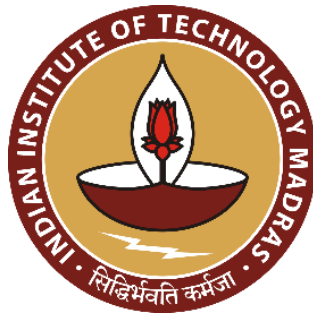


ASSIGNMENT
ME 5204 – FINITE ELEMENT ANALYSIS



Department of Mechanical Engineering
Machine Design Section
Indian Institute of Technology, Madras - 600036

Submitted By,
N. SAI TEJA
ME22S084

Problem Statement 1:

For a given 2-Dimensional structure as shown in figure 1, evaluate the displacements at each point and also evaluate the stresses, following a complete finite element procedure using python.

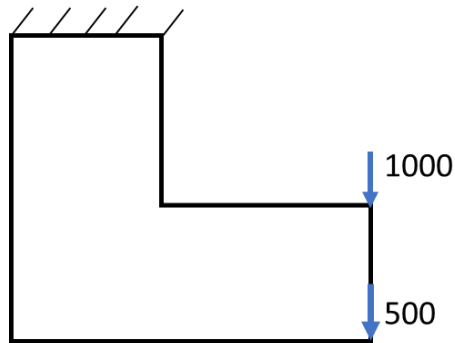


Fig. 1: 2-D Structure

Solution Methodology:

1. Pre-Processing:

i. Discretization: Dividing the given structure into finite element entities i.e., nodes and elements as shown in figure 2.

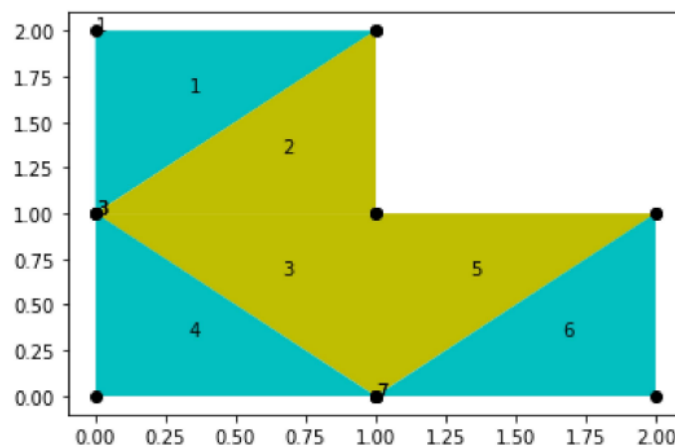


Fig. 2: FEM Model

The given structure is divided into 6 Constant Strain Triangular (CST) elements and 8 nodes.

The structure is made up of two different kinds of elements, which is indicated in the figure with two different colors.

The following text documents are created for the extraction of the data of the structure.

1. COORD
2. NCA
3. MAT
4. LOAD_BC
5. DISP_BC

The first two documents, i.e., 'COORD' and 'NCA' contains the data related to the geometry of the structure, i.e., it contains information about the nodal coordinates and the element nodal connection array. 'MAT' contains information about the material properties i.e., it contains values of Young's modulus and poison's ratio. 'LOAD_BC' & 'DISP_BC' contains information about the external loads applied and supports given to the structure.

The figure below shows the data in the text document:

#ele	n1	n2	n3	mat
0	0	0	0	0
1	1	3	2	1
2	3	4	2	2
3	3	7	4	2
4	3	6	7	1
5	7	5	4	2
6	7	8	5	1

Fig. 3: Nodal Connection Array (NCA)

Following shows the flowchart for solving a Finite Element Problem:

