BAREKENG: Journal of Mathematics and Its Applications
September 2025 Volume 19 Issue 3 Page 1637-1648
P-ISSN: 1978-7227 E-ISSN: 2615-3017
https://doi.org/10.30598/barekengvol19iss3pp1637-1648

THE SHOELACE ALGORITHM IN ENGINEERING: PYTHON APPLICATIONS FOR AREA AND INERTIAL ANALYSIS

Pankaj Dumka [1*], **Dhananjay R. Mishra**[2]

[1,2]*Department of Mechanical Engineering, Jaypee University of Engineering and Technology A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India*

*Corresponding author's e-mail: * **p.dumka.ipec@gmail.com***

Article History:

Received: 22nd January 2024
Revised: 10th February 2025
Accepted: 11th March 2025
Published: 1st July 2025

Keywords:

Computational Geometry;
Engineering Analysis;
Geometric Properties;
Python Programming;
Shoelace Method.

ABSTRACT

The Shoelace Method is a classic mathematical formula for the determination of the area of polygons. This method is based on the vertex coordinates of a polygon and has significant applications in science and engineering. This article explores the method's extension to calculate the centroids and moments of inertia of plane shapes, which is essential for structural and mechanical analysis. By executing these calculations in Python programming, the study shows the method's practicality and flexibility for modern engineering tasks. The article introduces a Python-based structure using libraries like NumPy, Shapely, and Matplotlib for enabling efficient computational modelling and visualization. Through example problems, the versatility of the Shoelace Method in solving real-world engineering shapes is showcased.

How to cite this article:

P. Dumka and D. R. Mishra., "THE SHOELACE ALGORITHM IN ENGINEERING: PYTHON APPLICATIONS FOR AREA AND INERTIAL ANALYSIS," *BAREKENG: J. Math. & App.*, vol. 19, no. 3, pp. 1637-1648, September, 2025.

Journal homepage: *https://ojs3.unpatti.ac.id/index.php/barekeng/*
Journal e-mail: *barekeng.math@yahoo.com; barekeng.journal@mail.unpatti.ac.id*

Research Article · Open Access

1. INTRODUCTION

The Shoelace Method, which is also known as the “Surveyor's Formula” or “Gauss's Area Formula”, is a mathematical method for calculating the area of a simple polygon when the coordinates of its vertices are known [1], [2]. The origin of the method dates back to the works of Carl Friedrich Gauss in the 18th century, although the method's widespread use became famous in the field of surveying, where accurate calculations of land areas are essential [3], [4]. The method is named for the way its formula involves, viz., summing and subtracting cross-products of coordinates. That’s why it looks like the interweaving of laces, and hence the name, Shoelace.

Over time, this method has gone beyond its original use, i.e., finding applications in structural and mechanical analysis for calculating the key geometric properties, viz., centroids and moments of inertia [5]. These properties are fundamental to engineering design and analysis, thereby aiding in the estimation of stability, strength, and load distribution in structures. Therefore, the Shoelace Method acts as a bridge between geometry and engineering mechanics, offering a powerful approach to solving practical problems that arise in engineering.

Integrating the Shoelace Method into engineering education not only enhances students’ perception of computational geometry but also stresses the importance of algorithmic thinking in solving real-world problems. With the growing dependence on computational tools in engineering the programming skills are necessary [6], [7]. Among modern programming languages the Python has come out as a handy and open tool, celebrated for its ease of use and rich system of libraries (NumPy [8]–[12], SymPy [13]–[15], Matplotlib [16], Shapely [17], etc) that can simplify complex tasks [18].

This article presents a novel Python implementation of the Shoelace Method, presenting its application to the calculation of centroids and moments of inertia. By utilizing Python's computational power, the conventional Shoelace Method is changed into a dynamic tool for engineering education, which will enable students and professionals to automate calculations and visualize results with precision and efficiency. The proposed approach not only emphasizes the significance of programming in present engineering practices but also displays the innovative ability of Python in increasing the understanding of foundational concepts.

2. RESEARCH METHODS

2.1 Mathematics of the Shoelace Method

The method is called a shoelace as it looks very similar to the way one ties the laces of a shoe, as shown in Figure 1.

