Finite Element Method Implementation using PyGMSH meshing in Python

We aim to calculate the finite element solution of the partial differential eqeuation given by:

$$-\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)=1, \quad x^2+y^2<1,$$

Subject to Dirichlet boundary conditions on the whole boundary, given by:

$$u(x,y)=0, \quad x^2+y^2=1$$

This boundary value problem has solution:

$$u(x,y)=\frac{1}{4}(1-x^2-y^2)$$

Bounded on the interval, $x^2+y^2=1$.

We start by determining the week solution of the PDE. As outlined by Whitley, Equation 7.10 defines the weak form of the PDE as follows:

$$\int_\Omega (\nabla v)\cdot(\nabla u)\,dA=\int_\Omega v\,dA$$

By Equation 7.25, the finite element solution is as follows:

$$U(x,y)=\sum_{j=1}^{N_{\text{node}}} U_j\phi_j(x,y)$$

By Equation 7.36, the local stiffness matrix is defined as follows:

$$A^{(k)}_{\text{local},ij}=\int_{e_k}(\nabla\phi_i)\cdot(\nabla\phi_{k_j})\,dA=\int_{e_k}\left(\frac{\partial\phi_{k_i}}{\partial x}\frac{\partial\phi_{k_j}}{\partial x}+\frac{\partial\phi_{k_i}}{\partial y}\frac{\partial\phi_{k_j}}{\partial y}\right)dA=\int_\Delta\left(\frac{\partial\phi_{\text{local},i}}{\partial x}\frac{\partial\phi_{\text{local},j}}{\partial x}+\frac{\partial\phi_{\text{local},i}}{\partial y}\frac{\partial\phi_{\text{local},j}}{\partial y}\right)\text{de}$$

Where F, or the Jacobian/Transformation Matrix, is defined as follows:

$$F=\begin{pmatrix}\frac{\partial x}{\partial X} & \frac{\partial x}{\partial Y}\\ \frac{\partial y}{\partial X} & \frac{\partial y}{\partial Y}\end{pmatrix}=\begin{pmatrix}x_{k_2}-x_{k_1} & x_{k_3}-x_{k_1}\\ y_{k_2}-y_{k_1} & y_{k_3}-y_{k_1}\end{pmatrix}$$

Equation 7.41 simplifies the integral through the following:

$$A^{(k)}_{\text{local},ij}=\frac{1}{2}\det(F)\left(\frac{\partial\phi_{\text{local},i}}{\partial x}\frac{\partial\phi_{\text{local},j}}{\partial x}+\frac{\partial\phi_{\text{local},i}}{\partial y}\frac{\partial\phi_{\text{local},j}}{\partial y}\right)$$

The local load vector can be defined as follows:

$$b^{(k)}_{\text{local},i}=\int_{e_k}\phi_{k_1}\,dA=\int_\Delta\phi_{\text{local},i}(X,Y)\det(F)\,dA_x$$

Which can be updated as follows for our interval:

$$b^{(k)}_{local,i} = \int_0^{1-Y} \int_0^1 \phi_{local,i}(X,Y)det(F)dxdy$$

Finally, the basis functions will be defined as follows for triangular elements:

$$\phi_1 = 1 - X - Y,\ \phi_2 = X,\ and\ \phi_2 = Y$$

With the system defined, we now seek to define the mesh for the given interval. Using the Python library pygmsh,the 2-dimensional circle with radius 1 and center at (0,0) will be meshed. For reference, an example mesh output is included below

Mesh Size	N_x	N_y
1	13	13

Node ID	x	y	z
1	1	0	0
2	-0.5	0.866025	0
3	-0.5	-0.86603	0
4	0.766044	0.642788	0
5	0.173648	0.984808	0
6	-0.93969	0.34202	0
7	-0.93969	-0.34202	0
8	0.173648	-0.98481	0
9	0.766044	-0.64279	0
10	-0.26604	-0.22324	0
11	0.256771	0.215456	0
12	-0.38302	0.321394	0
13	0.25	-0.43301	0

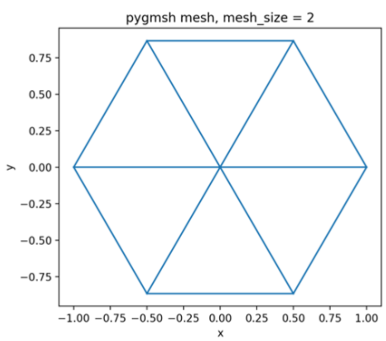
Element	Node 1	Node 2	Node 3
1	11	5	12
2	4	5	11
3	10	3	13
4	7	10	12
5	3	8	13
6	6	7	12
7	5	2	12
8	9	1	13
9	1	11	13
10	1	4	11
11	7	3	10
12	10	11	12
13	11	10	13
14	2	6	12
15	8	9	13

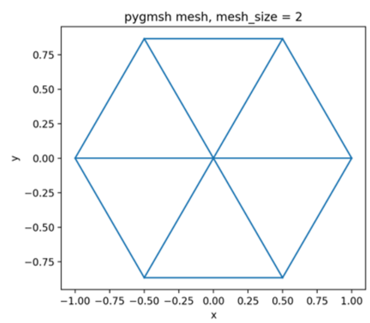
To interpret the meshsize refinement and how it relates to the number of nodes, the following table is presented.

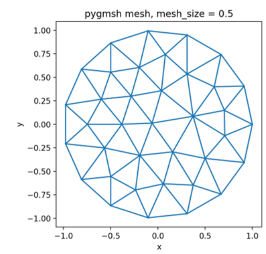
Mesh size	N_x	N_y

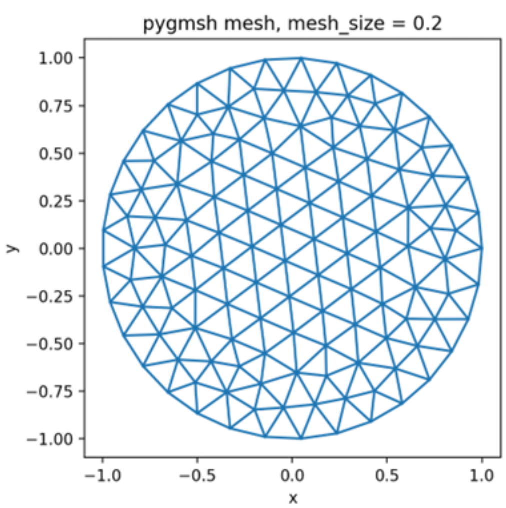
2	7	7
1	13	13
0.5	37	37
0.2	129	129
0.1	413	413

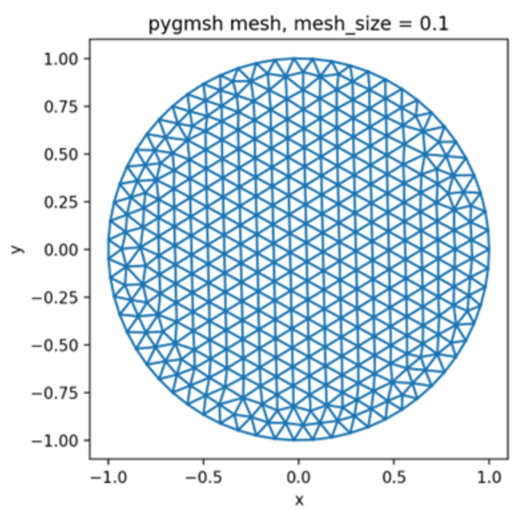
Now, varying mesh sizes are as follows:



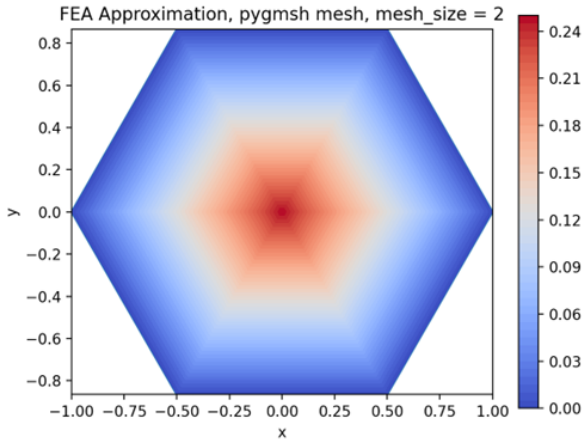


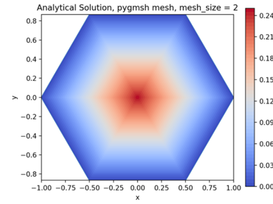


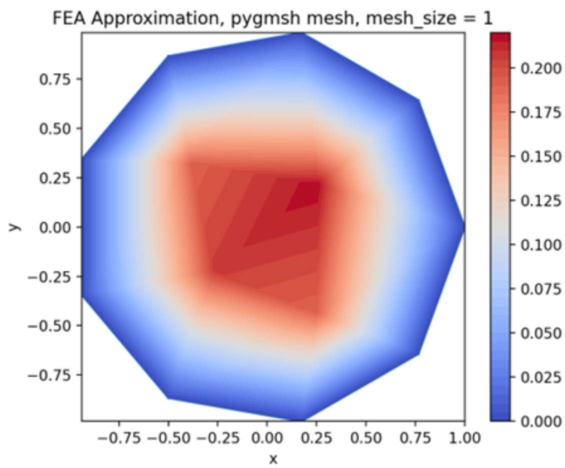


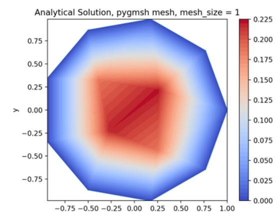


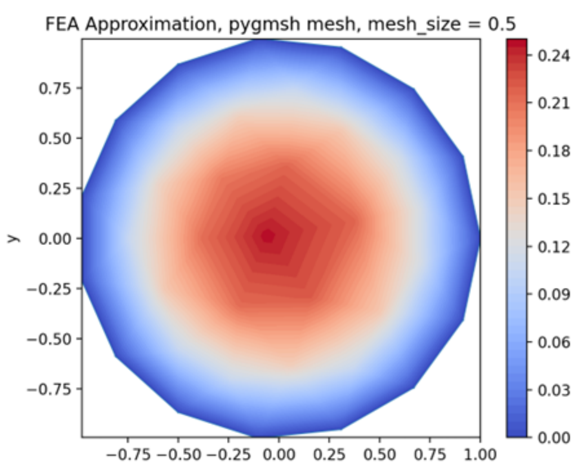
Now, plotting the calculating solution using a contour plot for varying mesh sizes. The contour plots are included in pairs – the first plot is the FEA approximation and the second plot is the analytical solution.

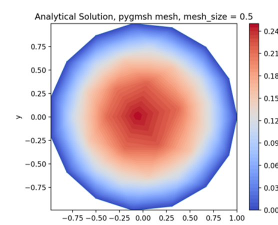

FEA Approximation, pygmsh mesh, mesh_size = 2
y
x
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.24
0.21
0.18
0.15
0.12
0.09
0.06
0.03
0.00

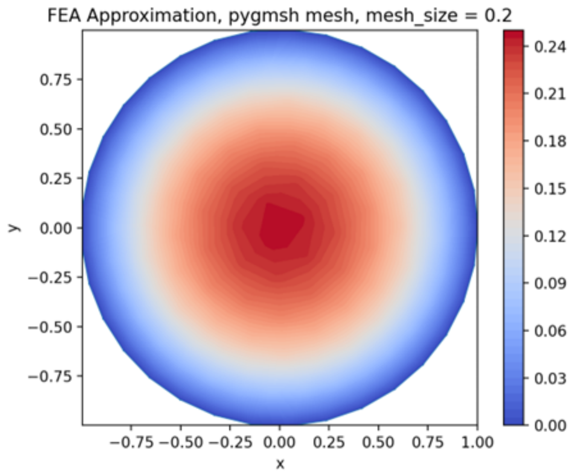

Analytical Solution, pygmsh mesh, mesh_size = 2

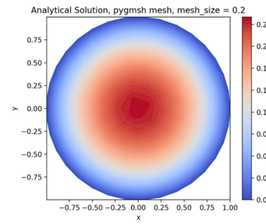

FEA Approximation, pygmsh mesh, mesh_size = 1
y
x
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000

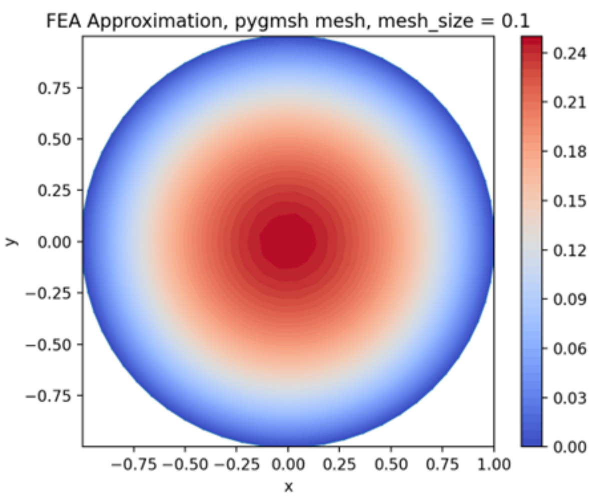

Analytical Solution, pygmsh mesh, mesh_size = 1

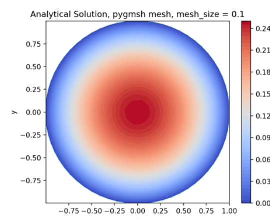

FEA Approximation, pygmsh mesh, mesh_size = 0.5
y
x
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.24
0.21
0.18
0.15
0.12
0.09
0.06
0.03
0.00


Analytical Solution, pygmsh mesh, mesh_size = 0.5









Work Cited

Whiteley, J. (2017). *Finite element methods: A Practical Guide*. Springer.

Appendix

```
import pygmsh
import numpy as np
from sympy import symbols, integrate, simplify
import matplotlib.pyplot as plt
from sympy import Matrix
import matplotlib.tri as tri
import time

X=symbols('X')
Y=symbols('Y')
```

```
def u_analytical(x,y):
    return (0.25)*(1-(x**2)-(y**2))

size = 2
with pygmsh.geo.Geometry() as geom:
    r = 1.0
    circle = geom.add_circle([0, 0, 0], r, mesh_size=size)
    mesh = geom.generate_mesh()

point = mesh.points
x= mesh.points[1:,0]
y= mesh.points[1:,1]
loca = mesh.cells_dict["triangle"]

#Determining Nx and Ny
Nx = len(np.unique(x))
print(f"Nx: {Nx}")
Ny = len(np.unique(y))
print(f"Ny: {Ny}")

#Defining size of global matrix and vector
n=len(point[:,0])-1

#Define global force vector
K=np.zeros((n,n),dtype=float)

#Define global stiffness matrix
f=np.zeros((n,1),dtype=float)

#Building locals and adding to globals for each element
for e in range(len(loca[:,0])):

    #Extracting physical dimensions for each node
    nodes_element=loca[e]

    #Basis functions for triangle mesh
    phi = Matrix([1 - X - Y, X, Y])

    #Defining local stiffness matrix at each element
    K_local=np.zeros((3,3),dtype=float)

    f_local=np.zeros((3,1),dtype=float)

    #Extracting physical dimensions from mesh
    coords=np.array([mesh.points[nodes_element[0]],mesh.points[nodes_element[1]],mesh.points[nodes_element[2]]])
    xk1,yk1=coords[0,:2]
    xk2,yk2=coords[1,:2]
    xk3,yk3=coords[2,:2]

    #Jacobian from physical dimensions
    Jac = np.array([[xk2 - xk1, xk3 - xk1],
              [yk2 - yk1, yk3 - yk1]])
    detJ = np.linalg.det(Jac)
    invJT = np.linalg.inv(Jac).T  # J^{-T}
```

```
    #Defining gradient
    gradient = np.array([[-1,-1],[1,0],[0,1]])

    #Defining gradient in physical dimensions
    gradient_X_Y = gradient@invJT.T

    #Determining area based on determininat
    area = abs(detJ)/2

    #Defining Local for each element
    for i in range(0,3):
        for j in range(0,3):
            K_local[i,j] = np.dot(gradient_X_Y[i], gradient_X_Y[j])*area
        aux2= simplify(phi[i]*float(abs(detJ)))
        f_local[i]=float(integrate(integrate(simplify(aux2), (X, 0, 1 - Y)), (Y, 0, 1)))

    #Adding locals to globals
    for i in range(3):
        for j in range(3):
            K[nodes_element[i]-1,nodes_element[j]-1]+=K_local[i,j]
        f[nodes_element[i]-1]+=f_local[i]

#Boundary Conditions
zeros =[]
aux=1e-6
i = 0
#Determining where boundary conditions should apply
for (x,y,_) in (mesh.points):
    if abs(x**2+y**2-1)<aux:
        zeros.append(i)
    i+=1

#Applying the boundary conditions
for i in zeros:
    K[i-1,:]=0
    K[:,i-1]=0
    f[i-1]=0
    K[i-1,i-1]=1

#Backslash
ts_bs_1 = time.time()
u=np.linalg.solve(K,f)
ts_bs_2 = time.time()
print(f"Backslah Performance:{ts_bs_2-ts_bs_1}")

#Regular matrix multiplication
ts_l_1=time.time()
u = np.linalg.inv(K)@f
ts_l_2 = time.time()
print(f"Linear Algebra Performance:{ts_l_2-ts_l_1}")

u =np.vstack([np.zeros((1,1)),u])
```

```
#Determning analytical solution
x_analytical =mesh.points[:,0]
y_analytical =mesh.points[:,1]
u_an=u_analytical(x_analytical,y_analytical)


#Processing u for plotting
u_flat=u.flatten()
x= mesh.points[:,0]
y= mesh.points[:,1]
triangle=tri.Triangulation(x,y,loca)

#Plotting FEA Approximation
plt.triplot(x, y, loca)
plt.gca().set_aspect("equal")
cont = plt.tricontourf(triangle,u_flat, levels=50, cmap="coolwarm")
plt.colorbar(cont)
plt.title(f"FEA Approximation, pygmsh mesh, mesh_size = {size}")
plt.xlabel('x')
plt.ylabel('y')

#Plotting Analytical Solution
plt.figure(2)
triangle=tri.Triangulation(x,y,loca)

plt.triplot(x, y, loca)
plt.gca().set_aspect("equal")
cont2 = plt.tricontourf(triangle,u_an, levels=50, cmap="coolwarm")
plt.colorbar(cont2)
plt.title(f"Analytical Solution, pygmsh mesh, mesh_size = {size}")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```