2,5] = B3/Two_Delta
    Two_Delta_2 = np.linalg.det(Two_Delta_matrix)
    Mat_Num = NCA[elem,4]
    match Mat_Num:
        case 1:
            E = MAT[Mat_Num,1]
```

```
            PR = MAT[Mat_Num,2]
        case 2:
            E = MAT[Mat_Num,1]
            PR = MAT[Mat_Num,2]
    D = np.zeros((Num_Nodes_PE,Num_Nodes_PE))
    match Prob_Type:
        case 21:
            CONST = E/(1-PR**2)
            D[0,0] = 1*CONST
            D[0,1] = PR*CONST
            D[0,2] = 0*CONST
            D[1,0] = PR*CONST
            D[1,1] = 1*CONST
            D[1,2] = 0*CONST
            D[2,0] = 0*CONST
            D[2,1] = 0*CONST
            D[2,2] = 0.5*(1-PR)*CONST
        case 22:
            CONST = E/((1+PR)*(1-2*PR))
            D[0,0] = (1-PR)*CONST
            D[0,1] = PR*CONST
            D[0,2] = 0*CONST
            D[1,0] = PR*CONST
            D[1,1] = (1-PR)*CONST
            D[1,2] = 0*CONST
            D[2,0] = 0*CONST
            D[2,1] = 0*CONST
            D[2,2] = 0.5*(1-2*PR)*CONST
    CN = [2*N1-2, 2*N1-1, 2*N2-2, 2*N2-1,2*N3-2, 2*N3-1]
    u = DISP[CN]
    STRAIN = B@u
    STRESS = D@STRAIN
    STRESS = STRESS.reshape(-1,3)
    STRESS_XX = STRESS[:,0]
    STRESS_YY = STRESS[:,1]
    STRESS_XY = STRESS[:,2]
    V = np.sqrt((STRESS_XX**2-(STRESS_XX*STRESS_YY))+(STRESS_YY**2)+(3*(STRESS_X

    EQ_STRESS[elem] = V