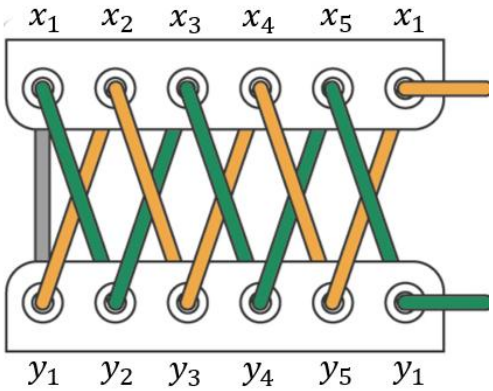

Figure 1. Graphical Representation of the Shoelace Problem

The method is traditionally used for computing the area of a polygon based on its vertex coordinates. However, this method can be expanded to determine additional geometric properties such as centroids and moments of inertia by modifying the summation process. The centroid, representing the geometric center, is obtained by calculating the first moment of area using weighted coordinate summations, while the moment

of inertia is derived from second-order moment summations. These expansions transform the Shoelace Method into a more comprehensive computational tool for engineering analysis.

In this section, a detailed mathematical background of the method is presented, i.e., how this can be used to evaluate the area and the moment of inertia.

2.1.1 Formula for Area

Given a polynomial with the vertices $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, the area can be computed by dividing the polygon into trapezoidal strips along the x-axis [3]:

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})\right| \tag{1}$$

where $x_{n+1} = x_1$ and $y_{n+1} = y_1$. This will ensure that the polynomial is closed. This formula calculates the signed area of the trapezoids formed by the polygon edges and the x-axis. As can be seen in Figure 2, half of the cross product of the vectors (x_i, y_i) and (x_{i+1}, y_{i+1}) will be equal to the area of the trapezoid they form, i.e., $\frac{1}{2}(x_i y_{i+1} - y_i x_{i+1})$. Thus, summing all such areas will return the final area of the full polygon.

$$|\vec{A} \times \vec{B}| = (x_i y_{i+1} - y_i x_{i+1})$$

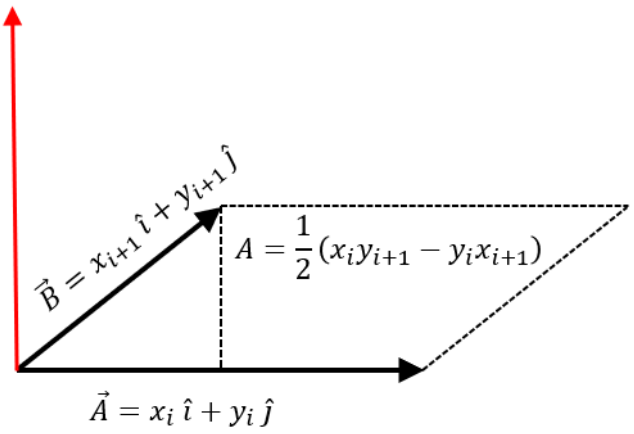

Figure 2. Area of Trapezoid Formed by The Vectors

2.1.3 Formula for Centroid

The centroid $(\bar{x}, \bar{y})$ is the centre of the area of the polygon (Refer to Figure 3). Assuming uniform distribution of area, the derivation of the formulas is as follows [19].

The x-coordinate of the centroid is determined by summing the moment contribution of small trapezoidal areas weighted by their average x coordinate, as shown in Equation (2).

$$x_c = \frac{1}{A}\sum_{i=1}^{n} \bar{x_i} a \tag{2}$$

Here, $\bar{x_i}$ is the average x-coordinate of the strip between the points (x_i, y_i) and (x_{i+1}, y_{i+1}) as shown below:

$$\bar{x_i} = \frac{x_i + x_{i+1}}{2} \tag{3}$$

Since the area of each strip is:

$$A_i = \frac{1}{2}(x_i y_{i+1} - x_{i+1} y_i) \tag{4}$$

By substituting into the centroid formula, one can get:

$$x_c = \frac{1}{A}\sum_{i}^{n}\left(\frac{x_i + x_{i+1}}{2}\right)\frac{1}{2}\left(x_i y_{i+1} - x_{i+1}y_i\right) = \frac{1}{A}\sum_{i}^{n}\left(\frac{(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1}y_i)}{4}\right) \tag{5}$$

Since A is commuted using the shoelace formula, the factor of ¼ cancels out with the area's ½ factor, resulting in **Equation (6)**.

$$\bar{x} = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - y_i x_{i+1}) \tag{6}$$

Similarly, the y centroid can also be obtained, $\bar{y}$ can be obtained as follows:

$$\bar{y} = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - y_i x_{i+1}) \tag{7}$$

