ED5015 - COMPUTATIONAL METHODS IN DESIGN

MINI PROJECT 1

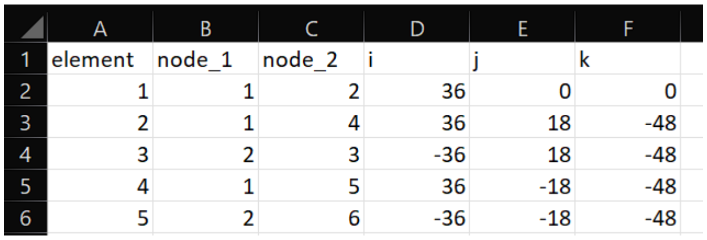
IMPORTING LIBRARIES AND GETTING DATA:

First, we would like to import all the necessary libraries like pandas,numpy,...

```
%matplotlib notebook

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
```

%matplotlib notebook enables to use functions like zoom and rotate in the notebook
pandas is used to read data from a csv and save it as a dataframe
numpy is used to make n-dimensional matrices and do functions like inverse and dot-product
axes3d is used to visualize the graph in 3D

A csv file which contains the vector forms of every element is made. This csv file looks like this,



	A	B	C	D	E	F
1	element	node_1	node_2	i	j	k
2	1	1	2	36	0	0
3	2	1	4	36	18	-48
4	3	2	3	-36	18	-48
5	4	1	5	36	-18	-48
6	5	2	6	-36	-18	-48

Then the data from a csv file and make a dataframe using ***pandas*** and set the index column to be ***‘element’.***

```
data = pd.read_csv("MP1_Data_conv.csv", index_col="element")
print(data)
```

A sample of the data in the dataframe is shown below.

```
         node_1  node_2   i   j   k
element
1             1       2  36   0   0
2             1       4  36  18 -48
3             2       3 -36  18 -48
4             1       5  36 -18 -48
```

Setting some default parameters given in the question.

```
E=3e7                           # Young's modulus
A=np.pi                         # Cross-Sectional Area
poisson = 0.3                   # Poisson's ratio
mass_density = 7.3e-4
```

CALCULATING THE LOCAL STIFFNESS MATRIX:

To calculate the local stiffness matrix, we need the find the angle of the local element wrt. the global coordinate system. We can also find the length and mass of each element at the same time and add everything to the dataframe.

```
data['theta_x']=np.arccos(data['i']/(data['i']**2 + data['j']**2 + data['k']**2 )**0.5)
data['theta_y']=np.arccos(data['j']/(data['i']**2 + data['j']**2 + data['k']**2 )**0.5)
data['theta_z']=np.arccos(data['k']/(data['i']**2 + data['j']**2 + data['k']**2 )**0.5)
data['length']=(data['i']**2 + data['j']**2 + data['k']**2)**0.5
data['mass']=data['length']*np.pi*mass_density
```

After calculating these values, we can make a local K matrix using a user defined function.

```
def localK(element):
    _,_,i,j,k,theta_x,theta_y,theta_z,L,_ = data.loc[element].values.tolist()
    l=np.cos(theta_x)
    m=np.cos(theta_y)
    n=np.cos(theta_z)
    trans = np.array([[l**2, l*m, l*n, -1*(l**2), -1*(l*m), -1*(l*n)],
                      [l*m, m**2, m*n, -1*(l*m), -1*(m**2), -1*(n*m)],
                      [l*n, m*n, n**2, -1*(l*n), -1*(n*m), -1*(n**2)],
                      [-1*(l**2), -1*(l*m), -1*(l*n), l**2, l*m, l*n],
                      [-1*(l*m), -1*(m**2), -1*(m*n), l*m, m**2, n*m],
                      [-1*(l*n), -1*(m*n), -1*(n**2), l*n, n*m, n**2]])
    trans[abs(trans)<0.01]=0
    k=np.dot(((E*A)/L*(1+2*(poisson**2))), trans)

    return k
```

This function would take values from the dataframe and calculate the local stiffness matrix of the element when called.