XCOORD = COORD[1:, 1]
YCOORD = COORD[1:, 2]
ELEM_CON = NCA[1:,1:-1]-1
VonMises = EQ_STRESS[1:]
plt.tripcolor(XCOORD, YCOORD, ELEM_CON, VonMises, cmap='jet')
plt.colorbar()
plt.show()
```



Deformed shape Vs Un-deformed Shape Plot

In [15]:
```
COORD_NEW = COORD[1:, [1,2]] + DISP_XY
for elem in range(1, Num_Elems+1,1):
    N1 = NCA[elem,3]
    N2 = NCA[elem,2]
    N3 = NCA[elem,1]

    X1N1 = COORD[N1,1]
    X2N1 = COORD[N1,2]
    X1N2 = COORD[N2,1]
    X2N2 = COORD[N2,2]
    X1N3 = COORD[N3,1]
    X2N3 = COORD[N3,2]

    X = [X1N1,X1N2,X1N3,X1N1]
    Y = [X2N1,X2N2,X2N3,X2N1]

    plt.plot(X, Y,  color='lightgrey')

    X1N1_NEW = COORD_NEW[N1-1,0]
    X2N1_NEW = COORD_NEW[N1-1,1]
    X1N2_NEW = COORD_NEW[N2-1,0]
    X2N2_NEW = COORD_NEW[N2-1,1]
    X1N3_NEW = COORD_NEW[N3-1,0]
    X2N3_NEW = COORD_NEW[N3-1,1]
    Mat_Num = NCA[elem, 4]
    X = [X1N1_NEW,X1N2_NEW,X1N3_NEW,X1N1_NEW]
    Y = [X2N1_NEW,X2N2_NEW,X2N3_NEW,X2N1_NEW]

    plt.plot(X, Y,  color='red')
```

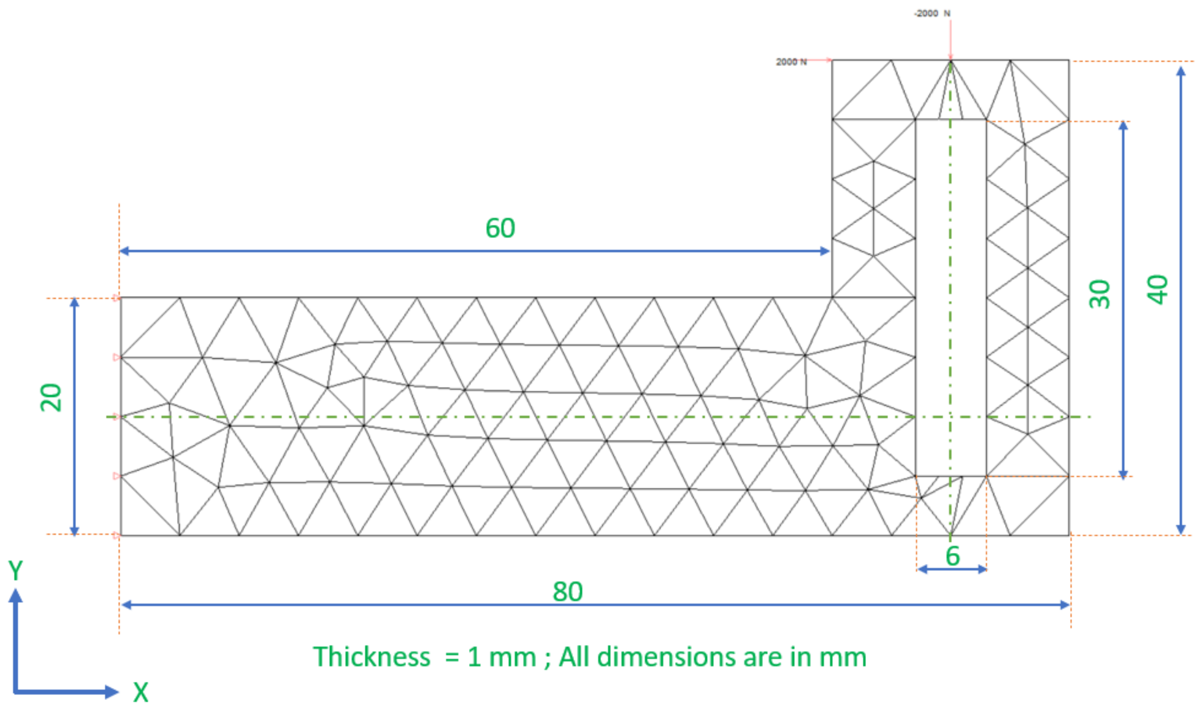


In [ ]:

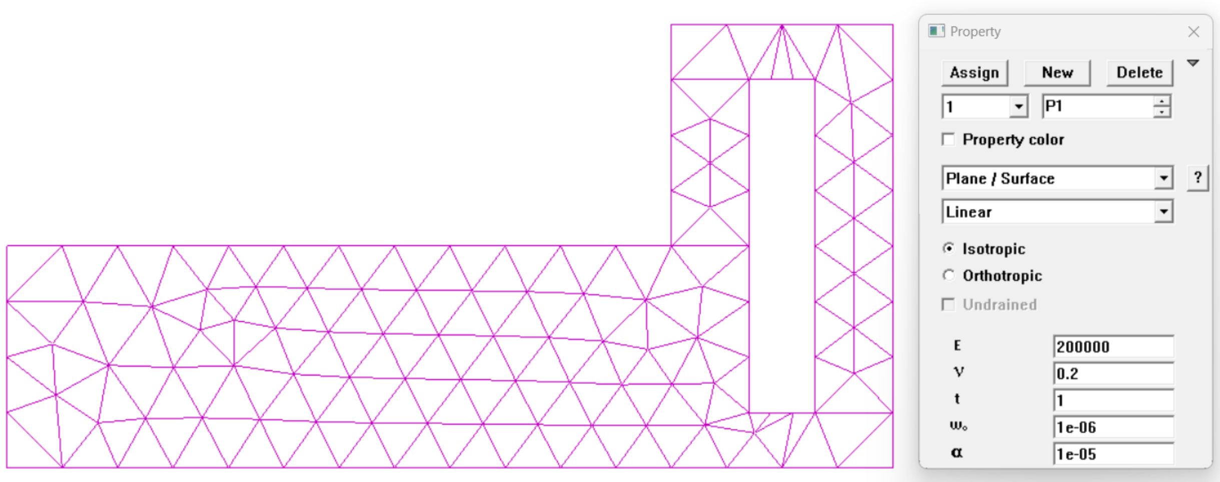
Analysis By VisualFEA

Problem Definition

Plane Stress Analysis:



Property



- Element Type: 3 Node Tria (CST)
- No. of Elements: 184
- No. of Nodes: 125

Boundary:

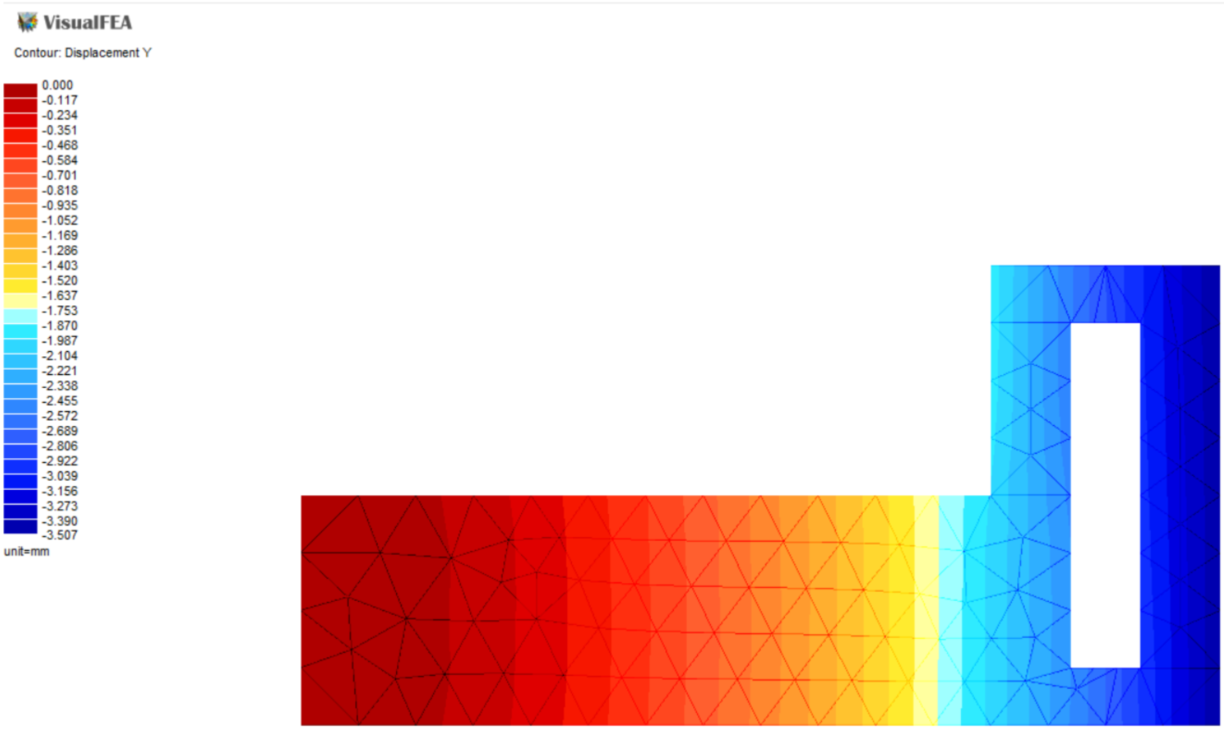
- Nodes in left most surface has fixed.