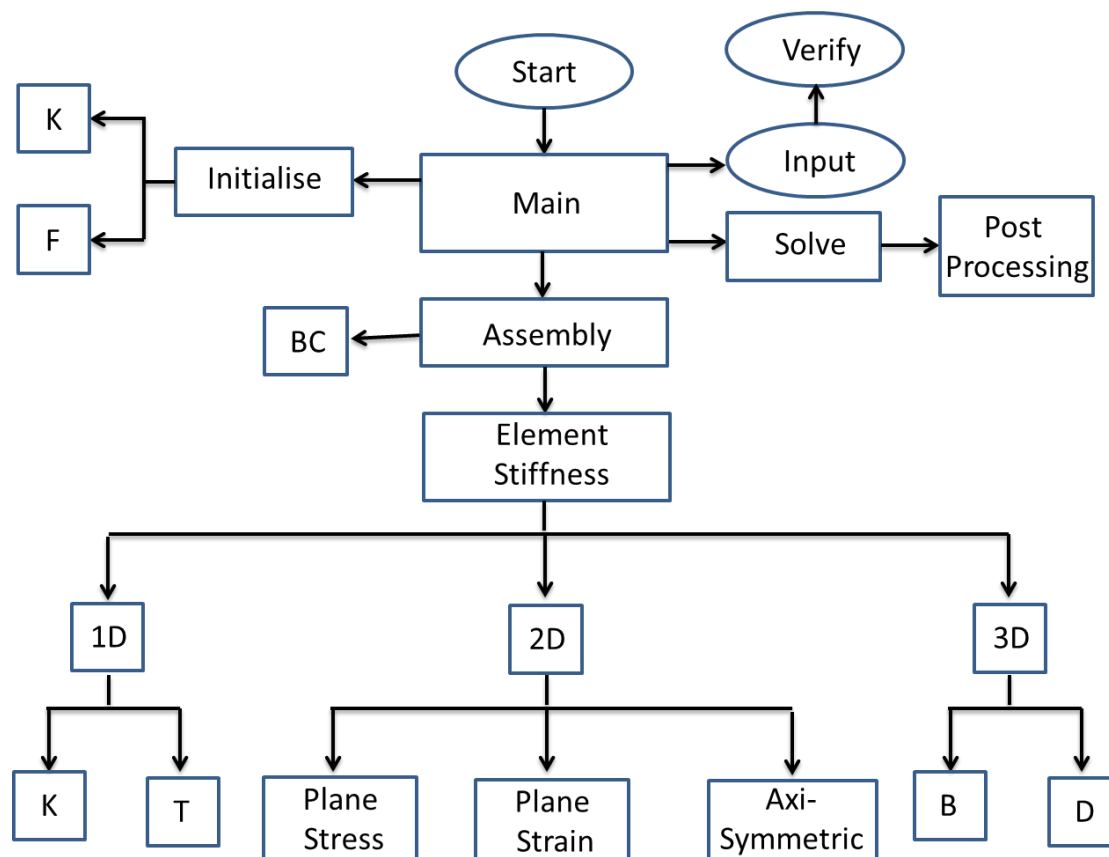


Fig. 4: Flow Chart

The following shows the implemented python code for solving of the given 2D structure:

Problem Statement 2:

Using Galerkin approximation solve for approximate solution of '**sin(x)**'. Use complete finite element procedure and make use of monomial bases.

The element stiffness matrix can be calculated using:

$$K_{ij} = \int b_i b_j d\omega$$

The force matrix can be calculated using:

$$F_i = \int f b_i d\omega$$

Where, $f = \sin(x)$

The following is the implemented Python code for the approximate solution of "**sin(x)**":

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import pandas as pd
```

```
In [2]: num_nodes = 8
num_elements = 6 #cst element
num_materials = 2
problem_type = 21 #plane stress
thickness = 0.5
num_load_bc = 2 #number of loads applied
num_disp_bc = 2 #number of places displacement condition is applied
dofpn = 2 #degrees of freedom per node
tdof = num_nodes*dofpn #total degrees of freedom
```

```
In [3]: COORD = np.loadtxt(fname = 'D:\\SAI TEJA\\DATA\\COORD.txt').astype(np.float32)
NCA = np.loadtxt(fname = 'D:\\SAI TEJA\\DATA\\NCA.txt').astype(np.int64)
MAT = np.loadtxt(fname = 'D:\\SAI TEJA\\DATA\\MAT.txt').astype(np.float32)
LOAD_BC = np.loadtxt(fname = 'D:\\SAI TEJA\\DATA\\LOAD_BC.txt').astype(np.float32)
DISP_BC = np.loadtxt(fname = 'D:\\SAI TEJA\\DATA\\DISP_BC.txt').astype(np.float32)
#print(np.shape(NCA))
#print(COORD[0][1])
#N1 = NCA[1][1]
#print(N1)
```