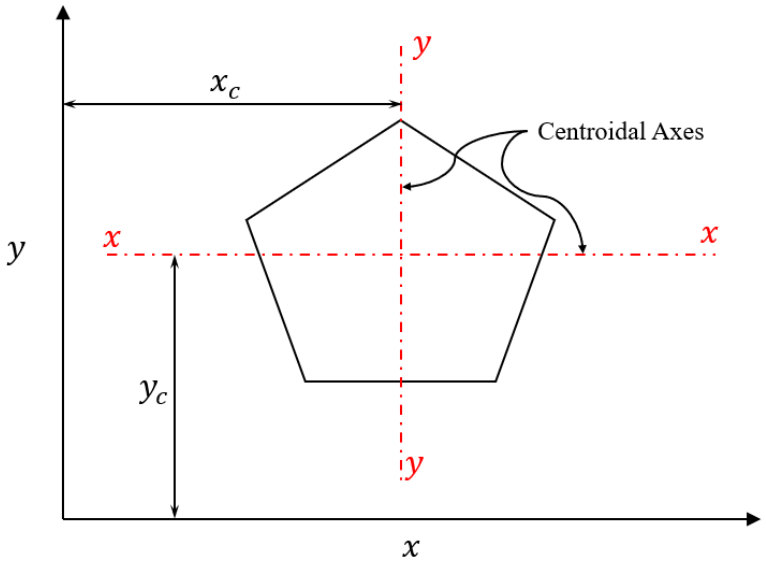

Figure 3. Representation of General $x - y$ Axis and Centroidal Axis

These equations represent the first moment of the individual moments of the areas about the respective axes. The factor of 6 in the centroid formula arises from the integration of x and y coordinates over the triangular areas that make up the polygon.

2.1.3 Formula for Moment of Inertia

The moment of inertia, which is the second moment of area, is obtained by taking the moment twice. For a polygon, it is computed as a summation of area-weighted squared distances from a reference axis.

Using a similar approach as in the centroid derivation, the moment of inertia formulas are obtained by summing the second moment of area contributions from each polygon edge. The moment of inertia about the x-axis and y-axis is calculated by weighting each trapezoidal strip's contribution with the squared distance of its centroid from the respective axis. The average squared coordinate values for each edge are computed as $\frac{y_i^2 + y_i y_{i+1} + y_i^2}{3}$ for I_x and $\frac{x_i^2 + x_i x_{i+1} + x_i^2}{3}$ for I_y. These terms account for the distribution of mass within each segment. Thus, the resulting expressions for I_x and I_y as follows **[19]**:

$$I_x = \frac{1}{12}\sum_{i=1}^{n}\left(y_i^2 + y_i y_{i+1} + y_{i+1}^2\right)(x_i y_{i+1} - y_i x_{i+1}) \tag{8}$$

$$I_y = \frac{1}{12}\sum_{i=1}^{n}(x_i^2 + x_i x_{i+1} + x_{i+1}^2)(x_i y_{i+1} - y_i x_{i+1}) \quad (9)$$

where I_x and I_y are the moment of inertia about x and y axis respectively (Refer **to Figure 3**). The factor of 12 in the moment of inertia formula comes from the integration of x^2 and y^2 over those same triangular areas.

For obtaining the moment of inertia about the centroidal axis, the parallel axis theorem can be used **[20]**. Thus, the respective moment of inertia about $x - x$ and $y - y$ axes can be obtained by using **Equation (10)** and **Equation (11)**.

$$I_{xx} = I_x - A \times y_c^2 \quad (10)$$

$$I_{yy} = I_y - A \times x_c^2 \quad (11)$$

2.2 Computational Complexity Analysis

The proposed algorithm for calculating area, centroid, and moment of inertia using the Shoelace Method operates efficiently with a time complexity of $O(n)$, where $\boldsymbol{n}$ represents the number of polygon vertices. This efficiency is due to the method's reliance on single-pass summations over the polygon's vertex coordinates. Since the area, centroid, and moment of inertia computation requires a single pass over the vertices, they also operate in $O(n)$ time. Therefore, the overall computational complexity remains $O(n)$. This linear complexity ensures that the proposed method is highly efficient and scalable, making it suitable for large-scale polygonal computations in engineering applications. Moreover, if the same task has to be done with numerical integration or finite element analysis, then the time complexity will be $O(n^2)$ and $O(n^3)$, respectively **[21]**.