CALCULATING THE GLOBAL STIFFNESS MATRIX:

We can calculate the global stiffness matrix from the local stiffness matrix of all the elements. We do know the start and end nodes of each element from the dataframe. So, finding the nodes of the element in global and adding the local part would result in the final global stiffness matrix.

```
def globalK():
    gk=np.zeros([30,30])
    for element in range(1,26):
        node_1,node_2,_,_,_,_,_,_,_,_ = data.loc[element].values.tolist()
        node_1=int(node_1)
        node_2=int(node_2)
        lk = localK(element)
        for i in range(3):
            for j in range(3):
                gk[(node_1-1)*3+i][(node_1-1)*3+j] += lk[i][j]
                gk[(node_1-1)*3+i][(node_2-1)*3+j] += lk[i][j+3]
                gk[(node_2-1)*3+i][(node_1-1)*3+j] += lk[i+3][j]
                gk[(node_2-1)*3+i][(node_2-1)*3+j] += lk[i+3][j+3]

    return gk
```

If we print the global K matrix, we would get something similar to,

```
print(globalK())
```

```
[[ 3.06086871e+06 -8.73114914e-11 -1.12297311e+06 -2.21863888e+06
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00 -4.21114915e+05 -2.10557458e+05  5.61486553e+05
  -4.21114915e+05  2.10557458e+05  5.61486553e+05  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-8.73114914e-11  5.94729790e+05 -5.82076609e-11  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.92086166e+05
   5.12229777e+05 -2.10557458e+05 -1.05278729e+05  2.80743277e+05
   2.10557458e+05 -1.05278729e+05 -2.80743277e+05  0.00000000e+00
  -1.92086166e+05 -5.12229777e+05  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-1.12297311e+06 -5.82076609e-11  4.22918962e+06  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.12229777e+05
  -1.36594607e+06  5.61486553e+05  2.80743277e+05 -7.48648738e+05
```

APPLYING THE BOUNDARY CONDITIONS:

As the nodes 7,8,9 and 10 are fixed, they don't have displacements and we can eliminate those rows from the Nodal Displacement matrix and the Force matrix. And to make the equation work, we match the orders and reduce the Global Stiffness matrix to a 18X18 matrix.

```
# We need to eliminate rows and columns for nodes 7,8,9 and 10
K=globalK()
for i in range(18,30):
    K = np.delete(K, [18], 0)
    K = np.delete(K, [18], 1)
```

The Force matrix has two different causes. First is the given condition to deform. Second is due to the weight of elements.

```
#Given F
F=np.zeros([18,1])
F[1]=F[4]=60000

#Forces due to weight
F[2]=F[5]=  -1*data.iloc[0:1]['mass'].sum()*9.81/2
F[8]=F[11]=F[14]=F[17]= -1*data.iloc[0:13]['mass'].sum()*9.81/4
print(F)
```

The weight of elements can be calculated from the mass in the dataframe, multiplying it with acceleration due to gravity and dividing it between the nodes on that plane.

A sample of the force matrix is shown below.

```
[[ 0.00000000e+00]
 [ 6.00000000e+04]
 [-4.04961974e-01]
 [ 0.00000000e+00]
 [ 6.00000000e+04]
 [-4.04961974e-01]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [-3.57504619e+00]
```

NODAL DISPLACEMENT MATRIX:

The Nodal Displacement matrix can be calculated by multiplying the inverse of the Global Stiffness matrix with the Force matrix. The inverse of the GlobalK matrix can be calculated from the ***numpy*** library using the function ***np.linalg.inv()*** .