Load Condition:

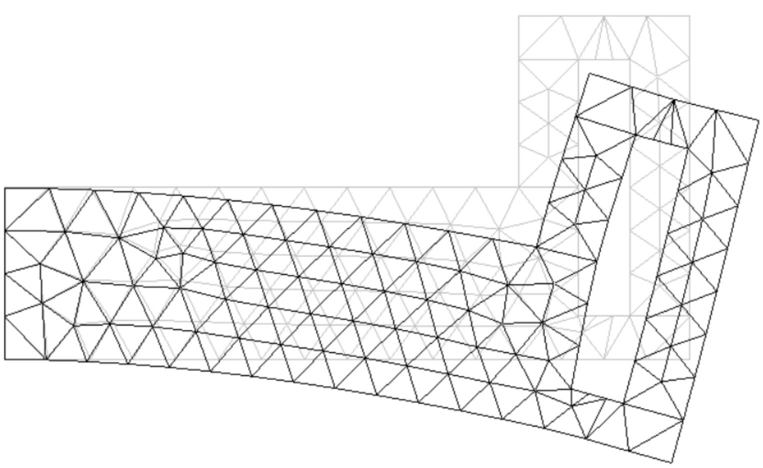
- P in X direction as per above figure: 2000 N
- P in -Y direction as per above figure: -2000 N

Results:

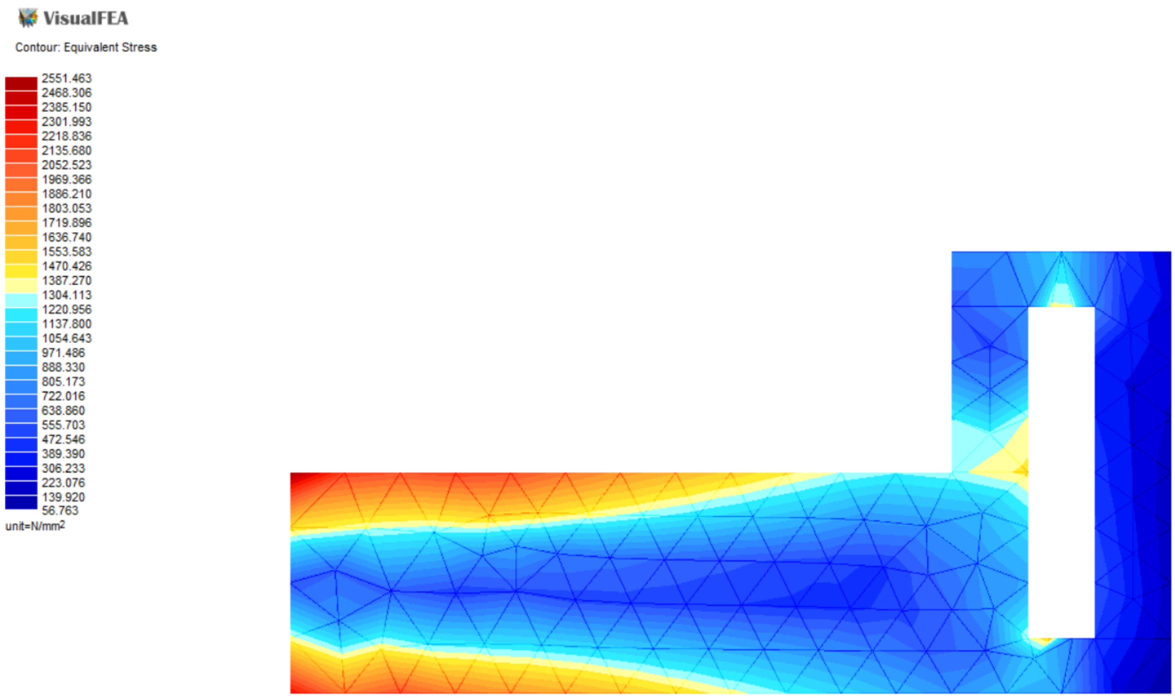
Displacements in Y



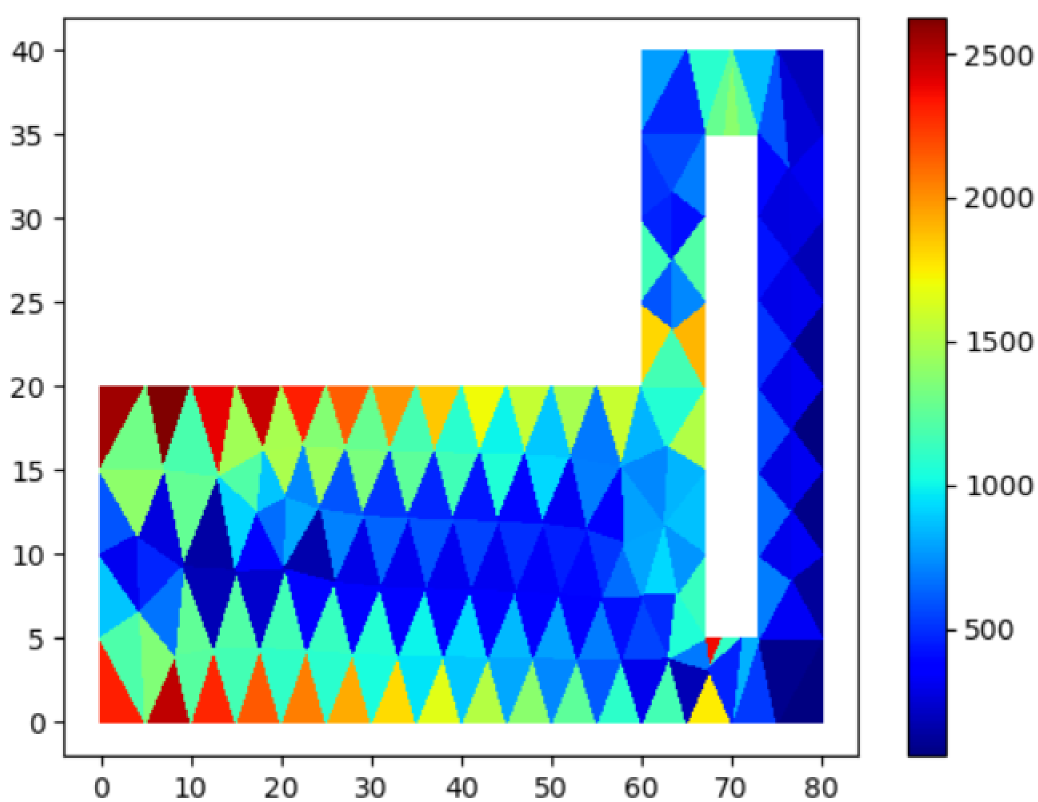
Deformed Shape



Equivalent Stress:

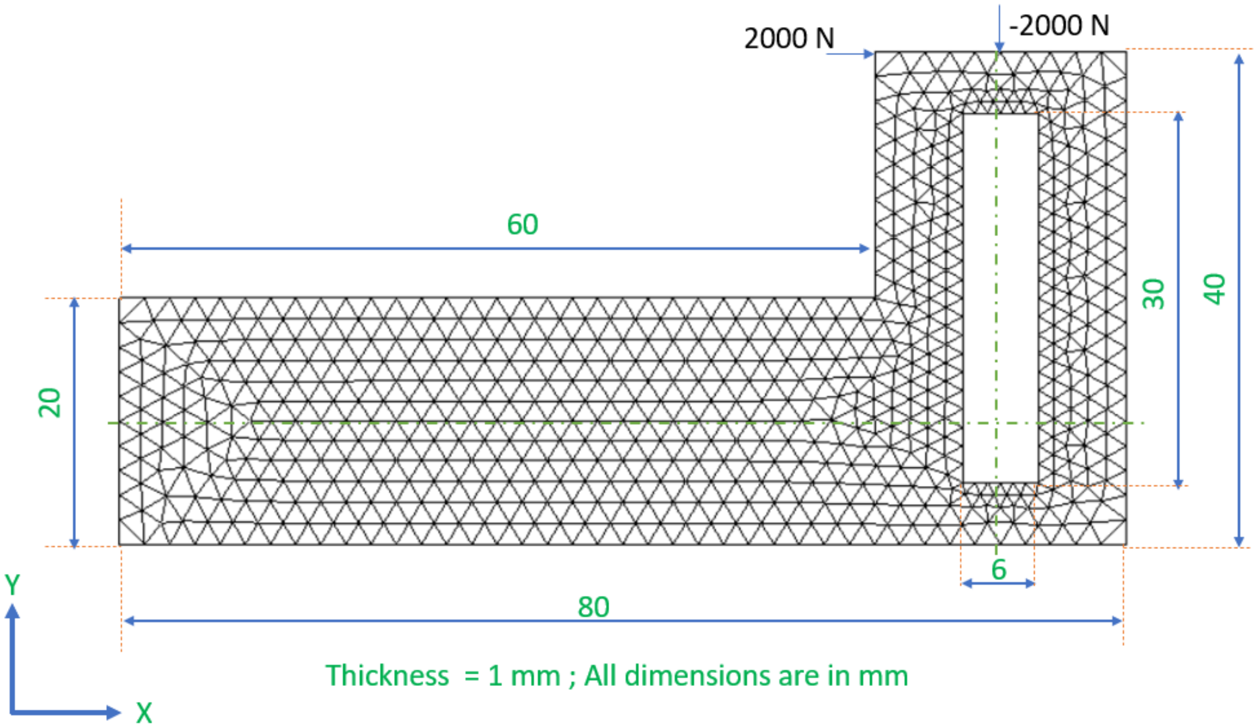


Equivalent Stress by Python code:

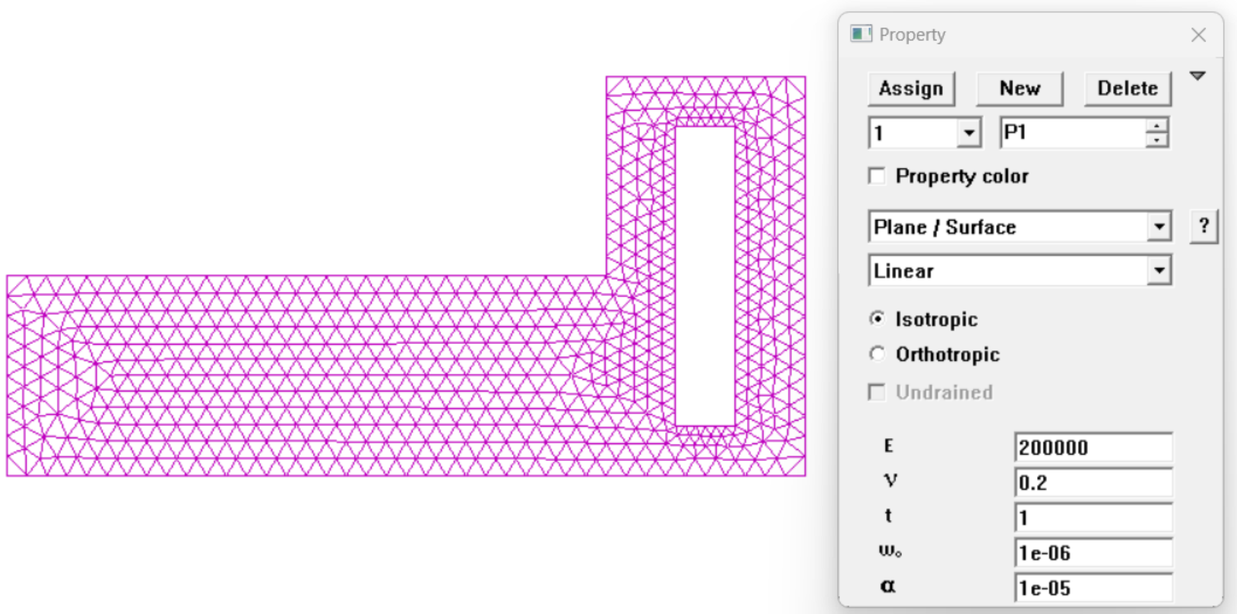


Problem Definition

Plane Stress Analysis: (geometry same as earlier one)



Property



- Element Type: 3 Node Tria (CST)
- No. of Elements: 708
- No. of Nodes: 1244

Boundary:

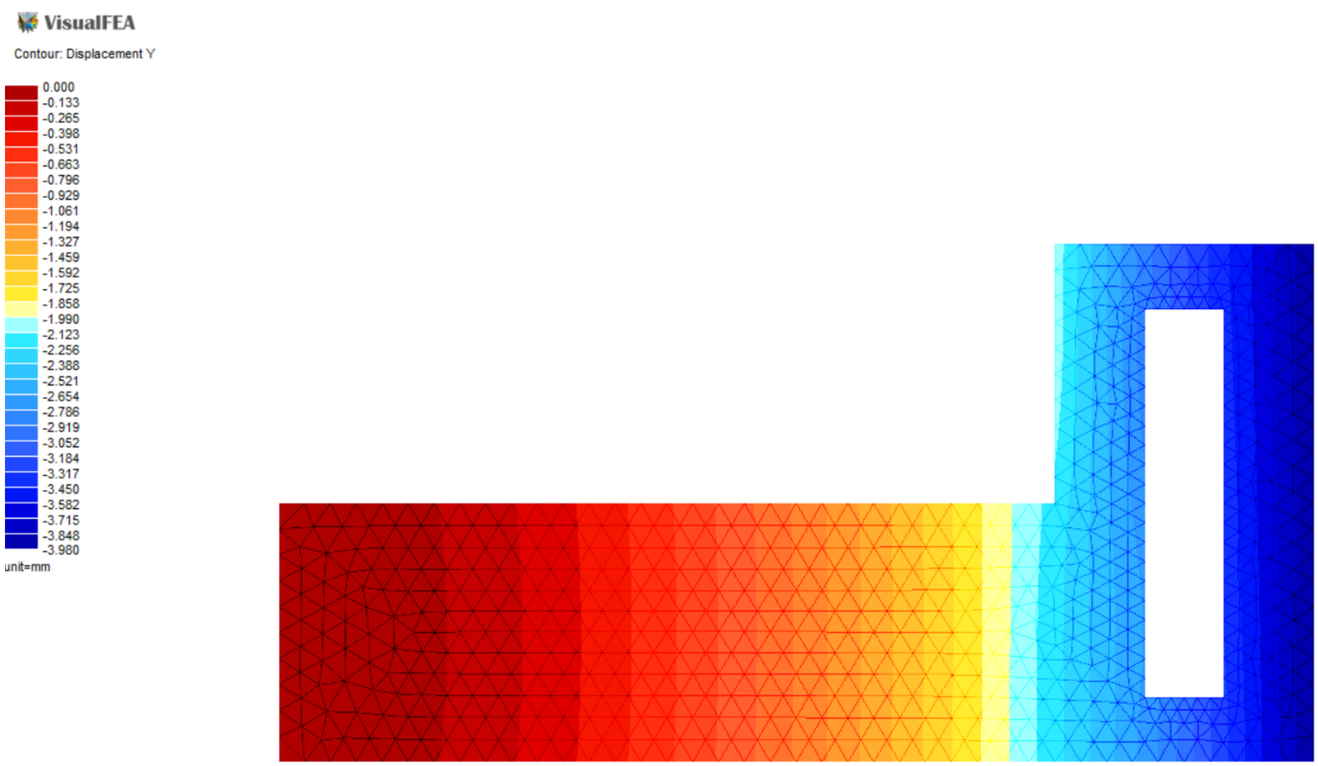
Nodes in left most surface has fixed.

Load Condition:

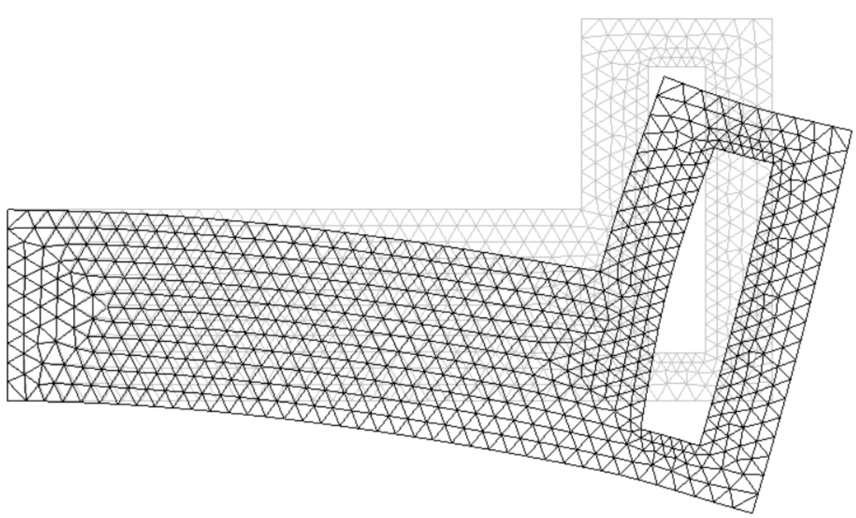
- P in X direction as per above figure: 2000 N
- P in -Y direction as per above figure: -2000 N

Results:

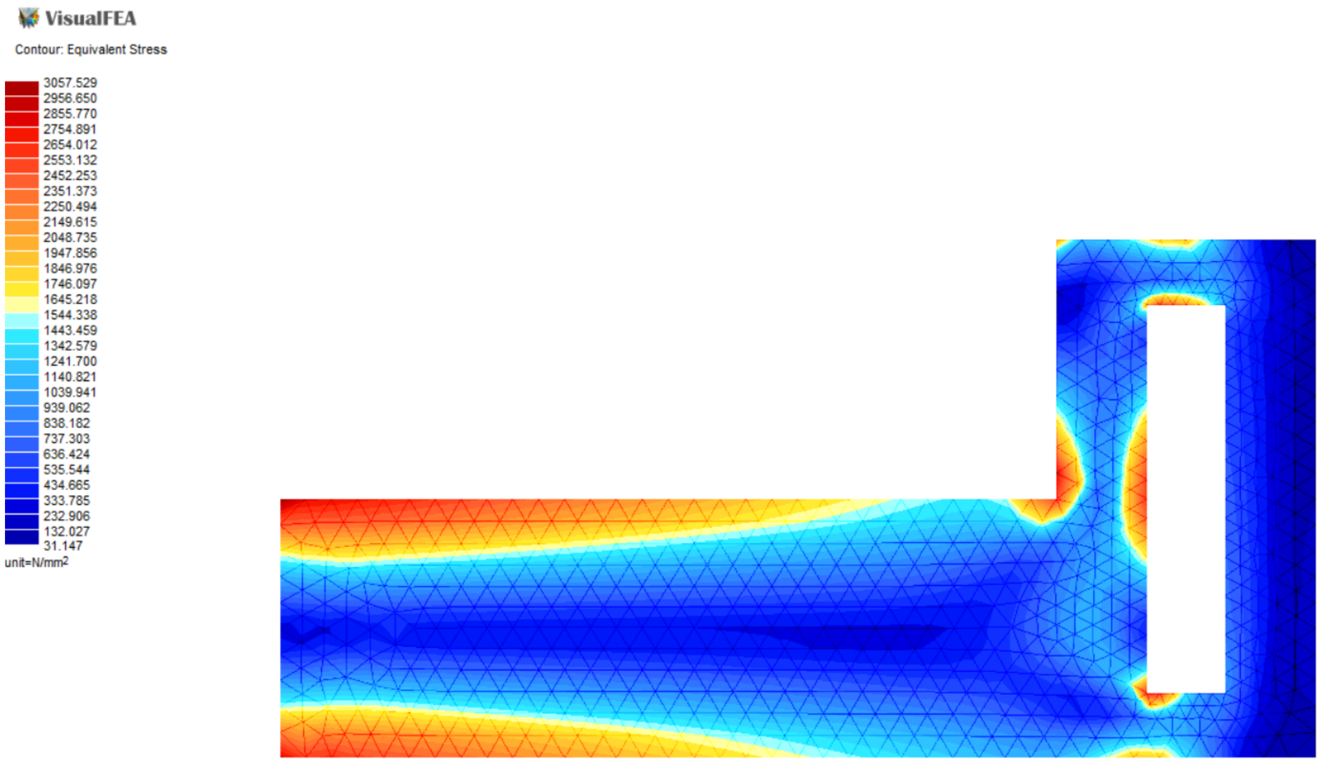
Displacements in Y



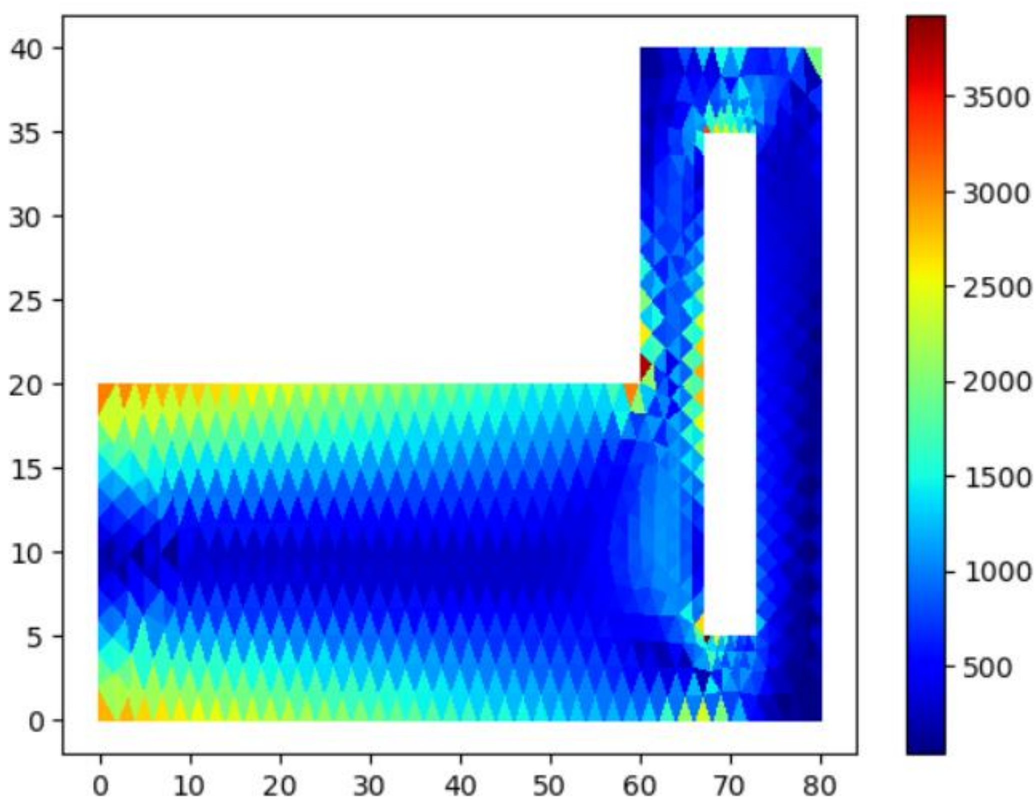
Deformed Shape