```
In [4]: for ele in range(1,num_elements+1,1):

    # Nodes for each element from 1 to 6

    N1=NCA[ele,1]
    N2=NCA[ele,2]
    N3=NCA[ele,3]

    # Nodal coordinates for each element from 1 to 6

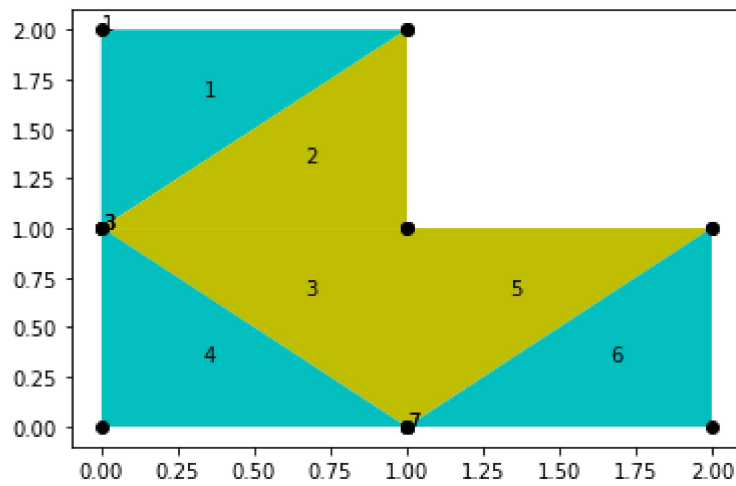
    X1N1=COORD[N1,1]
    X2N1=COORD[N1,2]
    X1N2=COORD[N2,1]
    X2N2=COORD[N2,2]
    X1N3=COORD[N3,1]
    X2N3=COORD[N3,2]

    #Assigning material for each element

    Mat_num=NCA[ele,4]

    #Verification of the geometry of the problem by plotting all the elements and

    X=[X1N1,X1N2,X1N3]
    Y=[X2N1,X2N2,X2N3]
    CGX=(X1N1+X1N2+X1N3)/3
    CGY=(X2N1+X2N2+X2N3)/3
    if Mat_num==1:
        plt.fill(X,Y,'c')
    else:
        plt.fill(X,Y,'y')
    plt.scatter(X,Y,c='black')
    plt.text(CGX,CGY,str(ele))
    plt.text(X1N1,X2N1,str(N1))
```



```
In [5]: GSTIFF=np.zeros((tdof,tdof)) # initialisation of Global stiffness matrix with all zeros
        F=np.zeros(tdof) # initialisation of force matrix
```

```
In [6]: for ele in range(1,num_elements+1,1):

        # Nodes for each element from 1 to 6

        N1=NCA[ele,1]
        N2=NCA[ele,2]
        N3=NCA[ele,3]

        # Nodal coordinates for each element from 1 to 6

        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]]) #creating 3x3 matrix
        two_delta = np.linalg.det(two_delta_matrix) #calculating det of the matrix

        #calculation of B matrix

        B1 = X2N2-X2N3 #calculation of Beta values
        B2 = X2N3-X2N1
        B3 = X2N1-X2N2
        G1 = X1N3-X1N2 #calculation of Gamma values
        G2 = X1N1-X1N3
        G3 = X1N2-X1N1
        B = np.zeros((3,6)) #initialisation of B matrix to Zero
        B[0,0] = B1
        B[0,2] = B2
        B[0,4] = B3
        B[1,1] = G1
        B[1,3] = G2
        B[1,5] = G3
        B[2,0] = G1
        B[2,1] = B1
        B[2,2] = G2
        B[2,3] = B2
        B[2,4] = G3
        B[2,5] = B3
        B=B/two_delta #calculation of B matrix

        Mat_num=NCA[ele,4]
```

```

E = MAT[Mat_num,1]
PR = MAT[Mat_num,2]
CONST = E/(1-PR**2)
D = np.zeros((3,3))
D[0,0] = 1
D[0,1] = PR
D[1,0] = PR
D[1,1] = 1
D[2,2] = (1-PR)/2
D=D*CONST