3. RESULT AND DISCUSSION

3.1 Modelling a Shoelace in Python

The important modules we will be using here are the **Shapely** and **Matplotlib.pylab**. The best part is that the **NumPy** module (for creating arrays and for performing numerical computations) is already present in both modules. The purpose of using Shapely is to generate the coordinates of the exterior of the polygon in a way that will close the curve. The **Pylab** will help in plotting the polygon.

First, a function **Polygon_xy()** will be created. This will accept the (x, y) coordinates of different exterior points of the polygon. The function will plot the polygon and return the array of x and y coordinates. The function **Polygon_xy()** is shown below as Program-1:

Program-1

```
def Polygon_xy(list_of_coord):
    """
    Input: List of coordinates of polygon
    Output: Returns x and y as array
    """
    polygon = Polygon(list_of_coord)
    # Getting exterior coordinates
    x, y = polygon.exterior.xy
    pl.fill(x,y)
    pl.xlabel('X')
    pl.ylabel('Y')
    pl.grid()

    return x,y
```

Next, the function of named **Area()** is developed, which accepts the arrays x and y as arguments and returns the area of the polygon based on **Equation (1)**. The function is shown in Program 2 as follows:

Program-2

```
def Area(x,y):
    """
    Input: Arrays x and y
    Output: Area of Polygon
    """
    A = 0
    for i in range(len(x)-1):
        A += x[i]*y[i+1]-y[i]*x[i+1]
    A = abs(A)/2

    return A
```

Then, a function for evaluating the centroid (**Centroid()**) is developed. The function accepts the arrays x and y as arguments and returns the centroids $\bar{x}$ and $\bar{y}$ of the polygon based on **Equation (2)** and **Equation (3)**. The function is shown as Program 3 below:

Program-3

```
def Centroid(x,y):
    """
    Input: Arrays x and y
    Output: Centroids x_bar and y_bar
    """
    x_bar = 0
    y_bar = 0
    for i in range(len(x)-1):
        x_bar += (x[i]+x[i+1])*(x[i]*y[i+1]-y[i]*x[i+1])
        y_bar += (y[i]+y[i+1])*(x[i]*y[i+1]-y[i]*x[i+1])
    x_bar = abs(x_bar/(6*Area(x,y)))
    y_bar = abs(y_bar/(6*Area(x,y)))

    return x_bar, y_bar
```

At last, the function for Moment of Inertia is evaluated, which is named **Mom_Inertia()**. The function is based on **Equation (4)** to **Equation (7)**, which will return the moment of inertia about the axis $x-y$ and as well as about the centroidal axes. The function is shown in Program 4 as follows:

Program-4

```
def Mom_Inertia(x,y):
    Ix  = 0
    Iy  = 0

    for i in range(len(x)-1):
        Iy  += (x[i]**2+x[i]*x[i+1]+x[i+1]**2)*(x[i]*y[i+1]-y[i]*x[i+1])
        Ix  += (y[i]**2+y[i]*y[i+1]+y[i+1]**2)*(x[i]*y[i+1]-y[i]*x[i+1])

    Ix  = abs(Ix/(12))
    Iy  = abs(Iy/(12))

    xc,yc = Centroid(x,y)
    Ixx = Ix-Area(x,y)*yc**2
    Iyy = Iy-Area(x,y)*xc**2

    return Ix,Iy, Ixx, Iyy
```

To give the user the independence of writing the coordinates either in clockwise or counterclockwise direction, the absolute values are returned in case of centroids and moment of inertia.

3.2 Application of Python Functions

In this section, two polygon problems will be solved using the function developed in the previous section.

Problem 1: Figure 4 presents a quadrilateral, specifically a trapezoid, which is bounded by four vertices with different coordinates. The vertices are located at (0,0), (50,0), (40,100), and (10,100), forming a closed polygon. These coordinates define the exact dimensions of the trapezoid, enabling the calculation of its geometric properties. Evaluate the area, centroid, and moment of inertia of the polygon.