Equivalent Stress:

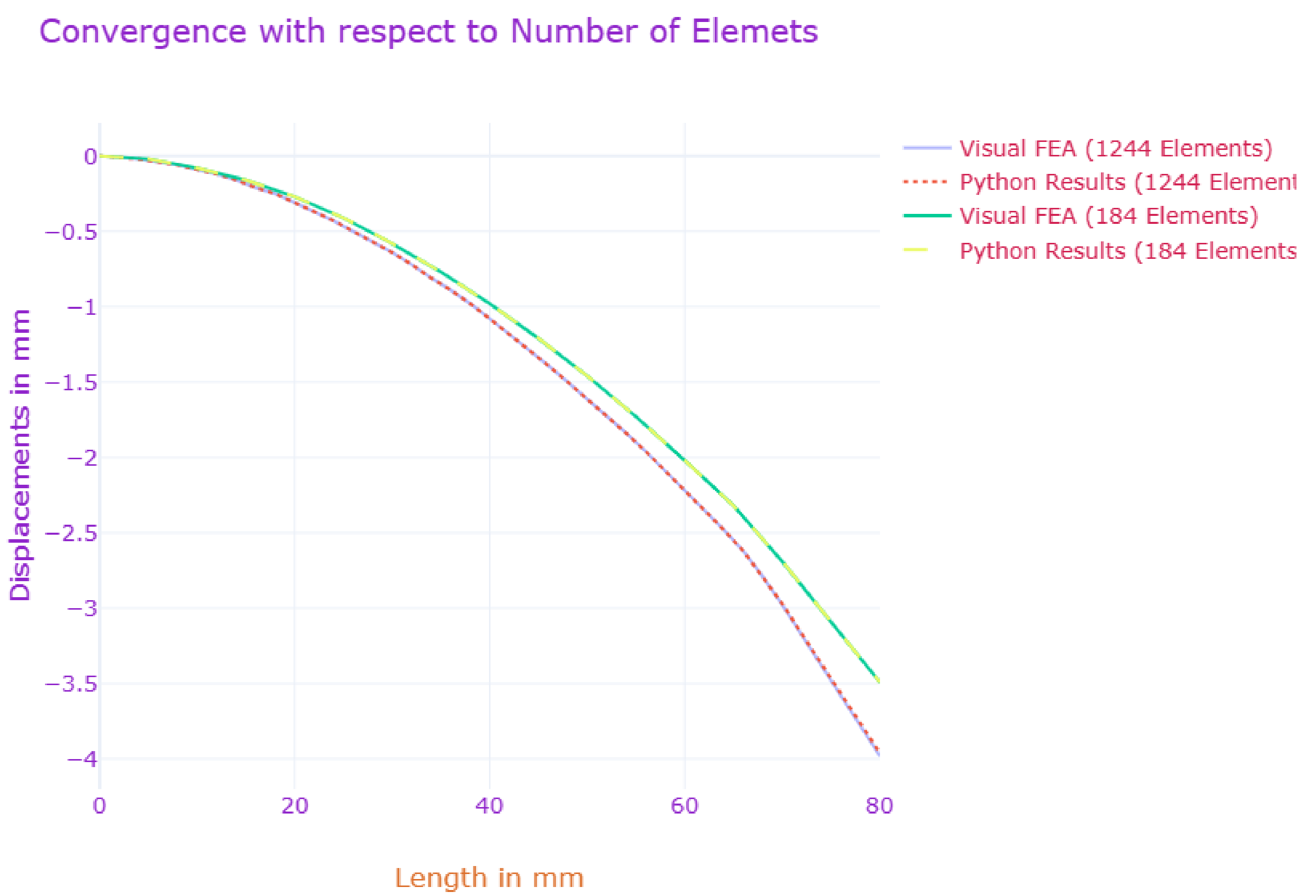


Equivalent Stress by Python code:



Convergence study with respect to No. of Elements

Displacements in Y direction has taken for Nodes in co-ordinate in y = 0.



For 184 Elements

X -Cordinates	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
Y Displacement by VisualFEA	0	-0.02	-0.08	-0.16	-0.27	-0.41	-0.58	-0.77	-0.98	-1.21	-1.46	-1.73	-2.02	-2.32	-2.69	-3.09	-3.49