ESTIFF=B.transpose()@D@B*thickness*two_delta*0.5 # Element stiffness matrix

#Assembly of Global Stiffness Matrix

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)
RN_IDX=CN_IDX.transpose()
GSTIFF[RN_IDX,CN_IDX]=GSTIFF[RN_IDX,CN_IDX]+ESTIFF

```

```

In [7]: for i in range (1,num_load_bc+1,1):
        load_type = LOAD_BC[i,2]
        if load_type==1:
            N=int(LOAD_BC[i,1])
            F[(2*N-2)]=F[(2*N-2)]+LOAD_BC[i,3] #Assembly of Global Force Vector
        elif load_type==2:
            N=int(LOAD_BC[i,1])
            F[(2*N-1)]=F[(2*N-1)]+LOAD_BC[i,4]
        else:
            N=int(LOAD_BC[i,1])
            F[(2*N-2)]=F[(2*N-2)]+LOAD_BC[i,3]
            F[(2*N-1)]=F[(2*N-1)]+LOAD_BC[i,4]

```

```

In [8]: GSTIFFCOPY=GSTIFF.copy() #Creating a copy of Global Stiffness matrix for solving

```

```

In [9]: for i in range (1,num_disp_bc+1,1):
        disp_type = DISP_BC[i,2]
        if disp_type==1:
            N=int(DISP_BC[i,1])
            F[(2*N-2)]=F[(2*N-2)]+(DISP_BC[i,3]*10**16)
            GSTIFFCOPY[2*N-2,2*N-2]=GSTIFFCOPY[2*N-2,2*N-2]+10**16
        elif disp_type==2:
            N=int(DISP_BC[i,1])
            F[(2*N-1)]=F[(2*N-1)]+(DISP_BC[i,4]*10**16)
            GSTIFFCOPY[2*N-1,2*N-1]=GSTIFFCOPY[2*N-1,2*N-1]+10**16
        else:
            N=int(DISP_BC[i,1])
            F[(2*N-2)]=F[(2*N-2)]+(DISP_BC[i,3]*10**16)
            GSTIFFCOPY[2*N-2,2*N-2]=GSTIFFCOPY[2*N-2,2*N-2]+10**16
            F[(2*N-1)]=F[(2*N-1)]+(DISP_BC[i,4]*10**16)
            GSTIFFCOPY[2*N-1,2*N-1]=GSTIFFCOPY[2*N-1,2*N-1]+10**16

```

```

In [10]: DISP=np.linalg.solve(GSTIFFCOPY,F)
         print(DISP.reshape(-1,2))

```

```
[[-8.97677140e-14  1.50000000e-13]
 [ 8.97677140e-14 -3.00000000e-13]
 [-2.77845775e-08  1.68650109e-08]
 [-3.29055550e-08 -3.25787519e-08]
 [-2.91215096e-08 -1.53677432e-07]
 [-1.12493656e-07  2.21209219e-08]
 [-1.17749567e-07 -3.80605723e-08]
 [-1.26697888e-07 -1.58729112e-07]]
```

```
In [11]: # calculation of stress and strain
strain=np.empty(shape=(3,0))
stress=np.empty(shape=(3,0))
```

```
In [12]: for ele in range(1,num_elements+1,1):

    # Nodes for each element from 1 to 6

    N1=NCA[ele,1]
    N2=NCA[ele,2]
    N3=NCA[ele,3]

    # Nodal coordinates for each element from 1 to 6

    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]]) #creation of two_delta_matrix
    two_delta = np.linalg.det(two_delta_matrix) #calculating det of the matrix

    #calculation of B matrix

    B1 = X2N2-X2N3 #calculation of Beta values
    B2 = X2N3-X2N1
    B3 = X2N1-X2N2
    G1 = X1N3-X1N2 #calculation of Gamma values
    G2 = X1N1-X1N3
    G3 = X1N2-X1N1
    B = np.zeros((3,6)) #initialisation of B matrix to Zero
    B[0,0] = B1
    B[0,2] = B2
    B[0,4] = B3
    B[1,1] = G1
    B[1,3] = G2
    B[1,5] = G3
    B[2,0] = G1
    B[2,1] = B1
    B[2,2] = G2
    B[2,3] = B2
    B[2,4] = G3
    B[2,5] = B3
    B=B/two_delta #calculation of B matrix
    print('B matrix for element',end=" ")
    print(ele)
    print(B)