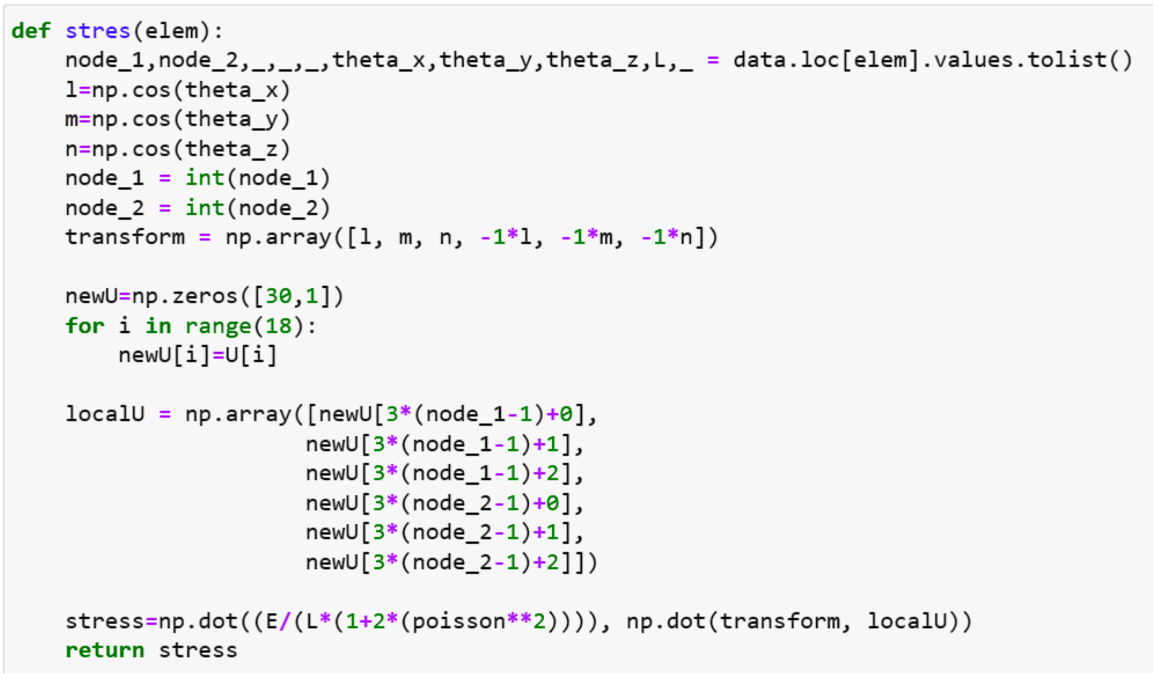
```
U = np.dot(np.linalg.inv(K), F)
print(U)
```

This will give a U matrix like,

```
[[-3.64755546e-08]
 [ 2.80241767e-01]
 [-3.45374254e-06]
 [ 3.64755545e-08]
 [ 2.80241767e-01]
 [-3.45374254e-06]
 [-2.04013586e-03]
 [ 1.84408931e-02]
```

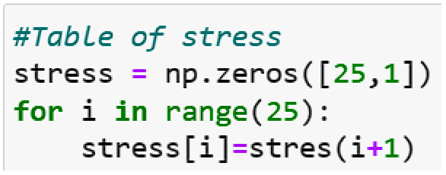
STRESS MATRIX:

To find the stress in each element, we can define a function that takes the element, finds the local nodal displacement matrix and uses it to find the stress in it.



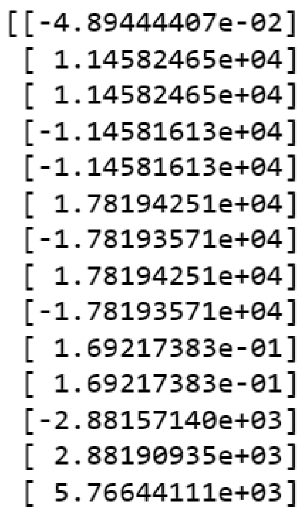
```python
def stres(elem):
    node_1,node_2,_,_,_,theta_x,theta_y,theta_z,L,_ = data.loc[elem].values.tolist()
    l=np.cos(theta_x)
    m=np.cos(theta_y)
    n=np.cos(theta_z)
    node_1 = int(node_1)
    node_2 = int(node_2)
    transform = np.array([l, m, n, -1*l, -1*m, -1*n])

    newU=np.zeros([30,1])
    for i in range(18):
        newU[i]=U[i]

    localU = np.array([newU[3*(node_1-1)+0],
                       newU[3*(node_1-1)+1],
                       newU[3*(node_1-1)+2],
                       newU[3*(node_2-1)+0],
                       newU[3*(node_2-1)+1],
                       newU[3*(node_2-1)+2]])

    stress=np.dot((E/(L*(1+2*(poisson**2)))), np.dot(transform, localU))
    return stress
```