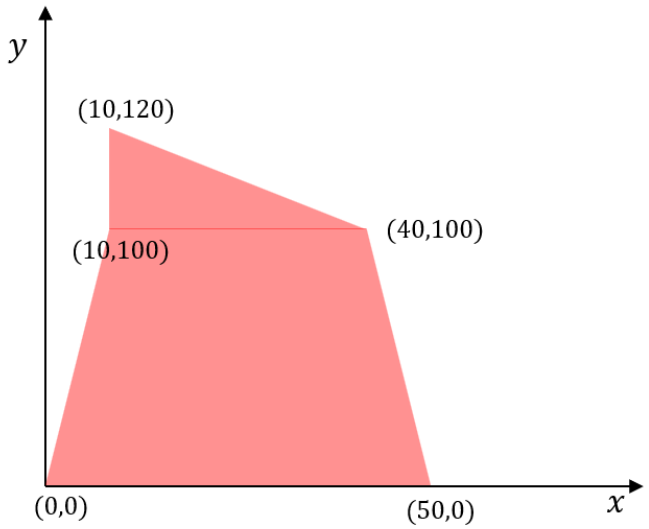

Figure 4. Schematic of Polygon for Problem 1

The first thing that will be done here is to create the list of coordinates, then the **Polygon_xy()** function will be called to get the arrays for x and y. Then, centroids x_c and y_c will be evaluated by the **Centroid() function,** and finally**,** the **Mom_Inertia()** function will be used to get the moment of inertia. The program and outputs are as follows:

Program

```
# Define a polygon (give counterclockwise coordinates)
#list of coordinates
coordinates = [(0, 0), (10, 100), (10, 120), (40, 100), (50, 0)]
# evaluation x and y
x,y = Polygon_xy(coordinates)
# evaluating centroids
x_c,y_c =Centroid(x,y)
# evaluating MOI
Ix, Iy, Ixx, Iyy, =Mom_Inertia(x,y)
# printing results
print(f'Area = {Area(x,y)}')
print(f'x_c = {x_c}, y_c = {y_c}')
print(f'Ix = {Ix}, Iy = {Iy}\nIxx = {Ixx}, Iyy = {Iyy}')
```

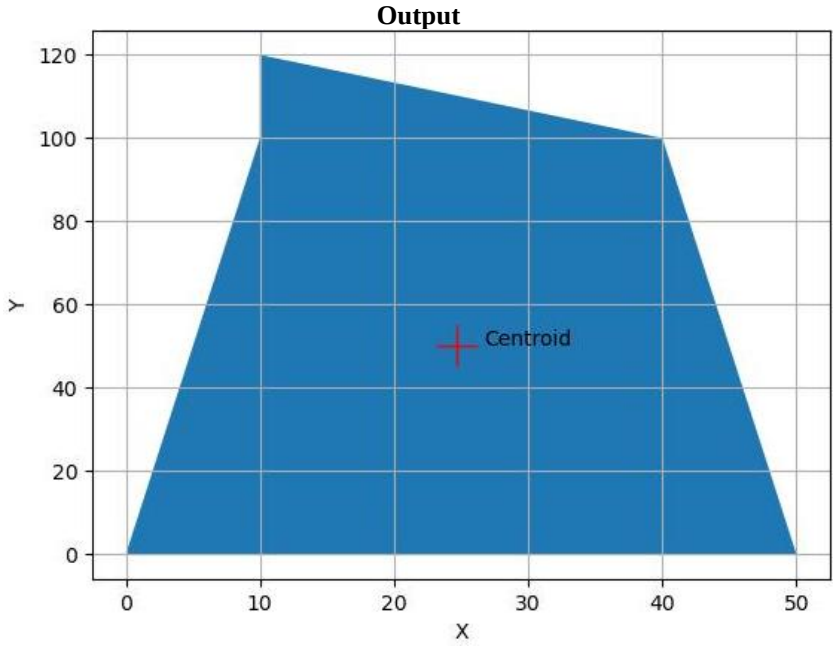

Figure 5. Image Generated by the Python Code for Problem 1

Figure 5 is the image generated by the code as a plot for the problem shown in **Figure 4**. The value of area, centroids, and moment of inertia evaluated by the program are as follows:

```
Area  =   4300.0
x_c = 24.651162790697676,
y_c = 50.07751937984496
Ix  = 15086666.666666666,
Iy  = 3201666.6666666665
Ixx = 4303307.493540052,
Iyy = 588643.410852713
```

Problem 2: **Figure 6** depicts a composite shape constructed from three rectangles and a pentagon, with labeled vertices and their corresponding coordinates. These coordinates define the boundaries and dimensions of each individual shape, enabling calculation of the geometrical parameters. Evaluate the area, centroid, and moment of inertia of the polygon.