    Mat_num=NCA[ele,4]
    E = MAT[Mat_num,1]
    PR = MAT[Mat_num,2]
    CONST = E/(1-PR**2)
```



```

D = np.zeros((3,3))
D[0,0] = 1
D[0,1] = PR
D[1,0] = PR
D[1,1] = 1
D[2,2] = (1-PR)/2
D=D*CONST
print('D matrix for element',end=" ")
print(ele)
print(D)

u=np.array([DISP[2*N1-2],DISP[2*N1-1],DISP[2*N2-2],DISP[2*N2-1],DISP[2*N3-2],DISP[2*N3-1]])
print('u matrix for element',end=" ")
print(ele)
print(u)

strain=np.concatenate([strain,np.array((B@u)).reshape(3,1)],axis=1)
stress=np.concatenate([stress,np.array((D@B@u)).reshape(3,1)],axis=1)

```

```

B matrix for element 1
[[-1.  0.  0.  0.  1.  0.]
 [ 0.  1.  0. -1.  0.  0.]
 [ 1. -1. -1.  0.  0.  1.]]
D matrix for element 1
[[2.38095235e+11 9.52380952e+10 0.00000000e+00]
 [9.52380952e+10 2.38095235e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 7.14285697e+10]]
u matrix for element 1
[[-8.97677140e-14]
 [ 1.50000000e-13]
 [-2.77845775e-08]
 [ 1.68650109e-08]
 [ 8.97677140e-14]
 [-3.00000000e-13]]
B matrix for element 2
[[-1.  0.  1.  0.  0.  0.]
 [ 0.  0.  0. -1.  0.  1.]
 [ 0. -1. -1.  1.  1.  0.]]
D matrix for element 2
[[3.19999985e+11 7.99999962e+10 0.00000000e+00]
 [7.99999962e+10 3.19999985e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.19999994e+11]]
u matrix for element 2
[[-2.77845775e-08]
 [ 1.68650109e-08]
 [-3.29055550e-08]
 [-3.25787519e-08]
 [ 8.97677140e-14]
 [-3.00000000e-13]]
B matrix for element 3
[[-1.  0.  0.  0.  1.  0.]
 [ 0.  0.  0. -1.  0.  1.]
 [ 0. -1. -1.  0.  1.  1.]]
D matrix for element 3
[[3.19999985e+11 7.99999962e+10 0.00000000e+00]
 [7.99999962e+10 3.19999985e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.19999994e+11]]
u matrix for element 3
[[-2.77845775e-08]
 [ 1.68650109e-08]
 [-1.17749567e-07]
 [-3.80605723e-08]
 [-3.29055550e-08]
 [-3.25787519e-08]]
B matrix for element 4
[[ 0.  0. -1.  0.  1.  0.]
 [ 0.  1.  0. -1.  0.  0.]
 [ 1.  0. -1. -1.  0.  1.]]
D matrix for element 4
[[2.38095235e+11 9.52380952e+10 0.00000000e+00]
 [9.52380952e+10 2.38095235e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 7.14285697e+10]]
u matrix for element 4
[[-2.77845775e-08]
 [ 1.68650109e-08]
 [-1.12493656e-07]
 [ 2.21209219e-08]
 [-1.17749567e-07]
 [-3.80605723e-08]]
B matrix for element 5
[[ 0.  0.  1.  0. -1.  0.]
 [ 0. -1.  0.  0.  0.  1.]
 [-1.  0.  0.  1.  1. -1.]]

```

```

D matrix for element 5
[[3.19999985e+11 7.99999962e+10 0.00000000e+00]
 [7.99999962e+10 3.19999985e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.19999994e+11]]
u matrix for element 5
[[-1.17749567e-07]
 [-3.80605723e-08]
 [-2.91215096e-08]
 [-1.53677432e-07]
 [-3.29055550e-08]
 [-3.25787519e-08]]
B matrix for element 6
[[-1.  0.  1.  0.  0.  0.]
 [ 0.  0.  0. -1.  0.  1.]
 [ 0. -1. -1.  1.  1.  0.]]
D matrix for element 6
[[2.38095235e+11 9.52380952e+10 0.00000000e+00]
 [9.52380952e+10 2.38095235e+11 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 7.14285697e+10]]
u matrix for element 6
[[-1.17749567e-07]
 [-3.80605723e-08]
 [-1.26697888e-07]
 [-1.58729112e-07]
 [-2.91215096e-08]
 [-1.53677432e-07]]

```

In [13]:

```

print(strain)
print(stress)

```

```

[[ 1.79535428e-13 -5.12097748e-09 -5.12097748e-09 -5.25591090e-09
   3.78404535e-09 -8.94832040e-09]
 [-1.68648609e-08 3.25784519e-08 5.48182045e-09 -5.25591090e-09
   5.48182045e-09 5.05167994e-09]
 [ 2.77840377e-08 -1.65381181e-08 3.54002493e-08 2.45275846e-08
  -3.62546678e-08 -2.30921616e-08]]
[[-1606.13448695  967.56330966 -1200.16710167 -1751.97028274
  1649.44007011 -1649.44007011]
 [-4015.42592525 10015.42592525 1344.50427996 -1751.97028274
  2056.9060729  350.55992989]
 [ 1984.57407475 -1984.57407475 4248.02971726 1751.97028274
 -4350.55992989 -1649.44007011]]

```

In []:

In []:

In [66]:

```
import numpy as np
import scipy.integrate as evaluation_integral
import math
import matplotlib.pyplot as plot
```

In [67]:

```
n = int(input("Dimension = "))
# Define the lower limit & upper limit for the integrals
lower_limit = 0
upper_limit = math.pi
```

Dimension = 4

In [68]:

```
# Define a function that generates the base functions for the polynomial function
def base_function_generator(p):
    # Define a base function as a lambda function of x raised to the power p
    base_function = lambda x:x**p
    return base_function
```

In [69]:

```
# Create a list of base functions for the polynomial space up to degree n-1
b = []
for i in range(n):
    b.append(base_function_generator(i))
```

In [70]:

```
# Initialize the vector F, which contains the integrals of the product of each base func
F = np.zeros(n)
for i in range(n):
    F[i] = (evaluation_integral.quad(lambda x:(b[i](x)* math.sin(x)),lower_limit, upper_limit)[0]
```

In [71]:

```
K = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        K[i,j] = (evaluation_integral.quad(lambda x:(b[i](x)*b[j](x)),lower_limit, upper_limit)[0]
```

In [72]:

```
C = np.linalg.solve(K, F)
```

In [73]:

```
x = np.arange(lower_limit, upper_limit, 0.01)
# Define the actual function to be approximated
y1 = np.sin(x)
```

In [74]:

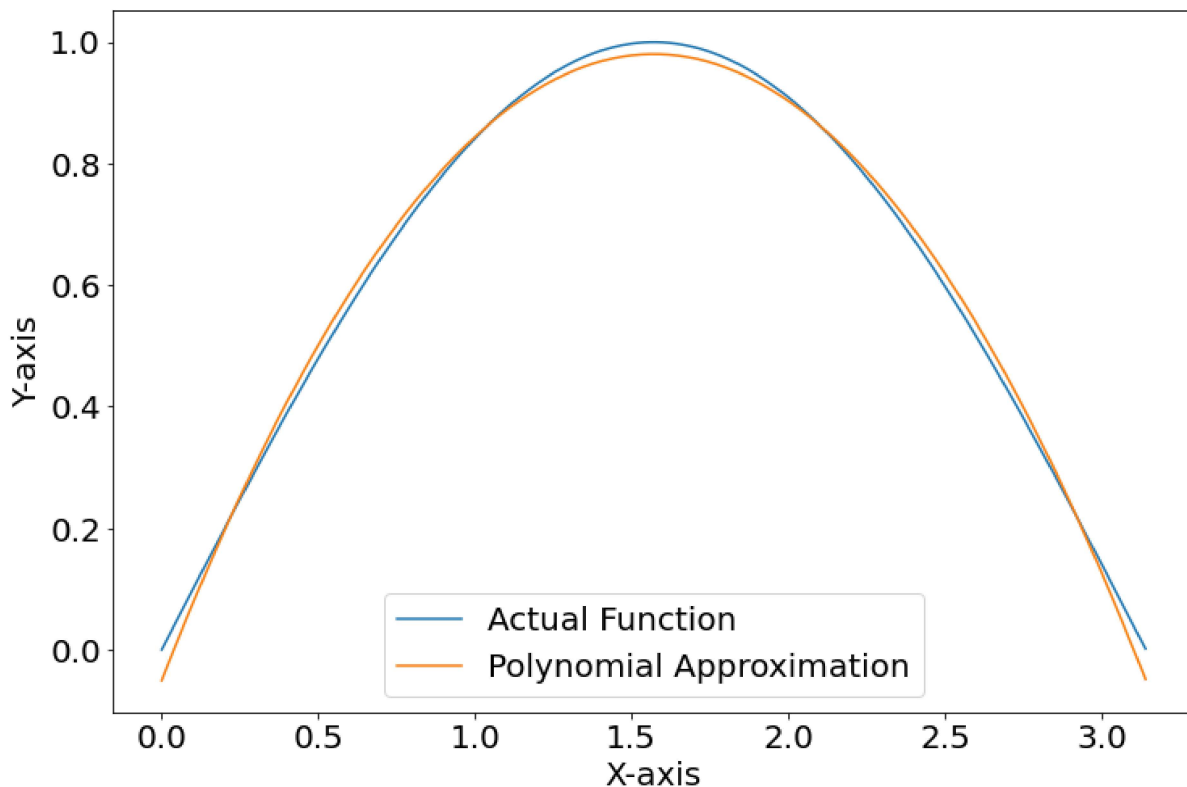
```
l = len(C)