Y Displacement by Python Code	0	-0.02	-0.08	-0.16	-0.27	-0.41	-0.58	-0.77	-0.98	-1.21	-1.46	-1.73	-2.02	-2.32	-2.69	-3.09	-3.49

For 1244 Elements

X -Cordinates	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
Y Displacement by VisualFEA	0	-0.01	-0.02	-0.04	-0.06	-0.09	-0.12	-0.16	-0.21	-0.25	-0.31	-0.37	-0.43	-0.5	-0.57	-0.64	-0.72	-0.81	-0.89	-0.98	-1.08
Y Displacement by Python Code	0	-0.01	-0.02	-0.04	-0.06	-0.09	-0.12	-0.16	-0.21	-0.25	-0.31	-0.37	-0.43	-0.5	-0.57	-0.64	-0.72	-0.81	-0.89	-0.98	-1.08

X -Cordinates	42	44	46	48	50	52	54	56	58	60	62	64	66	68	70	72	74	76	78	80
Y Displacement by VisualFEA	-1.18	-1.28	-1.39	-1.5	-1.61	-1.72	-1.84	-1.96	-2.09	-2.21	-2.34	-2.48	-2.62	-2.79	-2.98	-3.18	-3.38	-3.58	-3.78	-3.98
Y Displacement by Python Code	-1.18	-1.28	-1.39	-1.5	-1.61	-1.72	-1.84	-1.96	-2.09	-2.21	-2.34	-2.48	-2.62	-2.79	-2.97	-3.17	-3.37	-3.57	-3.76	-3.96

Conclusion:

it can be concluded that the **Python code** and **VisualFEA** software produce identical results when subjected to the same property and boundary conditions. Furthermore, it was found that increasing the number of elements improved the accuracy of the results. These findings demonstrate the effectiveness and **reliability of both the Python code and VisualFEA software** in analyzing structures. Therefore, Python code can be considered suitable for use and it is recommended to continue exploring the capabilities of these tools in future research to further enhance their accuracy and efficiency.