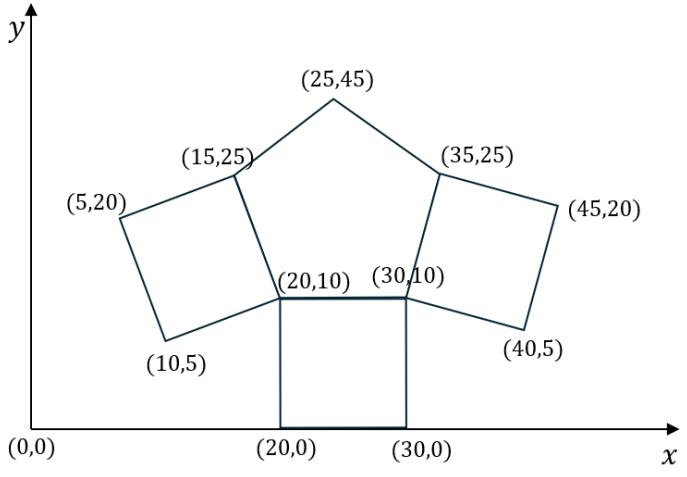

Figure 6. Schematic of Polygon for Problem 2

Following the footsteps of problem 1, the solution and code are as follows:

Program

```
 # Define a polygon (give counterclockwise coordinates if possible)
#list of coordinates
coordinates = [(20,0), (20,10), (10,5), (5,20), (15,25),
               (25,45),(35,25), (45,20), (40,5), (30,10),
               (30,0)]
# evaluation x and y
x,y = Polygon_xy(coordinates)
# evaluating centroids
x_c,y_c =Centroid(x,y)
# evaluating MOI
Ix, Iy, Ixx, Iyy, =Mom_Inertia(x,y)
# printing results
print(f'Area = {Area(x,y)}')
print(f'x_c = {x_c}, y_c = {y_c}')
print(f'Ix = {Ix}, Iy = {Iy}\nIxx = {Ixx}, Iyy = {Iyy}')
```

Output

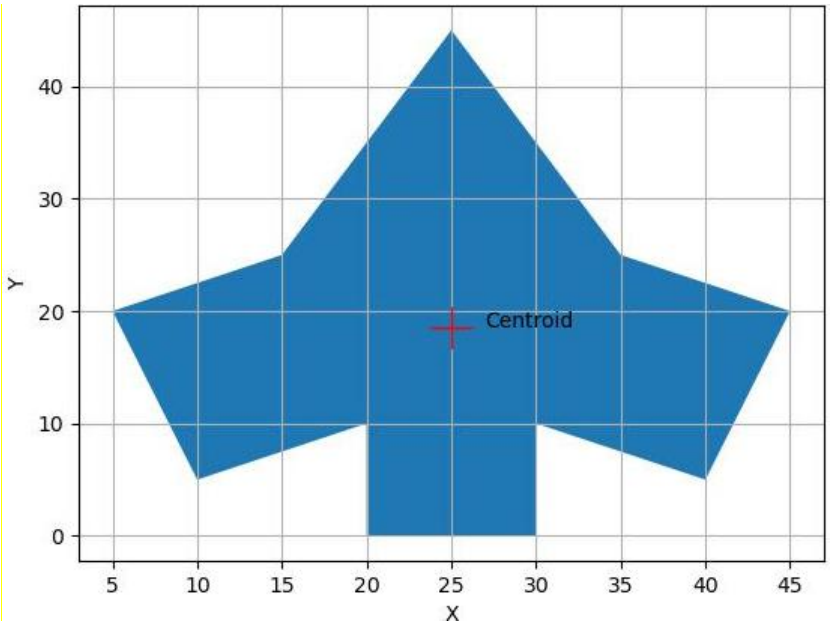

Figure 7. Image Generated by the Python Code for Problem 2

Figure 7 is the image generated by the code as a plot, which is the one shown in **Figure 6**. The value of area, centroids, and moment of inertia evaluated by the program are as follows:

```
Area= 875.0
x_c = 25.0, y_c = 18.523809523809526
Ix  = 374062.5, Iy = 614062.5
Ixx = 73822.42063492053, Iyy = 67187.5
```

Problem 3: **Figure 8** represents an irregular polygon defined by seven vertices with given coordinates. The shape features a combination of straight lines connecting these vertices, forming a closed boundary. The provided coordinates define the exact dimensions and angles within the shape, which would be used to calculate geometric properties. Evaluate the area, centroid, and moment of inertia of this polygon.