Then we can calculate the stress in each element with this function and append it in a table.



```python
#Table of stress
stress = np.zeros([25,1])
for i in range(25):
    stress[i]=stres(i+1)
```

So, if we print the stress values it will be similar to this.



```
[[-4.89444407e-02]
 [ 1.14582465e+04]
 [ 1.14582465e+04]
 [-1.14581613e+04]
 [-1.14581613e+04]
 [ 1.78194251e+04]
 [-1.78193571e+04]
 [ 1.78194251e+04]
 [-1.78193571e+04]
 [ 1.69217383e-01]
 [ 1.69217383e-01]
 [-2.88157140e+03]
 [ 2.88190935e+03]
 [ 5.76644111e+03]
```

FINDING THE COORDINATES OF NODES:

Using the data in the dataframe, And assuming the first node is at the point ***P(0,0,0)*** , We can find the location of all the other nodes. For every element, we need to add the vectors to the start node coordinates to find the end node coordinates.

```
nodes_coords=np.zeros([10,3])
done=[1]
for element in range(1,26):
    a,b,i,j,k,_,_,_,_,_ = data.loc[element].values.tolist()
    if b not in done and a in done:
        nodes_coords[int(b)-1]=nodes_coords[int(a)-1]+[i,j,k]
        done.append(b)
```

Now, to find the node coordinates after deformation, we need to add the Nodal Displacement matrix to the original Node coordinates.

```
def_nodes_coords = nodes_coords.copy()
for i in range(6):
    def_nodes_coords[i] = nodes_coords[i] + np.reshape(U[(3*i):(3*i+3)], (1,3))
print(def_nodes_coords)
```

The final Deformed Nodal Coordinates will be similar to,

```
[[-3.64755546e-08  2.80241767e-01 -3.45374254e-06]
 [ 3.60000000e+01  2.80241767e-01 -3.45374254e-06]
 [-2.04013586e-03  1.80184409e+01 -4.80598047e+01]
 [ 3.60020401e+01  1.80184409e+01 -4.80598047e+01]
 [ 3.59979596e+01 -1.79815588e+01 -4.79402022e+01]
 [ 2.04040851e-03 -1.79815588e+01 -4.79402022e+01]
 [-3.00000000e+01  4.80000000e+01 -9.60000000e+01]
 [ 6.60000000e+01  4.80000000e+01 -9.60000000e+01]
 [ 6.60000000e+01 -4.80000000e+01 -9.60000000e+01]
 [-3.00000000e+01 -4.80000000e+01 -9.60000000e+01]]
```

PLOTTING THE GRAPH:

To plot a graph, we first need to make a figure using ***plt.figure()***. Then we need to create an axis. The axis should be made in such a way that it should have a projection of 3D.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

Now we can add the nodes in the graph as points. To do this we can use a function called ***scatter().*** We need to give the x,y,z coordinates of the nodes and also a color.

```
ax.scatter(nodes_coords[:,0], nodes_coords[:,1], nodes_coords[:,2], alpha=1, color='#8956d1')
```

Then, we need to plot the lines for elements. We use the data from the dataframe to find the nodes of each element and plot lines between them.

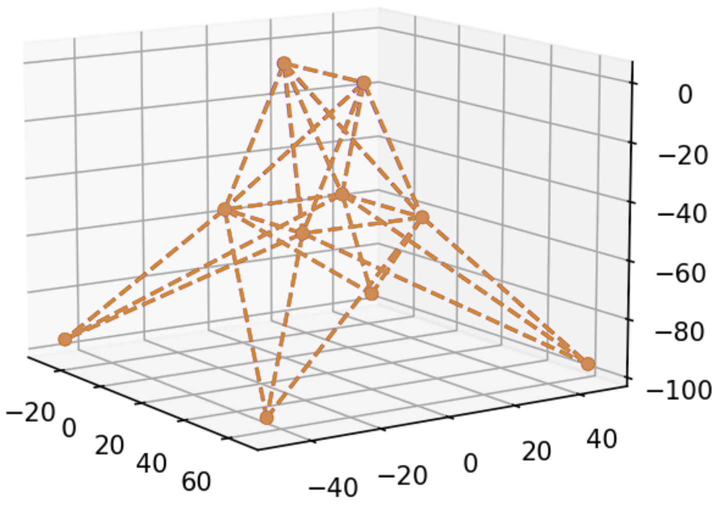
```
for element in range(1,26):
    a,b,_,_,_,_,_,_,_,_ = data.loc[element].values.tolist()
    x = [nodes_coords[int(a)-1,0], nodes_coords[int(b)-1,0]]
    y = [nodes_coords[int(a)-1,1], nodes_coords[int(b)-1,1]]
    z = [nodes_coords[int(a)-1,2], nodes_coords[int(b)-1,2]]
    ax.plot(x, y, z, linestyle="--", color='#8956d1')
```

Now, similarly for the Deformed Nodes, we only need to change the color and coordinates.

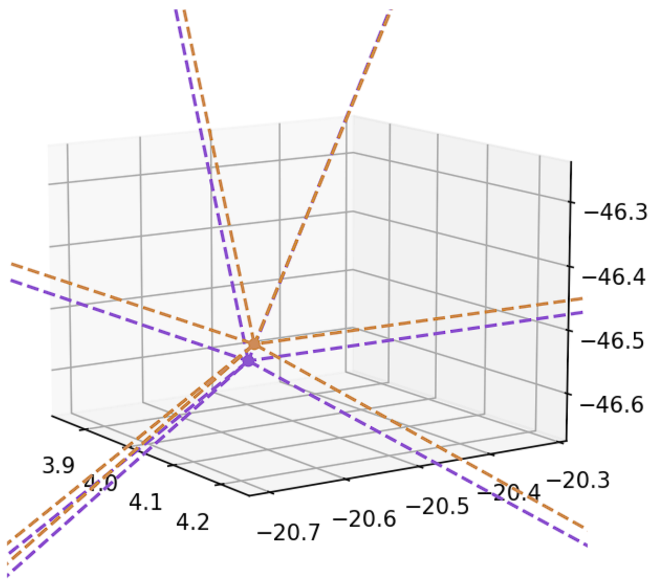
```
ax.scatter(def_nodes_coords[:,0], def_nodes_coords[:,1], def_nodes_coords[:,2], alpha=1, color="#d18b56")
```

```
for element in range(1,26):
    a,b,_,_,_,_,_,_,_,_ = data.loc[element].values.tolist()
    x = [def_nodes_coords[int(a)-1,0], def_nodes_coords[int(b)-1,0]]
    y = [def_nodes_coords[int(a)-1,1], def_nodes_coords[int(b)-1,1]]
    z = [def_nodes_coords[int(a)-1,2], def_nodes_coords[int(b)-1,2]]
    ax.plot(x, y, z, linestyle="--", color='#d18b56')

plt.show()
```

We finally end with a ***plt.show()***, which displays the plot.



Here, we can only see a single plot, as both the node coordinates before and after deformation are mostly the same. Their difference is in the order of -2. So we need to zoom in if we want to see the deformation.



This plot shows the deformation of the 6th node. We can clearly see how the node has moved due to these forces.

Thirumalai Jeyabalaji

ED20B070