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


#PARAMETERS
a = 0.2            # [m] Square cross section
L = 2              # [m] Length of the beam
E = 75e9           # [Pa] Young's Modulus
v = 0.33           # Poissons Ratio
G = E/(2*(1+v))
First  = E*(1-v)/((1+v)*(1-2*v))
Second = v*E/((1+v)*(1-2*v))
C_11 = First
C_22 = First
C_33 = First
C_12 = Second
C_13 = Second
C_23 = Second
C_44 = G
C_55 = G
C_66 = G


#Coordinates of the cross section
X1 = -0.1
Z1 = -0.1
X2 =  0.1
Z2 = -0.1
X3 =  0.1
Z3 =  0.1
X4 = -0.1
Z4 =  0.1


#Along the beam axis(Y)
n_elem = 1                                                # No of elements
per_elem = 2                                              # Type of the element
n_nodes  = (per_elem-1)*n_elem  + 1                       # Total no of nodes
xi =   np.array([0.57735,-0.57735])                       # Gauss points
W_Length   = 1                                            # Weight for gauss quadrature
Shape_func = np.array([1/2*(1-xi),1/2*(1+xi)])            # Shape functions
```

```
N_Der_xi = np.array([-1/2,1/2])                              # Derivative of the shape function (N,xi)
X_coor     = np.array([0,L])
J_Length   = N_Der_xi@np.transpose(X_coor)                                  #Jacobian for the length of the
N_Der      = np.array([-1/2*(1/J_Length),1/2*(1/J_Length)])            #Derivative of the shape functi


#Along the Beam cross section (X,Z)
#Lagrange polynomials
alpha = np.array([0.57735,0.57735,-0.57735,-0.57735])                           # Gauss points
beta  = np.array([0.57735,-0.57735,0.57735,-0.57735])
W_Cs  = 1                                                                        # weight for gauss quadrature
Lag_poly = np.array([1/4*(1-alpha)*(1-beta),1/4*(1+alpha)*(1-beta),1/4*(1+alpha)*(1+beta),1/4*(1-alpha)
n_cross_nodes = 4
DOF = 3

#Lagrange Derivatives
alpha_der = np.array([-1/4*(1-beta),1/4*(1-beta),1/4*(1+beta),-1/4*(1+beta)])         # Derivatives of
beta_der  = np.array([-1/4*(1-alpha),-1/4*(1+alpha),1/4*(1+alpha),1/4*(1-alpha)])     # with respect to

X_alpha = alpha_der[0]*X1 + alpha_der[1]*X2 + alpha_der[2]*X3 + alpha_der[3]*X4
X_beta  = beta_der[0] *X1 + beta_der[1]*X2  + beta_der[2] *X3 + beta_der[3] *X4
Z_alpha = alpha_der[0]*Z1 + alpha_der[1]*Z2 + alpha_der[2]*Z3 + alpha_der[3]*Z4
Z_beta  = beta_der[0] *Z1 + beta_der[1]*Z2  + beta_der[2] *Z3 + beta_der[3] *Z4

J_Cs = (Z_beta*X_alpha - Z_alpha*X_beta)   # Determinant of Jacobian matrix of the cross section
J_Cs = np.unique(J_Cs)



Elemental_stiffness_matrix = np.zeros((per_elem*n_cross_nodes*DOF,per_elem*n_cross_nodes*DOF))
sep = int((per_elem*n_cross_nodes*DOF)/per_elem)      # Seperation point for stacking element stiffness


for i in range(len(Shape_func)):
    for j in range(len(Shape_func)):
        #Fundamental nucleus of the stiffness matrix K_tsij using two point gauss quadrature
        Nodal_stiffness_matrix = np.zeros((n_cross_nodes*DOF,n_cross_nodes*DOF))
        for tau in range(n_cross_nodes):
            for s in range(n_cross_nodes):

                #Fundamental nucleus of the stiffness matrix
                #Derivative of F wrt to x and z for tau
```

```
                F_tau_x = 1/J_Cs*((Z_beta*alpha_der[tau])-(Z_alpha*beta_der[tau]))
                F_tau_z = 1/J_Cs*((-X_alpha*alpha_der[tau])+(X_beta*beta_der[tau]))

                #Derivative of F wrt to x and z for s
                F_s_x = 1/J_Cs*((Z_beta*alpha_der[s])-(Z_alpha*beta_der[s]))
                F_s_z = 1/J_Cs*((-X_alpha*alpha_der[s])+(X_beta*beta_der[s]))



                K_xx =  C_22*np.sum(W_Cs*F_tau_x*F_s_x*J_Cs)*np.sum(W_Length*Shape_func[i]*Shape_func[j
                K_xy =  C_23*np.sum(W_Cs*Lag_poly[tau]*F_s_x*J_Cs)*np.sum(W_Length*N_Der[i]*Shape_func[
                K_xz =  C_12*np.sum(W_Cs*F_tau_z*F_s_x*J_Cs)*np.sum(W_Length*Shape_func[i]*Shape_func[j
                K_yx =  C_44*np.sum(W_Cs*Lag_poly[tau]*F_s_x*J_Cs)*np.sum(W_Length*N_Der[i]*Shape_func[
                K_yy =  C_55*np.sum(W_Cs*F_tau_z*F_s_z*J_Cs)*np.sum(W_Length*Shape_func[i]*Shape_func[j
                K_yz =  C_55*np.sum(W_Cs*Lag_poly[tau]*F_s_z*J_Cs)*np.sum(W_Length*N_Der[i]*Shape_func[
                K_zx =  C_12*np.sum(W_Cs*F_tau_x*F_s_z*J_Cs)*np.sum(W_Length*Shape_func[i]*Shape_func[j
                K_zy =  C_13*np.sum(W_Cs*Lag_poly[tau]*F_s_z*J_Cs)*np.sum(W_Length*N_Der[i]*Shape_func[
                K_zz =  C_11*np.sum(W_Cs*F_tau_z*F_s_z*J_Cs)*np.sum(W_Length*Shape_func[i]*Shape_func[j
                F_Nu = np.array([[K_xx,K_xy,K_xz],[K_yx,K_yy,K_yz],[K_zx,K_zy,K_zz]])


                Nodal_stiffness_matrix[3*s:3*(s+1) , 3*tau:3*(tau+1)]=F_Nu



        Elemental_stiffness_matrix[sep*j:sep*(j+1) , sep*i:sep*(i+1)]=Nodal_stiffness_matrix




Load_vector = np.zeros((n_nodes*n_cross_nodes*DOF,1))
Load_vector[n_nodes*n_cross_nodes*DOF-10] = -12.5
Load_vector[n_nodes*n_cross_nodes*DOF-7]  = -12.5
Load_vector[n_nodes*n_cross_nodes*DOF-4]  = -12.5
Load_vector[n_nodes*n_cross_nodes*DOF-1]  = -12.5



Displacement = np.linalg.solve(Elemental_stiffness_matrix[12:,12:],Load_vector[12:])
print("Displacement----------------------------------------------")
print(Displacement)

```

```
126
```

```
Displacement-------------------------------------------------
[[-2.67316803e-07]
 [-1.33658401e-08]
 [ 8.99834692e-08]
 [-2.67316803e-07]
 [ 1.33658401e-08]
 [ 8.99834692e-08]
 [-2.67316803e-07]
 [ 1.33658401e-08]
 [ 8.99834692e-08]
 [-2.67316803e-07]
 [-1.33658401e-08]
 [ 8.99834692e-08]]
```

In [ ]: