ME5204 - Finite Element Analysis

Assignment 2 - Numerical Integration

Name - Jay Prajapati
Roll number - ME21B143

Table of Contents

1 Extraction of Nodes and Elements

- From the previous assignment, we read the MSH file and figure out the nodes and elements from it. The nodes are divided into two parts first outer border points and the second ones are inner mesh points. The elements are read all at once.
- The following code snippet is what I have implemented to extract nodes and elements from msh file. The start and end points are hard coded by carefully observing the msh file.

```
    nodes = []
    elements = []

    with open(file_path, 'r') as file:
        lines = file.readlines()

        # Get Nodes for outer border points
        for i in range(node_start_line1 - 1, node_end_line1, 3):
            parts = lines[i].split()
            x, y, z = float(parts[0]), float(parts[1]), float(parts[2])
            nodes.append((x, y, z))

        # Get Nodes for inner mesh points
        for i in range(node_start_line2 - 1, node_end_line2):
            parts = lines[i].split()
            x, y, z = float(parts[0]), float(parts[1]), float(parts[2])
            nodes.append((x, y, z))

        # Get Elements
        for i in range(element_start_line - 1, element_end_line):
            parts = lines[i].split()
            t1, t2, t3 = int(parts[1]), int(parts[2]), int(parts[3])
            elements.append((t1, t2, t3))

    nodes = np.array(nodes)
    elements = np.array(elements)
```

- The nodes stores the vertex information for each point. The elements stores the index of triangle formed by three vertices. The size of nodes is **1924** and total elements i.e. number of triangular elements is **3659**.

2 Area of IITM

- From the previous assignment, we already have the meshing done of IITM map. For calculating the total area, we have to sum the areas of all small triangular elements using **Gauss quadrature**.

$$\int\int f(x,y)dA = \int\int f(\varepsilon,\eta)|J|d\varepsilon d\eta = \sum_I f(\varepsilon,\eta) * |J| * w_I$$

- Since f function is planar, it is constant for any value of ε and η. Mathematical expression for jacobian is as follows.

$$J = \begin{vmatrix} \frac{\partial x}{\partial \varepsilon} & \frac{\partial y}{\partial \varepsilon} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix}$$

where

$$x = (1-\varepsilon-\eta)x_0 + \varepsilon x_1 + \eta x_2$$
$$y = (1-\varepsilon-\eta)y_0 + \varepsilon y_1 + \eta y_2$$
$$\frac{\partial x}{\partial \varepsilon} = x_1 - x_0, \quad \frac{\partial y}{\partial \varepsilon} = y_1 - y_0$$
$$\frac{\partial x}{\partial \varepsilon} = x_2 - x_0, \quad \frac{\partial y}{\partial \varepsilon} = y_2 - y_0$$

Note:- All partial derivatives are independent of ε and η. I have used one point numerical integration method to calculate the area. The point is $(1/3, 1/3)$ and weight is $1/2$.

- The following code shows on how I calculated jacobian for each triangle.

```python
def jacobian(triangles):
    # 0-based indexing
    t1, t2, t3 = triangles[0] - 1, triangles[1] - 1, triangles[2] - 1

    x0, y0 = nodes[t1][0], nodes[t1][1]
    x1, y1 = nodes[t2][0], nodes[t2][1]
    x2, y2 = nodes[t3][0], nodes[t3][1]

    del_x_del_ep = x1 - x0
    del_y_del_ep = y1 - y0
    del_x_del_enta = x2 - x0
    del_y_del_enta = y2 - y0

    return del_x_del_ep * del_y_del_enta - del_x_del_enta * del_y_del_ep
```

- The Area function now calculates the area of all triangular elements by making use of jacobian function. The area that we get from here will not be exact because we measured it from an image. So, we will have to use the scaling factor from the scale given in the image. It comes out that for every 100 meters, it is 23 units. So, area is multiplied by $0.1^2/23^2$ after accounting for kilometer conversion.

```python
def Area_IITM():
    Area_of_IITM = 0
    Actual_Area = 2.5 # km^2 equivalent to 617 acres
    weight = 0.5

    for triangles in elements:
        J = jacobian(triangles)
        Area_of_IITM += (J * weight)

    # Scaling Factor applied and converted to sq.km
    Area_of_IITM = Area_of_IITM * 0.1 * 0.1 / (23 * 23)
    print("Area of IIT Madras is", Area_of_IITM,"km^2")

    Error = (Actual_Area - Area_of_IITM) / Actual_Area * 100
    print("Error =", Error, "%")
```

- Finally, the area of IITM calculated is **2.4604108695652225** km^2. For comapring it to actual area, I have calculated the error which is given as below.

$$Error = \frac{(Actual\ Area - Calculated\ Area)}{Actual\ Area}$$
$$= \mathbf{1.583565217391101}\%$$

3 Area Moment of Inertia wrt Centroid

- The formula to get area moment of inertia about centroid is given as below.

$$Area\,Moment\,of\,Inertia = \int\int((x - x_c)^2 + (y - y_c)^2)dA$$

$$= \sum_I f(\varepsilon, \eta) * |J| * w_I$$

- I have used **3 point numerical integration** method to evaluate the above integral assuming inside function as f.
- Firstly we will need the centroid of entire geometry which is evaluated as follows.

$$Centroid = \frac{\sum_i Area_i * centroid_i}{\sum_i Area_i}$$

```
    Total_area = 0
    Centroid = np.zeros((1, 2))

    for triangle in elements:
        area_of_triangle, centroid_of_triangle = jacobian(triangle)
        area_of_triangle /= 2
        Total_area += area_of_triangle
        Centroid += area_of_triangle * centroid_of_triangle

    Centroid /= Total_area
```

$$centroid = [339.84875955, 285.66608553]$$

- Now for the jacobian function, it is similar to the above just that we will evaluate the centroid of each triangle as well.

```
def jacobian(triangles):
    t1, t2, t3 = triangles[0] - 1, triangles[1] - 1, triangles[2] - 1
    centroid = np.zeros((1, 2))
    x0, y0 = nodes[t1][0], nodes[t1][1]
    x1, y1 = nodes[t2][0], nodes[t2][1]
    x2, y2 = nodes[t3][0], nodes[t3][1]

    centroid[0][0] = (x0 + x1 + x2)/3
    centroid[0][1] = (y0 + y1 + y2)/3
```

```
    del_x_del_ep = x1 - x0
    del_y_del_ep = y1 - y0
    del_x_del_enta = x2 - x0
    del_y_del_enta = y2 - y0
    return del_x_del_ep * del_y_del_enta - del_x_del_enta * del_y_del_ep
                                                                , centroid
```

- Another important function is evaluate the f(ε, η). The three points are (2/3,1/6), (1/6,2/3), (1/6,1/6) and weights are 1/6 each. The following represents teh relationship between x, y and ε, η.

$$x = (1 - \varepsilon - \eta)x_0 + \varepsilon x_1 + \eta x_2$$

$$y = (1 - \varepsilon - \eta)y_0 + \varepsilon y_1 + \eta y_2$$

```
def f(points, triangle, centroid):
    # 0-based indexing
    t1, t2, t3 = triangle[0] - 1, triangle[1] - 1, triangle[2] - 1
    c0, c1 = centroid[0][0], centroid[0][1]
    epsilon, enta = points[0], points[1]

    x0, y0 = nodes[t1][0], nodes[t1][1]
    x1, y1 = nodes[t2][0], nodes[t2][1]
    x2, y2 = nodes[t3][0], nodes[t3][1]

    X = (1.0 - epsilon - enta) * x0 + epsilon * x1 + enta * x2
    Y = (1.0 - epsilon - enta) * y0 + epsilon * y1 + enta * y2

    return (X - c0)**2 + (Y - c1)**2
```

- Finally we will evaluate the jacobian and f function for each triangular element and get the area moment as follows. Note that scaling factor comes here as well. So, area moment will be multiplied by $0.1^4/23^4$ after accounting for kilometer conversion.

```
points = np.array([[2/3, 1/6], [1/6, 2/3], [1/6, 1/6]])

def Area_Moment():
    Area_Moment = 0
    Total_area = 0
    Centroid = np.zeros((1, 2))

    for triangle in elements:
        area_of_triangle, centroid_of_triangle = jacobian(triangle)
        area_of_triangle /= 2
        Total_area += area_of_triangle
```

```
        Centroid += area_of_triangle * centroid_of_triangle

    Centroid /= Total_area

    for triangle in elements:
        J, _ = jacobian(triangle)
        Area_Moment += ((f(points[0], triangle, Centroid) +
                         f(points[1], triangle, Centroid) +
                         f(points[2], triangle, Centroid)) * J) / 6

    Area_Moment = Area_Moment * 0.1 ** 4/ (23 ** 4)
    print("Area Moment of Inertia about Centroid is",Area_Moment,"km^4")
```

- Finally the Area Moment of Inertia of IITM wrt Centroid is **1.5668184921026416** km^4.

4 Area Moment of Inertia wrt Gajendra Circle

- The coordinates of Gajendra circle that we get using GMSH are (286.9, 260.6) which is not the same as centroid so we will have to use parallel axis theorem given as below.

$$I_{GC} = I_{centroid} + Area * d^2$$

  where d is distance between centroid and Gajendra Circle

- The following code shows on how is area moment calculated using parallel axis theorem.

```
    Area_Moment_GC = Area_Moment +
                     Total_area * ((Centroid[0][0] - GC_coords[0])**2 +
                                   (Centroid[0][1] - GC_coords[1])**2)
    # Scaling factor
    Area_Moment_GC = Area_Moment_GC * 0.1 ** 4/ (23 ** 4)
    print("Area Moment of Inertia about GC is",Area_Moment_GC,"km*4")
```

- Finally the Area Moment of Inertia of IITM wrt Gajendra Circle is **1.7264372882816095** km^4.
- The following figure depicts the Gajendra circle and centroid in the map. (meshing is made invisible for clarity)

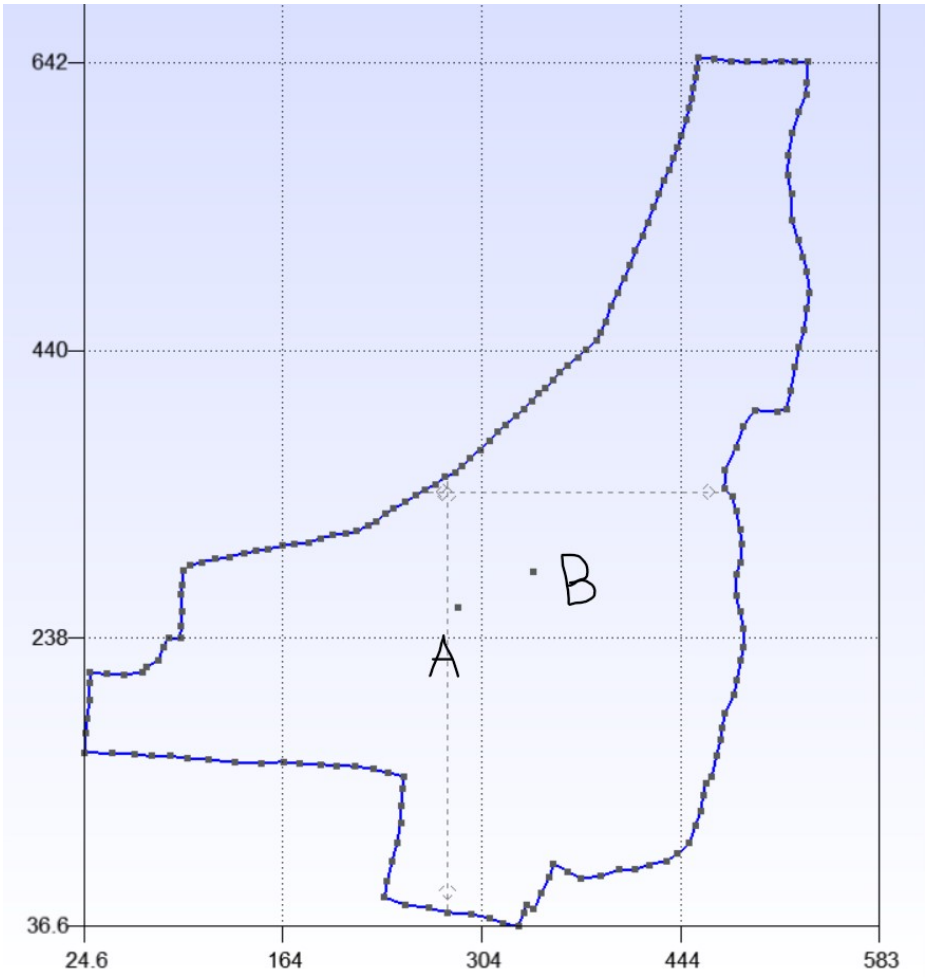

Figure 1: A is Gajendra circle and B is centroid

Figure 2: Scale