# Creates an array y2 with the same size as the x array and initializes all of its elements
y2 = np.zeros_like(x)

for i in range(n):
    y2 = C[i]*x**i + y2
```

In [75]:

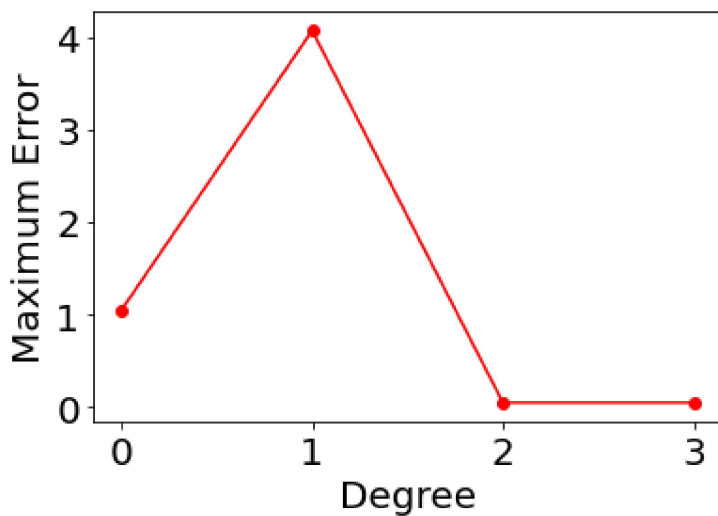
```
# Plot the actual function sin(x) and the polynomial approximation y2
plot.figure(figsize=(12,8))
plot.plot(x, y1, label="Actual Function")
plot.plot(x, y2, label="Polynomial Approximation")
plot.xlabel('X-axis', fontsize=20) # increase font size to 20
plot.ylabel('Y-axis', fontsize=20) # increase font size to 20
plot.tick_params(axis='both', which='major', labelsize=20) # increase font size to 20
Dimension = 4
# Add a Legend to the plot
plot.legend(fontsize=20)
plot.show()
```



In [76]:

```
# plot the error for every degree polynomial
error = []
for i in range(n):
    y = np.zeros_like(x)
    for j in range(i+1):
        y += C[j]*x**j
    err = np.abs(y1-y)
    error.append(np.max(err))

plot.figure(figsize=(6,4))
plot.plot(range(n), error, 'ro-')
plot.xlabel('Degree', fontsize=20)
plot.ylabel('Maximum Error', fontsize=20)
plot.tick_params(axis='both', which='major', labelsize=20)
plot.show()
Dimension = 4
```



In []:

In []: