Final assignment

C0TB1716 Tharit Sinsunthorn

Problem

Perform a finite element simulation assuming the object and boundary conditions depicted in the Fig. 1. Decide the dimensions and materials of the object, element type as you wish. Find and refer to Young's modulus E and Poisson's ratio ν from literature.

a) Show the deformed shape.
b) Estimate where the stress is highest.
c) Discuss the obtained results (Effect of the number of element divisions or the size of the object, Difference between plane strain problem and plane stress problem, Effect of element type, etc.

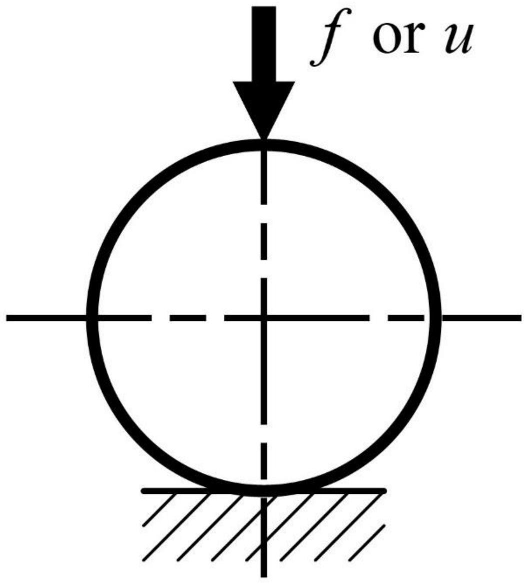


Figure 1. Boundary conditions for the problem.

Meshing

The processes of meshing are implemented in "*meshing.py*". Since the material has circular shape, so the idea of meshing and indices determination are created by iteration along the radial and tangential direction as shown in Fig. (2). In radial direction, the radius line is divided into N paths. Then, the divided points will be rotated counterclockwise with the angle of θ around the origin until the rotation finish one round of circle ($\theta = 2\pi$), which is the end of process of tangential division.

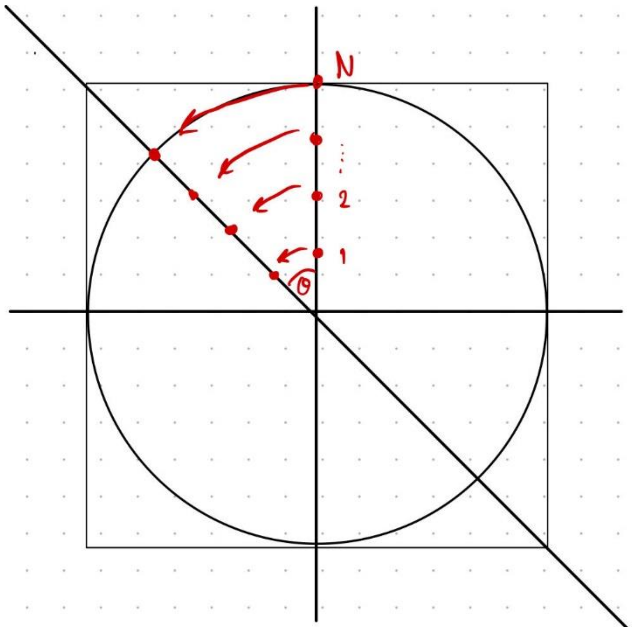
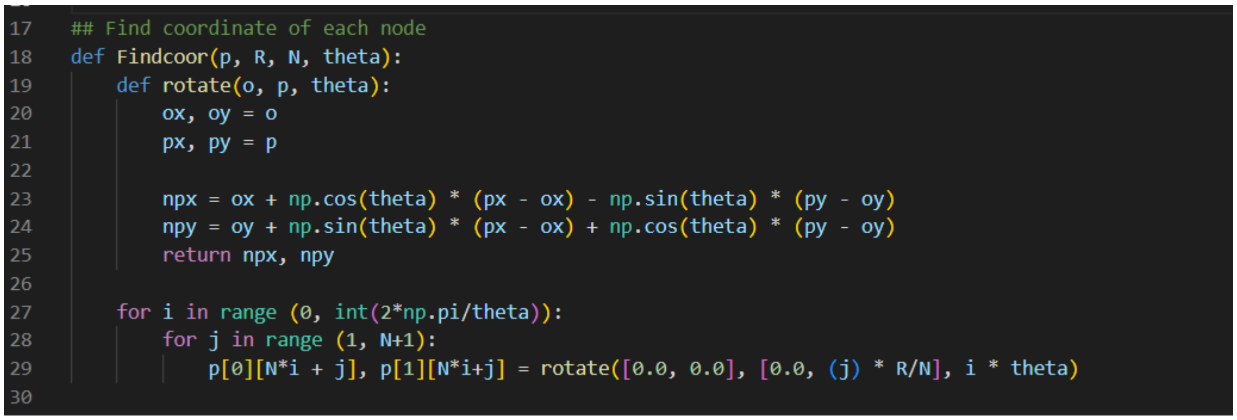


Figure 2. Process of nodes creation.

Therefore, we can obtain all the nodes that will be used for meshing the material. The total number of nodes is determined by multiplication of number of divisions in radial direction and number of divisions in tangential direction, as shown in Eq. (1).

$$number\ of\ nodes = N \times \frac{2\pi}{\theta} = N \times Sec \tag{1}$$



```python
## Find coordinate of each node
def Findcoor(p, R, N, theta):
    def rotate(o, p, theta):
        ox, oy = o
        px, py = p

        npx = ox + np.cos(theta) * (px - ox) - np.sin(theta) * (py - oy)
        npy = oy + np.sin(theta) * (px - ox) + np.cos(theta) * (py - oy)
        return npx, npy

    for i in range (0, int(2*np.pi/theta)):
        for j in range (1, N+1):
            p[0][N*i + j], p[1][N*i+j] = rotate([0.0, 0.0], [0.0, (j) * R/N], i * theta)

```

Figure 3. Node creation function in meshing.py.

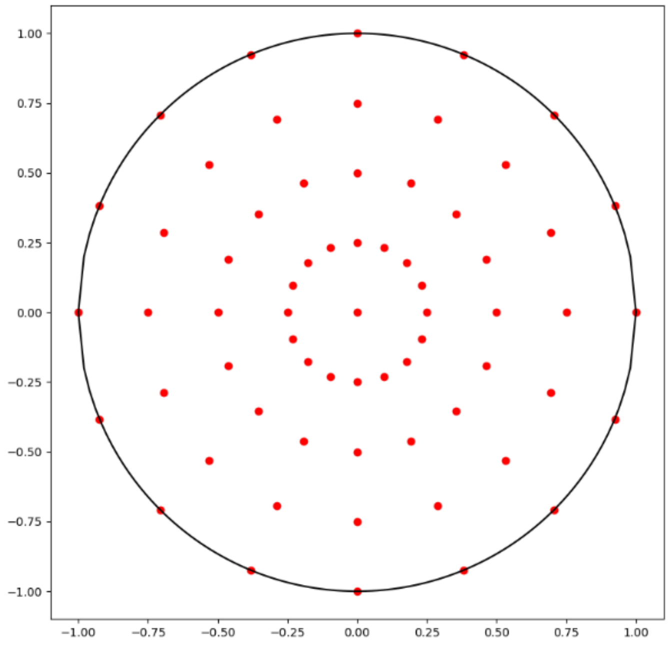


Figure 4. Node creation.

Finally, we can do the meshing by connecting all the nodes to make a triangulation of finite element, as shown in Fig. (5). The number of elements in each tangential division (elements in one slice of cake division) is shown in Eq. (2).

$$mps = 2N - 1 \tag{2}$$

To indicate the indices of each element (three nodes composing the triangular element), in this case, the elements is categorized into 3 types, where each type has different pattern of indices determination.

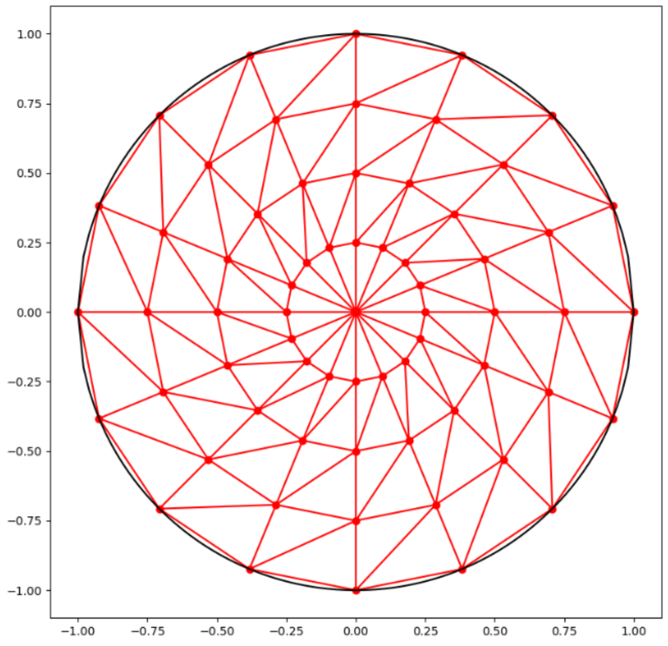
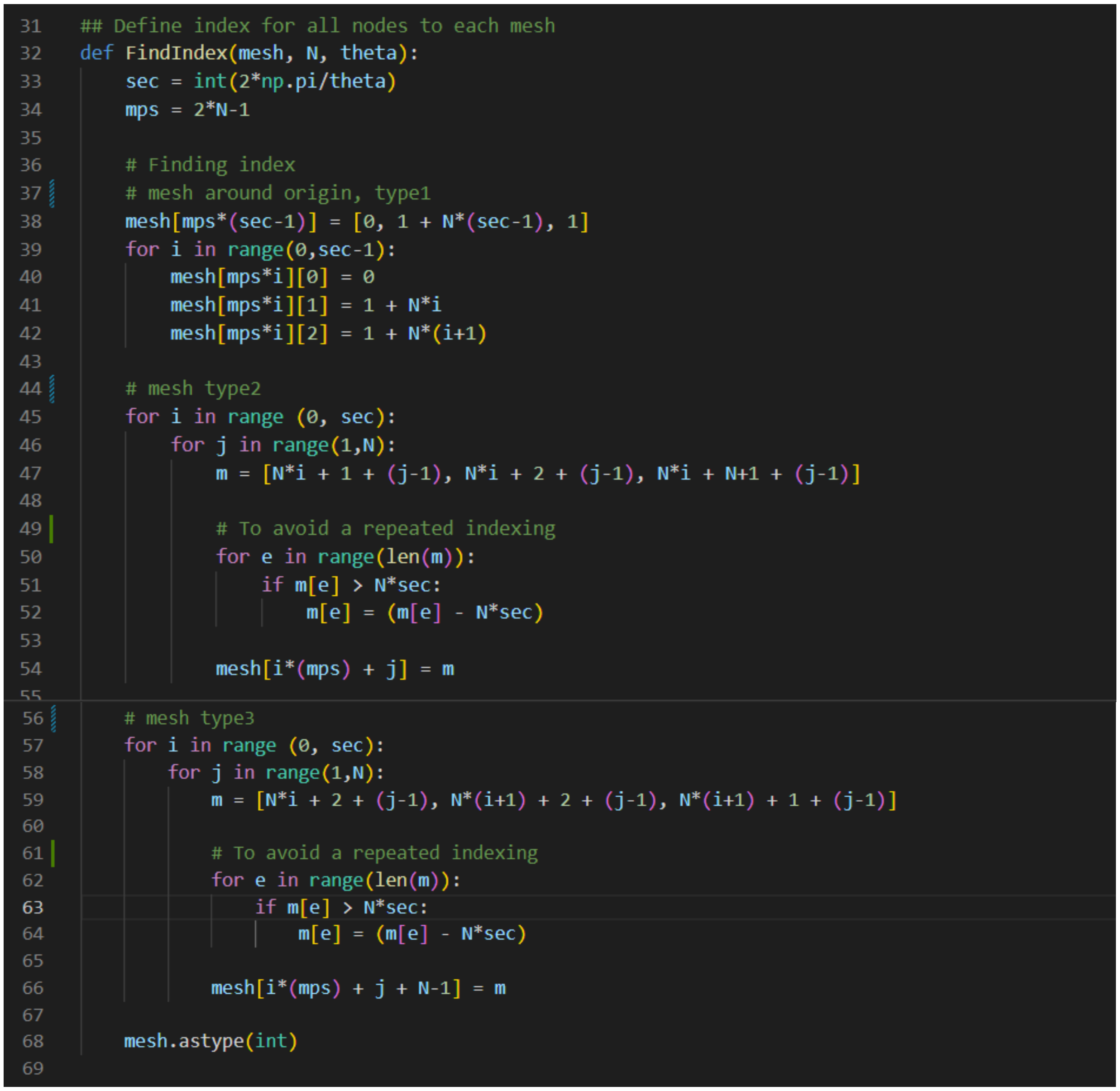


Figure 5. Triangulation of the shape.

The first type of elements is elements surrounding origin, which are green region in Fig. (7). The indices of these elements have a same pattern which is $[0,\ Ni+1,\ N(i+1)+1]$, where $i = 0, 1, 2, \ldots, Sec-1$. So, we can make an iteration function to assign indices to each element.

In the same way, we must investigate the pattern of the other types of elements. The second and third types are illustrated by Fig. (8) and (9). The second type has a nodes pattern of $[Ni+1+(j-1),\ Ni+2+(j-1),\ Ni+N+1+(j-1)]$, while a pattern of the third type is

$[Ni + 2 + (j-1),\ N(i+1) + 2 + (j-1),\ N(i+1) + 1 + (j-1)]$, where $i = 0, 1, 2, \dots, Sec - 1$ and $j = 1, 2, \dots, N - 1$.



```python
## Define index for all nodes to each mesh
def FindIndex(mesh, N, theta):
    sec = int(2*np.pi/theta)
    mps = 2*N-1

    # Finding index
    # mesh around origin, type1
    mesh[mps*(sec-1)] = [0, 1 + N*(sec-1), 1]
    for i in range(0,sec-1):
        mesh[mps*i][0] = 0
        mesh[mps*i][1] = 1 + N*i
        mesh[mps*i][2] = 1 + N*(i+1)

    # mesh type2
    for i in range (0, sec):
        for j in range(1,N):
            m = [N*i + 1 + (j-1), N*i + 2 + (j-1), N*i + N+1 + (j-1)]

            # To avoid a repeated indexing
            for e in range(len(m)):
                if m[e] > N*sec:
                    m[e] = (m[e] - N*sec)

            mesh[i*(mps) + j] = m

    # mesh type3
    for i in range (0, sec):
        for j in range(1,N):
            m = [N*i + 2 + (j-1), N*(i+1) + 2 + (j-1), N*(i+1) + 1 + (j-1)]

            # To avoid a repeated indexing
            for e in range(len(m)):
                if m[e] > N*sec:
                    m[e] = (m[e] - N*sec)

            mesh[i*(mps) + j + N-1] = m

    mesh.astype(int)
```

Figure 6. Function for giving index number to each element.

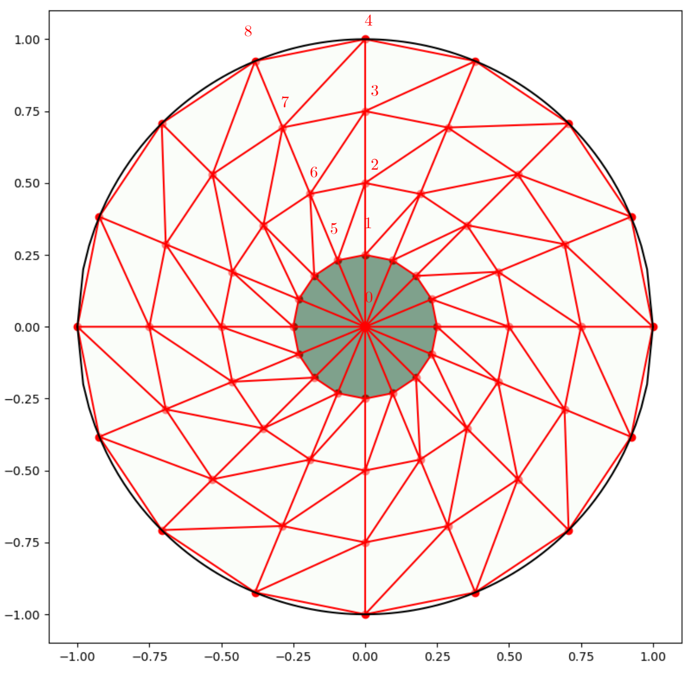


Figure 7. Region of the first type of elements.

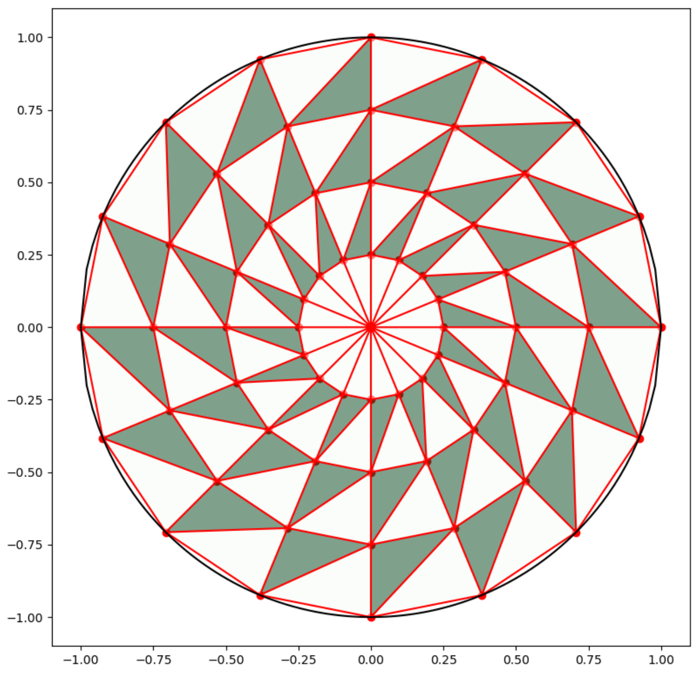


Figure 8. Region of the second type of elements

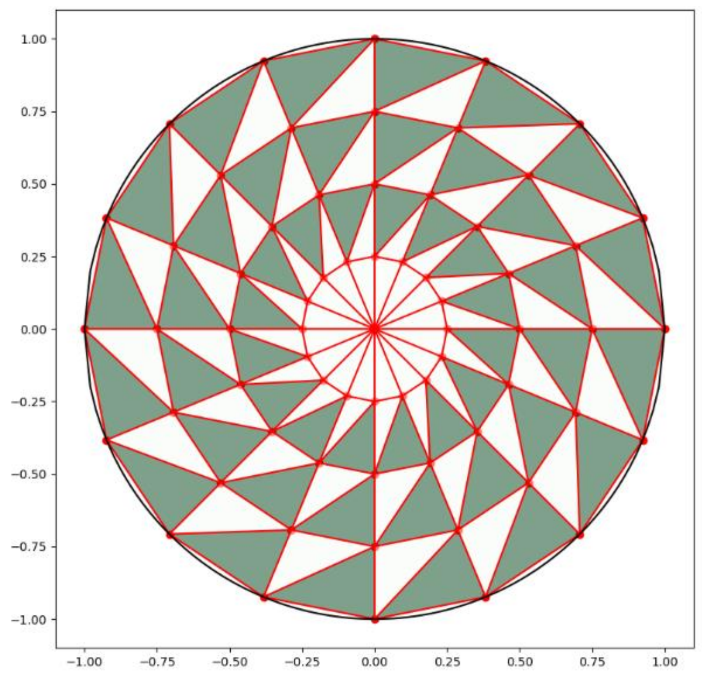


Figure 9. Region of the third type of elements

Stiffness matrix

The element stiffness matrix of the i^{th} element is determined by Eq. (3)

$$K_e{}^{(i)} = a_i B_i^T D_i B_i \tag{3}$$

Where a_i is an area of the element is given by Eq. (4), where x_i, y_i are coordinates of each node of the element.

$$a = \frac{1}{2}\det\left(\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}\right) \tag{4}$$

Matrix B is defined by Eq. (5)

$$B = \frac{1}{2a}\begin{bmatrix} y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{bmatrix} \tag{5}$$

Matrix D is defined by Eq. (6) and (7) for plane strain condition and plane stress condition, respectively.

$$D_{Plane\ strain} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{6}$$

$$D_{Plane\ stress} = \frac{E}{(1+\nu)(1-\nu)}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{7}$$

The element stiffness equation is given by Eq. (8) and (9)

$$K_e{}^{(i)} = u^{(i)} f^{(i)} \tag{8}$$

$$K_e{}^{(i)}\begin{bmatrix} u_{1x}{}^{(i)} \\ u_{1y}{}^{(i)} \\ u_{2x}{}^{(i)} \\ u_{2y}{}^{(i)} \\ u_{3x}{}^{(i)} \\ u_{3y}{}^{(i)} \end{bmatrix} = \begin{bmatrix} f_{1x}{}^{(i)} \\ f_{1y}{}^{(i)} \\ f_{2x}{}^{(i)} \\ f_{2y}{}^{(i)} \\ f_{3x}{}^{(i)} \\ f_{3y}{}^{(i)} \end{bmatrix} \tag{9}$$

To make the total stiffness equation, we have to combine the stiffness matrix of all elements together by matching the elements having corresponding index. The total stiffness matrix, K should have a dimension of $2(number\ of\ nodes) \times 2(number\ of\ nodes)$.

$$K\begin{bmatrix} {u_{1x}}^{(1)} \\ {u_{1y}}^{(1)} \\ {u_{2x}}^{(1)} \\ {u_{2y}}^{(1)} \\ \vdots \\ {u_{3y}}^{(nodes)} \end{bmatrix} = \begin{bmatrix} {f_{1x}}^{(1)} \\ {f_{1y}}^{(1)} \\ {f_{2x}}^{(1)} \\ {f_{2y}}^{(1)} \\ \vdots \\ {f_{3y}}^{(nodes)} \end{bmatrix} \tag{10}$$

The algorithm for constructing the total stiffness matrix is shown below, where M is an array containing indices of all triangle elements. We can see that, by iterate the program to search for indices of each meshing element, only the values from the elements containing a corresponding index will be added to each corresponding position of the matrix K.

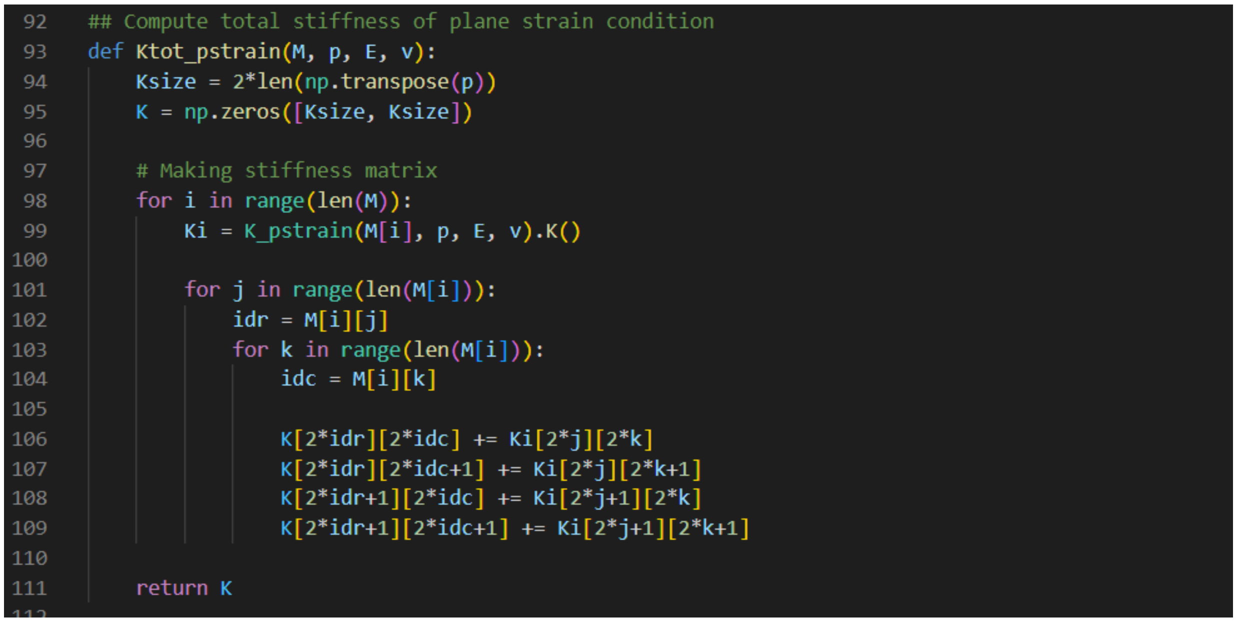
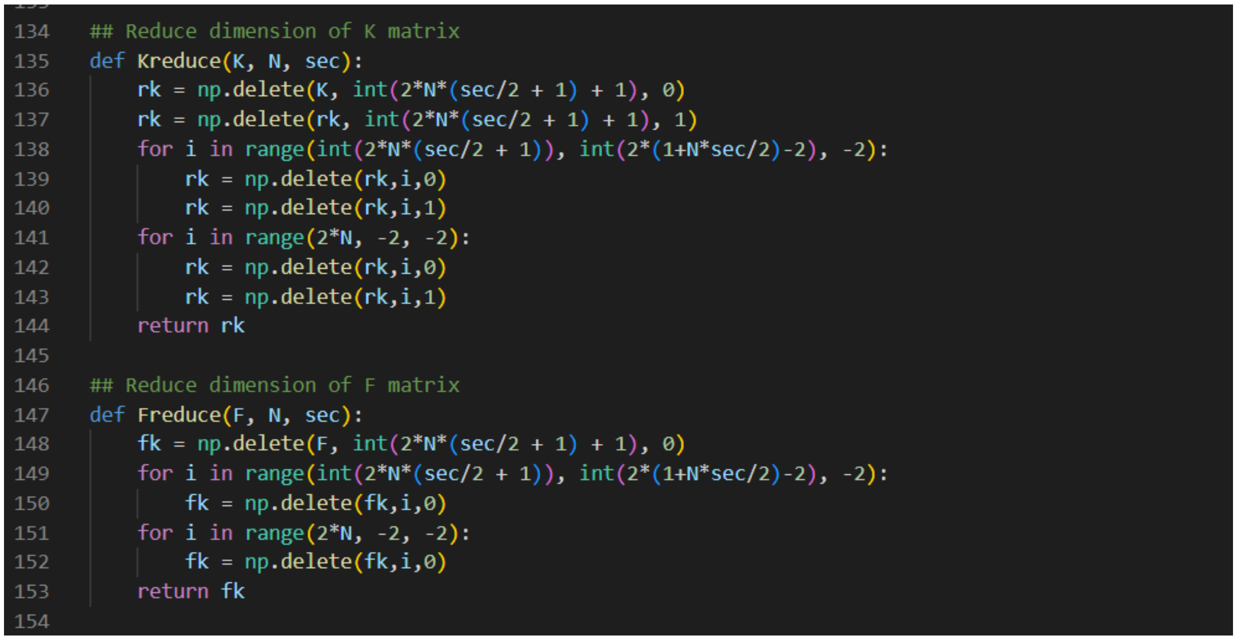

```python
## Compute total stiffness of plane strain condition
def Ktot_pstrain(M, p, E, v):
    Ksize = 2*len(np.transpose(p))
    K = np.zeros([Ksize, Ksize])

    # Making stiffness matrix
    for i in range(len(M)):
        Ki = K_pstrain(M[i], p, E, v).K()

        for j in range(len(M[i])):
            idr = M[i][j]
            for k in range(len(M[i])):
                idc = M[i][k]

                K[2*idr][2*idc] += Ki[2*j][2*k]
                K[2*idr][2*idc+1] += Ki[2*j][2*k+1]
                K[2*idr+1][2*idc] += Ki[2*j+1][2*k]
                K[2*idr+1][2*idc+1] += Ki[2*j+1][2*k+1]

    return K
```

Figure 10. Total stiffness determination function.

Displacement and Deformation

From the boundary conditions, given in Fig. 1, we get the assumption that nodal displacement in x- direction of the indices at $x = 0$ is zero because of symmetry of the applied load and the node at the floor has zero displacement in both directions. Consequently, we can reduce the dimension of the matrix K, by removing row ad column which corresponds to u_x of those indices and u_y of the node at the bottom of the surface. This gives an advantage to reduce calculation cost. Also, the dimension of nodal force matrix should be reduced to get the same size as reduced matrix K.



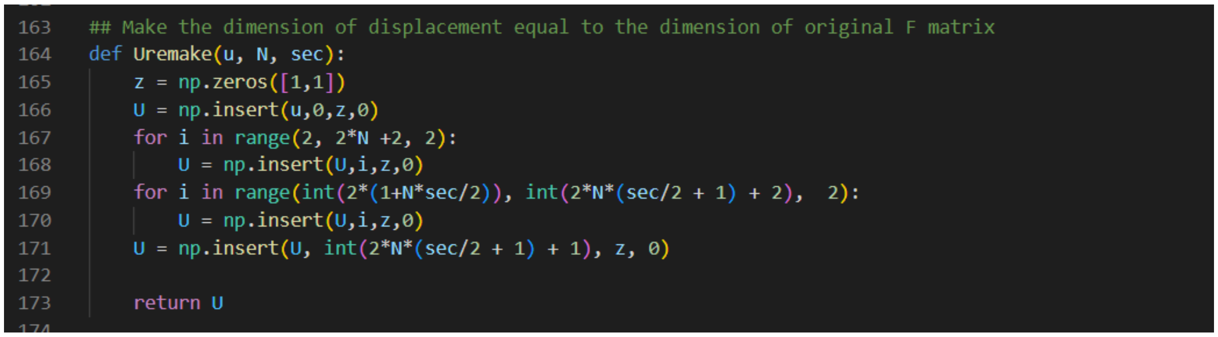
```python
## Reduce dimension of K matrix
def Kreduce(K, N, sec):
    rk = np.delete(K, int(2*N*(sec/2 + 1) + 1), 0)
    rk = np.delete(rk, int(2*N*(sec/2 + 1) + 1), 1)
    for i in range(int(2*N*(sec/2 + 1)), int(2*(1+N*sec/2)-2), -2):
        rk = np.delete(rk,i,0)
        rk = np.delete(rk,i,1)
    for i in range(2*N, -2, -2):
        rk = np.delete(rk,i,0)
        rk = np.delete(rk,i,1)
    return rk

## Reduce dimension of F matrix
def Freduce(F, N, sec):
    fk = np.delete(F, int(2*N*(sec/2 + 1) + 1), 0)
    for i in range(int(2*N*(sec/2 + 1)), int(2*(1+N*sec/2)-2), -2):
        fk = np.delete(fk,i,0)
    for i in range(2*N, -2, -2):
        fk = np.delete(fk,i,0)
    return fk
```

Figure 11. Function for reducing matrices.

According to Eq. (), the nodal displace matrix can be calculated by Eq. (8)

$$u_{reduced} = K_{reduced}{}^{-1} f_{reduced} \tag{11}$$

To convert $u_{reduced}$ to the original dimension, we must insert the removed nodal displacement of indices at $x = 0$ back to the matrix.



```
## Make the dimension of displacement equal to the dimension of original F matrix
def Uremake(u, N, sec):
    z = np.zeros([1,1])
    U = np.insert(u,0,z,0)
    for i in range(2, 2*N +2, 2):
        U = np.insert(U,i,z,0)
    for i in range(int(2*(1+N*sec/2)), int(2*N*(sec/2 + 1) + 2),  2):
        U = np.insert(U,i,z,0)
    U = np.insert(U, int(2*N*(sec/2 + 1) + 1), z, 0)

    return U
```

Figure 12. Function for inserting ignored elements back to the matrix.

Finally, we can simulate deformation of the object caused by a single load at the top of the surface by sum up the coordinate of nodes before the force is applied with nodal displacement. The deformation of material is illustrated in Fig. (13).

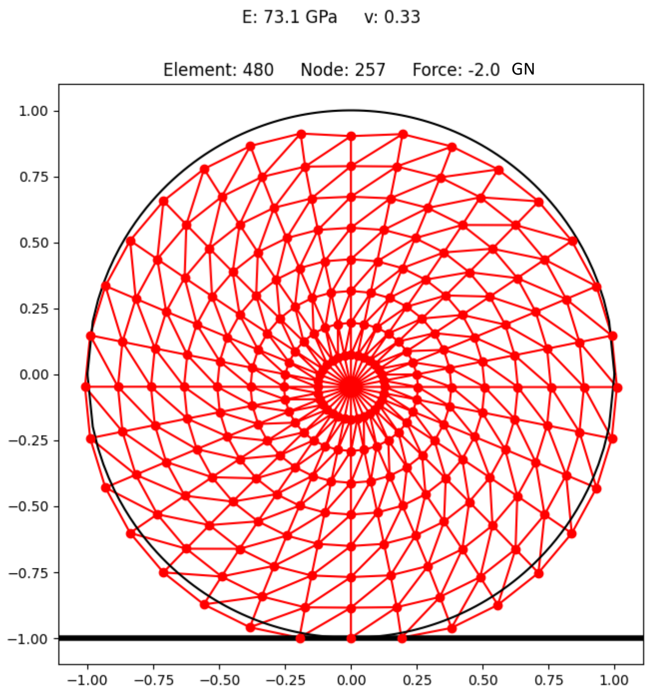
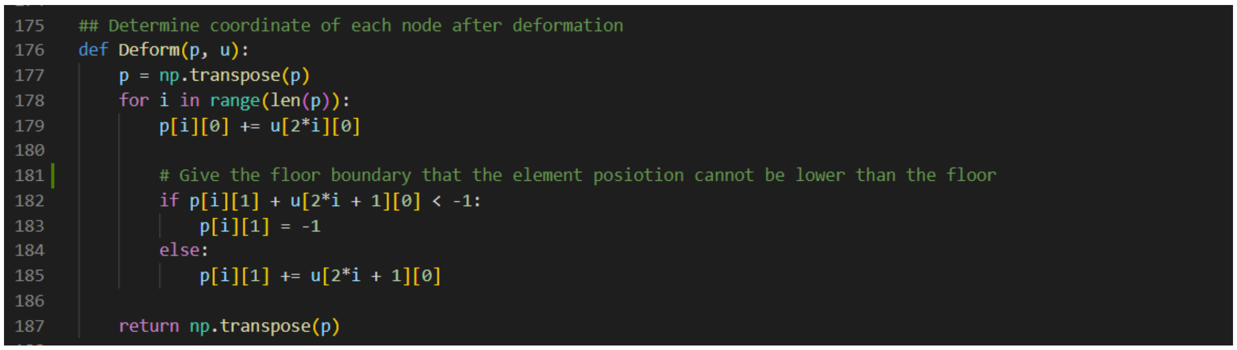


Figure 13. Deformation after force is applied.



```
## Determine coordinate of each node after deformation
def Deform(p, u):
    p = np.transpose(p)
    for i in range(len(p)):
        p[i][0] += u[2*i][0]

        # Give the floor boundary that the element posiotion cannot be lower than the floor
        if p[i][1] + u[2*i + 1][0] < -1:
            p[i][1] = -1
        else:
            p[i][1] += u[2*i + 1][0]

    return np.transpose(p)
```

Figure 14. Deformation function.

The boundary off the floor should be concerned also, so the coordinate of the node cannot be lower than the floor position.

In this case, the chosen material is Aluminum 2024-T4. The modulus of elasticity is $E = 73.1\ GPa$ and Poisson's ratio is $\nu = 0.33$.

(Reference of material's information: ASM Aerospace Specification Metals https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma2024t4&fbclid=IwAR3ZeUByBTPEBIZMDWcBXyi2moIzXyskv2zjbF48nPd-16sejS7SjtHiFkw)

Maximum stress

The stress components can be obtained by multiplication of matrix D, B, and nodal displacement u. The elastic constitutive equation is given by Eq. (12).

$$\sigma = DBu = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} \quad (12)$$

Using color contour, the maximum stress of each component can be determined, as shown in the results of Fig. 15, 16, and 17.

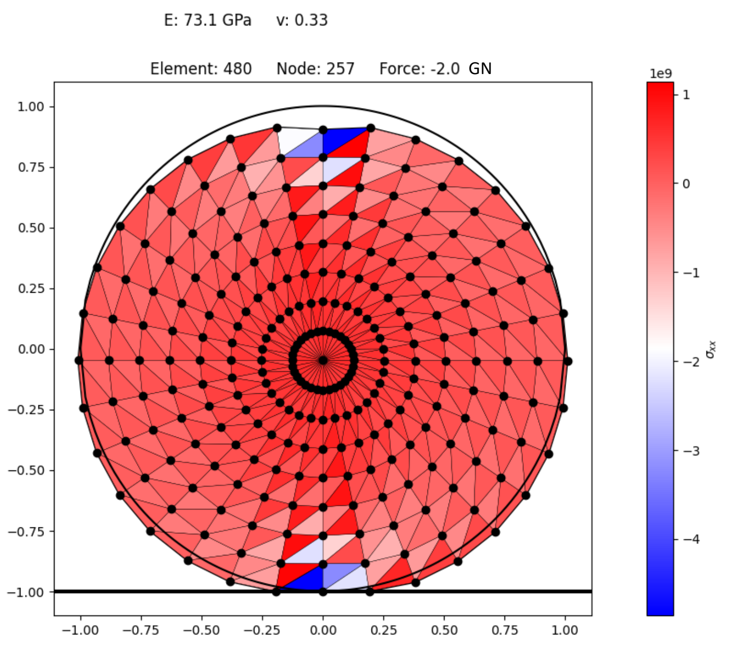


Figure 15. Deformation and color map of σ_{xx}.

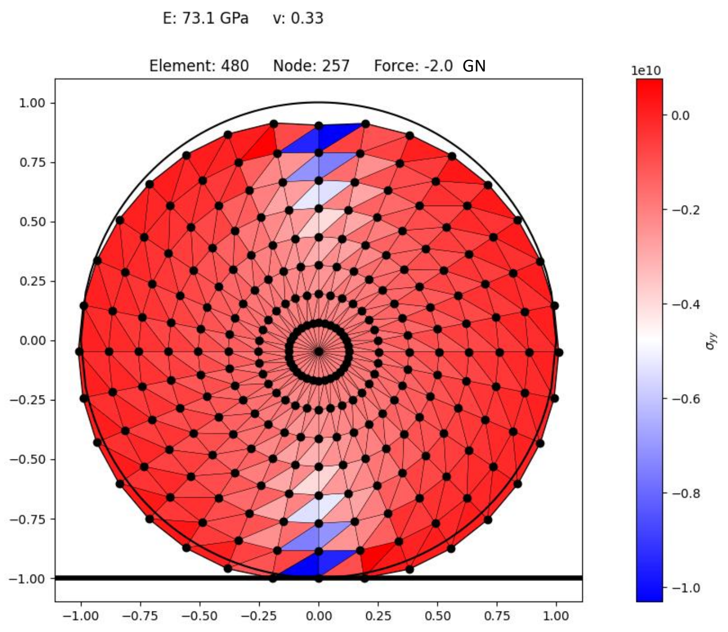


Figure 16. Deformation and color map of σ_{yy}.

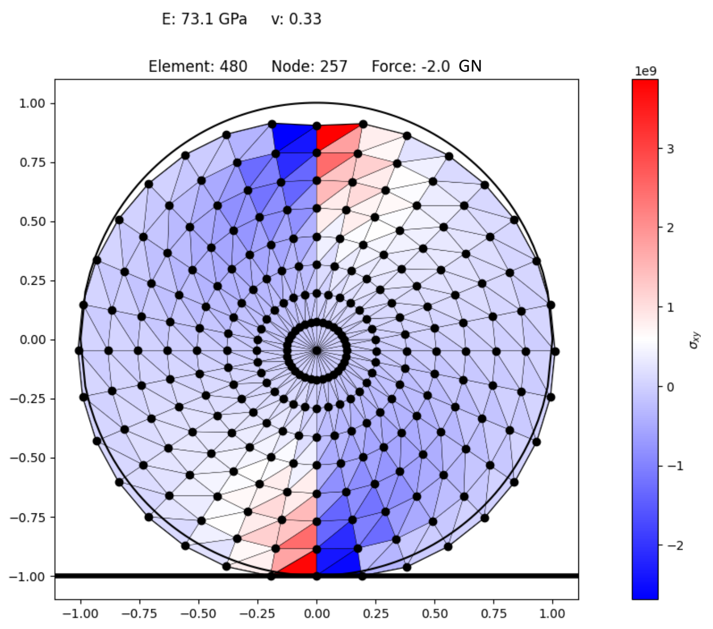
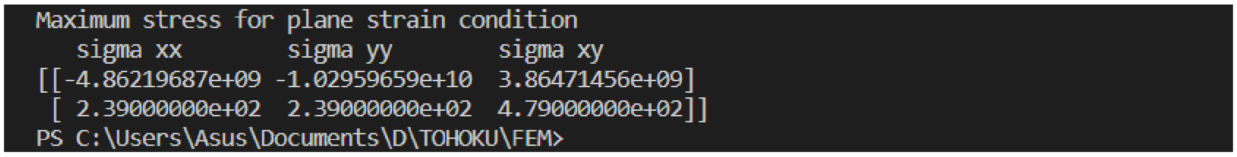


Figure 17. Deformation and color map of σ_{xy}.

We can see that maximum compressive stress occur around the region where the force is applied and the bottom of the surface, where the material contact with the floor. Tensile stress of σ_{xx} and σ_{yy} components are high around left and right of the surface, while that of σ_{xy} is quite sparse. In addition, the maximum stress components, and their positions (reported in the position number of the mesh) are also displayed.



```
Maximum stress for plane strain condition
   sigma xx        sigma yy        sigma xy
[[-4.86219687e+09 -1.02959659e+10  3.86471456e+09]
 [ 2.39000000e+02  2.39000000e+02  4.79000000e+02]]
PS C:\Users\Asus\Documents\D\TOHOKU\FEM>
```

Figure 18. Maximum stress and their positions.

Discussion

Effect of number of element divisions

Number of the elements is one of the variables which sensitively affect the result of the simulation. In this case, we use the same material, Aluminum 2024-T4. The applied force is $F = -1.0\ GN$ and the assumption follows plane strain problem. Fig. 19, 20 and 21 show the result of simulation with 30 element divisions. It is obvious that the shape is not fit to the original shape and the deformation is not detailed enough to be able to explain the behavior of deformation. The stress components are not incomprehensible since the size of the elements cover too much area of the material, so we cannot specify the information of the stress exactly.

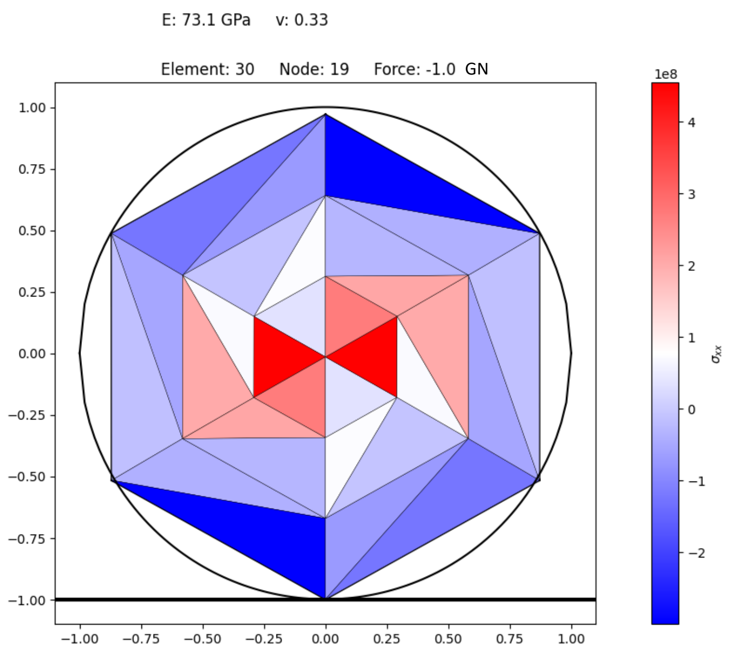


Figure 19. Simulation of σ_{xx} with 30 elements.

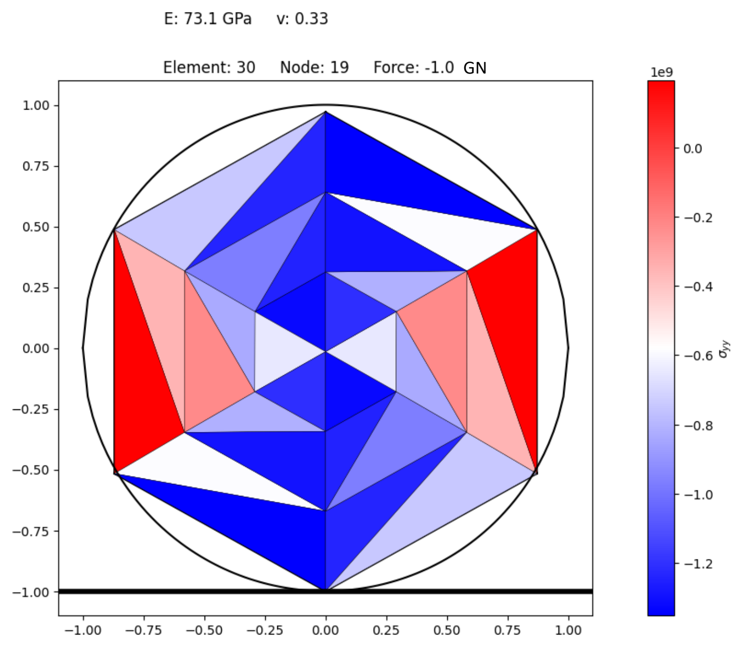


Figure 20. Simulation of σ_{yy} with 30 elements.

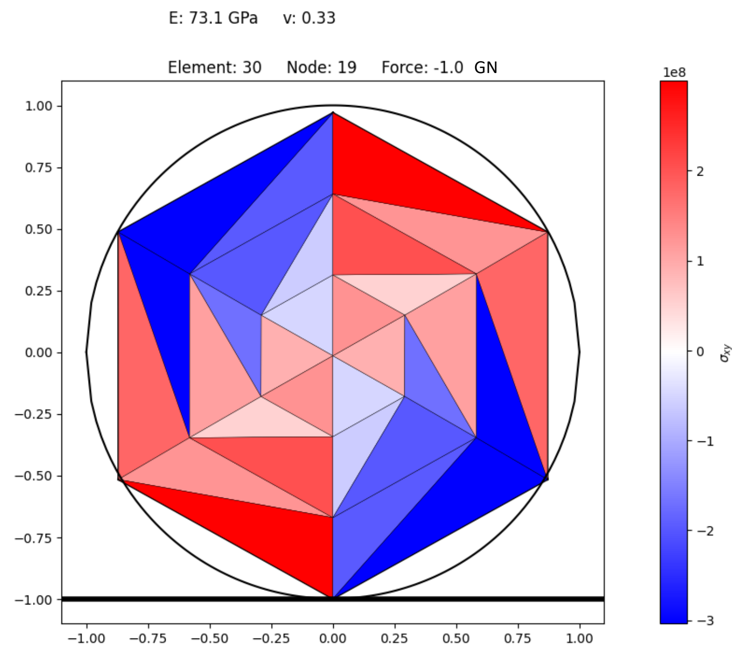


Figure 21. Simulation of σ_{xy} with 30 elements.

Compare to the results of meshing with 3,120 element divisions, as shown in Fig. 22, 23, and 24. The deformation shape is looked much more reasonable because more elements provides us more data to analyze the object. The element stress components also become more utilizable to describe the distribution of stress inside the material.

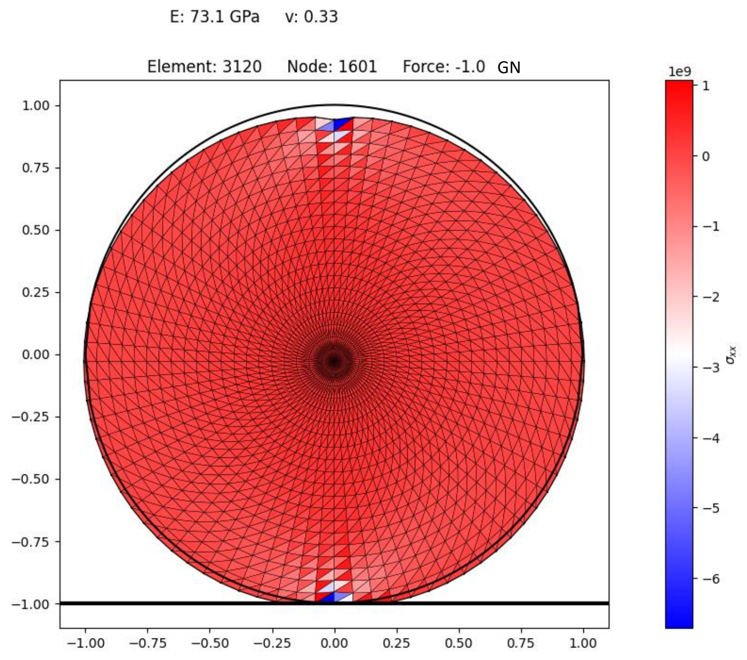


Figure 22. Simulation of σ_{xx} with 3120 elements.

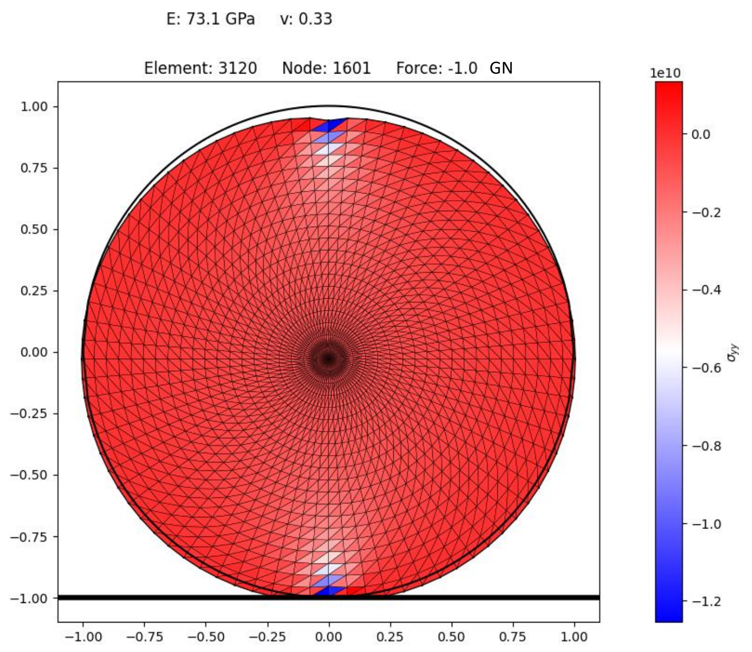


Figure 23. Simulation of σ_{yy} with 3120 elements.

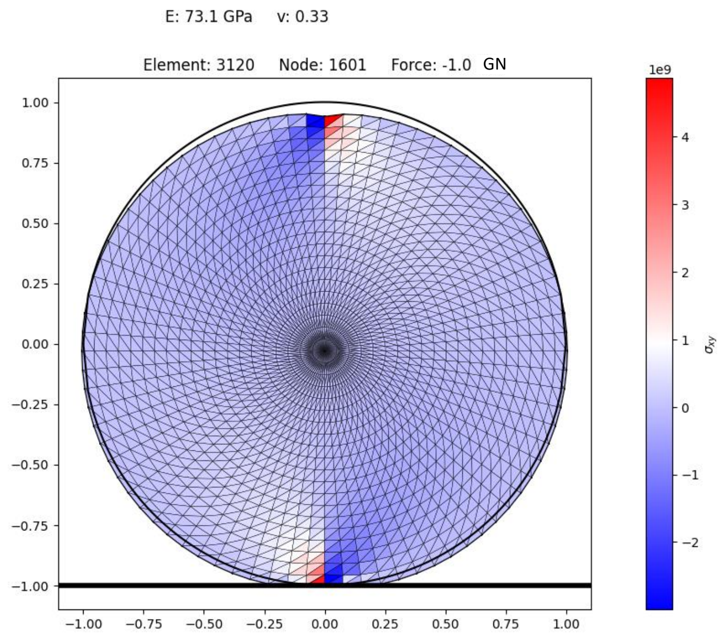
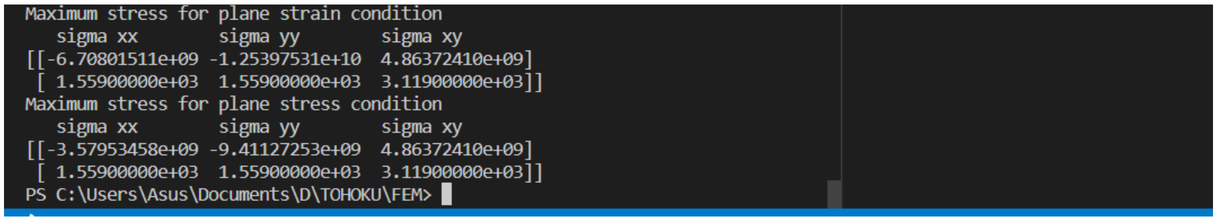


Figure 24. Simulation of σ_{xy} with 3120 elements.

Difference between plane strain problem and plane stress problem

Fig. 25, 26, and 27 show the result of simulation under plane stress problem. Other parameters are the same as result in Fig. 22, 23, and 24. The result show less different in deformation but the distribution of stress components and also the maximum of them are different. Since the assumption of the two problems affect some components of the D matrix.



```
Maximum stress for plane strain condition
    sigma xx        sigma yy        sigma xy
[[-6.70801511e+09 -1.25397531e+10  4.86372410e+09]
 [ 1.55900000e+03  1.55900000e+03  3.11900000e+03]]
Maximum stress for plane stress condition
    sigma xx        sigma yy        sigma xy
[[-3.57953458e+09 -9.41127253e+09  4.86372410e+09]
 [ 1.55900000e+03  1.55900000e+03  3.11900000e+03]]
PS C:\Users\Asus\Documents\D\TOHOKU\FEM>
```

Figure 25. Maximum stress components under plane strain problem and plane stress problem.

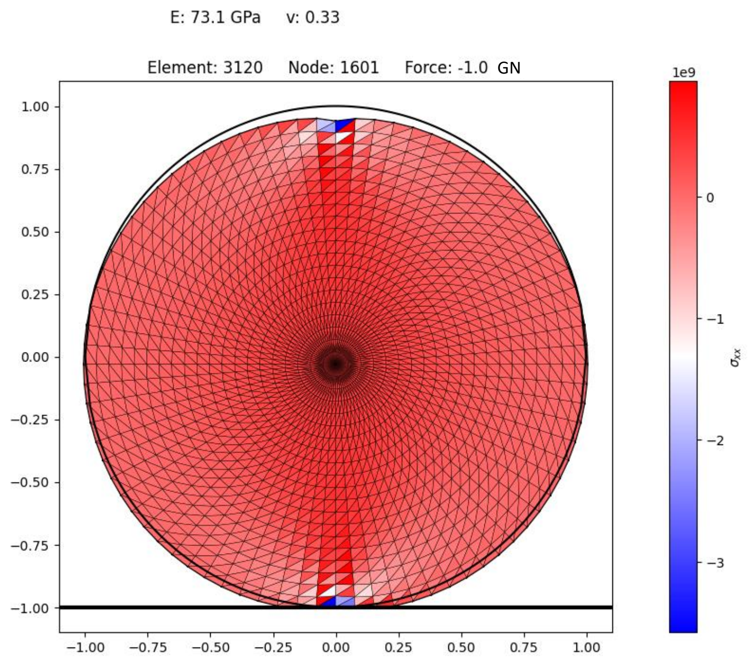


Figure 26. Deformation and σ_{xx} with plane stress condition.

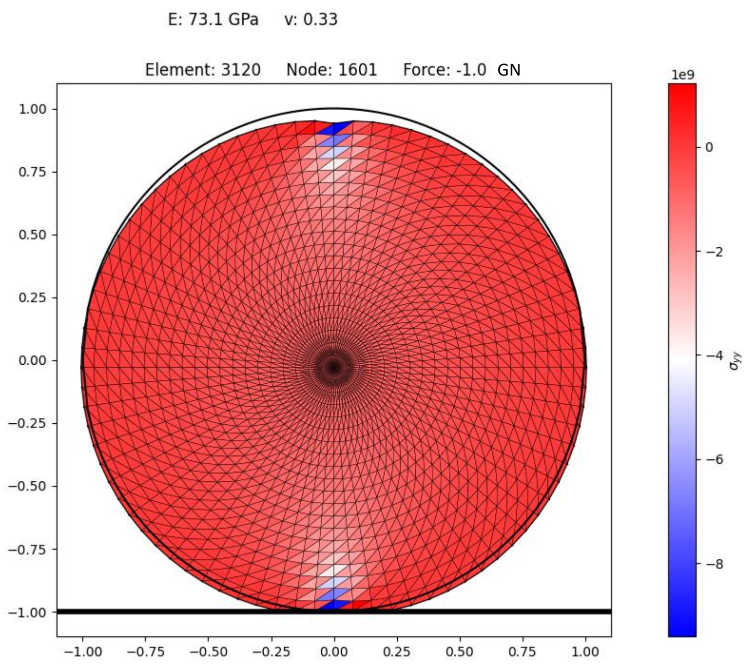


Figure 27. Deformation and σ_{yy} with plane stress condition.

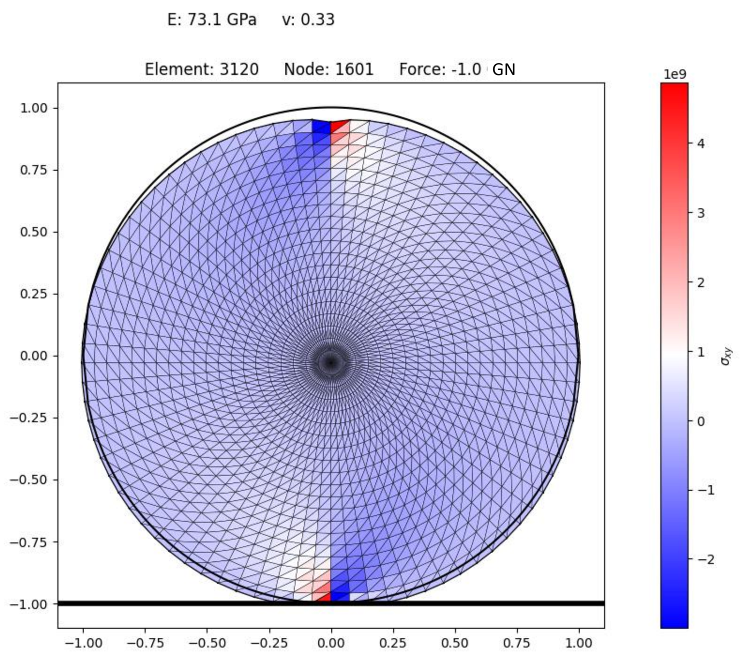


Figure 28. Deformation and σ_{xy} with plane stress condition.

Comparison of stiff material and soft material

In this section, acrylic material is examined. The Young modulus of acrylic is $E_{acry} = 3.2\ GPa$ and Poisson ratio is $\nu_{acry} = 0.37$. The applied force is $-0.1\ GN$. Acrylic is much softer than alloy so we can see that there is much more deformation occur on the acrylic material, as shown in Fig. 29. This high deformation also affects values of stress components. The lower value of Young's modulus make the stress components become smaller than that of aluminum. Fig. 30 shows us the higher intense of color of stress, which mean that stress component inside alloy body is higher than acrylic.

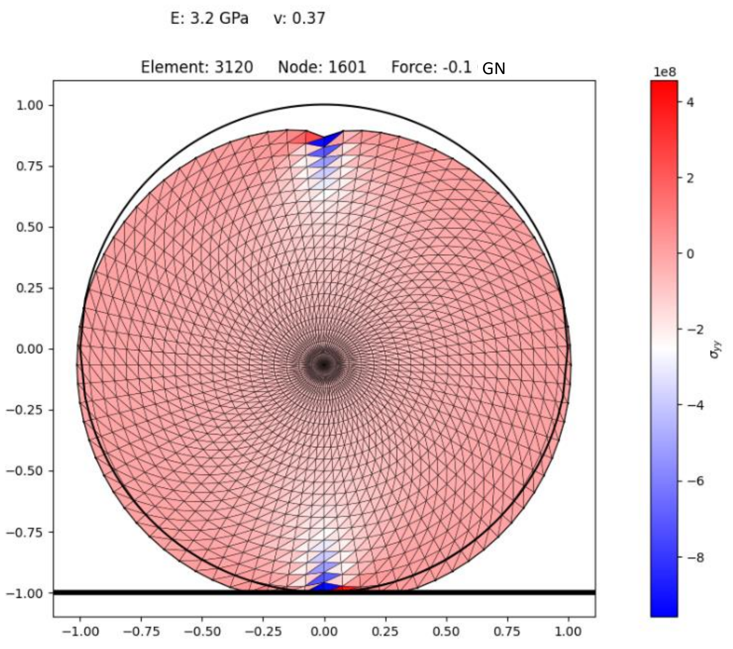


Figure 29. Deformation and σ_{yy} distribution of acrylic.

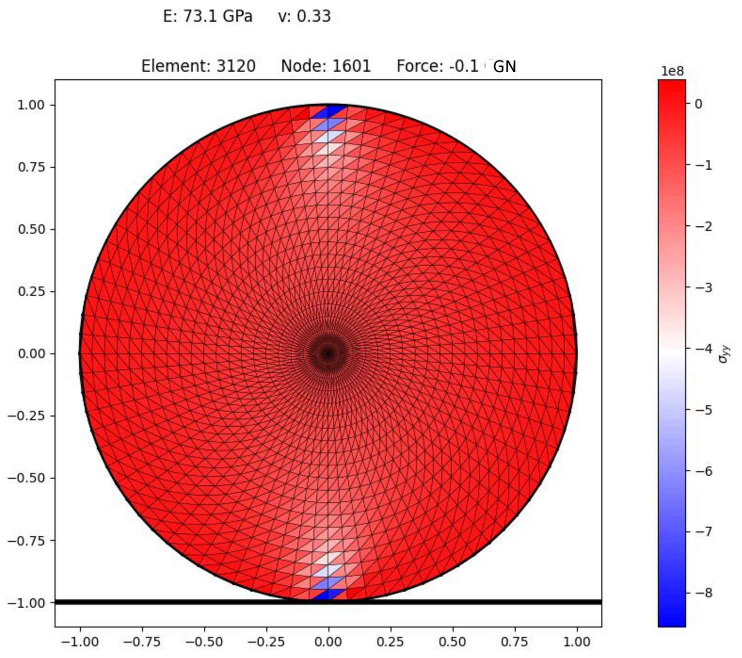


Figure 30. Deformation and σ_{yy} distribution of alloy 2024 T4.