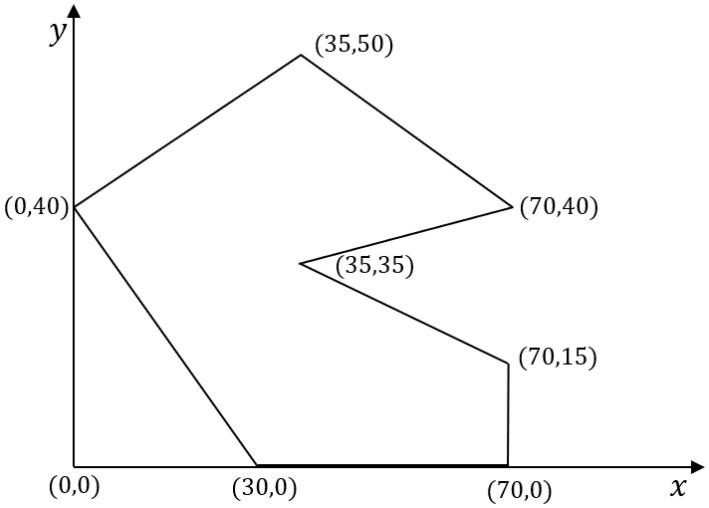

Figure 8. Schematic of Polygon for Problem 3

The code and the final solution to the above problem are as follows:

Program

```
# Define a polygon (give counterclockwise coordinates if possible)
#list of coordinates
coordinates = [(30,0), (0,40), (35,50), (70,40), (35,35), (70, 15), (70,0)]
# evaluation x and y
x,y = Polygon_xy(coordinates)
# evaluating centroids
x_c,y_c =Centroid(x,y)
# evaluating MOI
Ix, Iy, Ixx, Iyy, =Mom_Inertia(x,y)
# printing results
print(f'Area = {Area(x,y)}')
print(f'x_c = {x_c}, y_c = {y_c}')
print(f'Ix = {Ix}, Iy = {Iy}\nIxx = {Ixx}, Iyy = {Iyy}')
```

Output

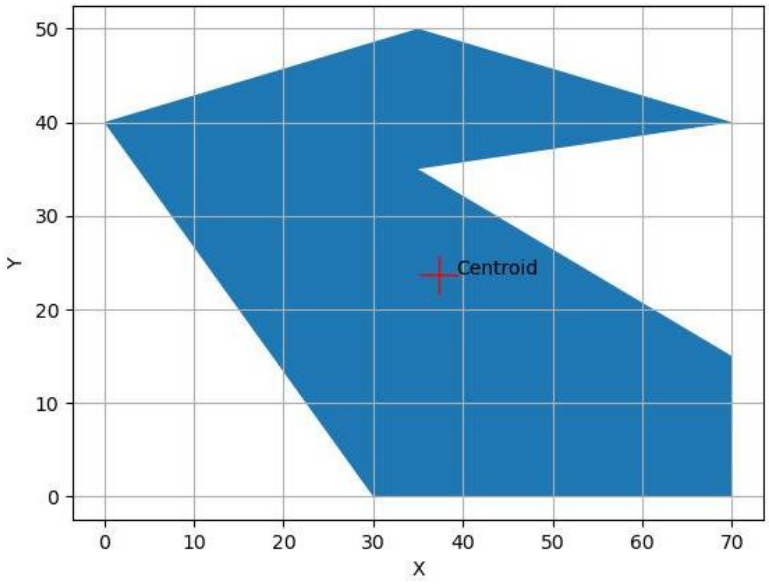

Figure 9. Image Generated by the Python Code for Problem 3

Figure 9 is the image generated by the code as a plot, which is the one shown in **Figure 8**. The value of area, centroids, and moment of inertia evaluated by the program are as follows:

```
Area = 2112.5
x_c = 37.26824457593688, y_c = 23.68836291913215
Ix = 1585989.5833333333, Iy = 3465052.0833333335
Ixx = 400584.4222550953, Iyy = 530954.2447403027
```

4. CONCLUSIONS

The Shoelace Method, which was originally developed for land surveying, proves to be a powerful tool in engineering for calculating geometric properties such as area, centroid, and moments of inertia. By applying this method in Python, the engineers and educators gain a computationally efficient method for solving structural and mechanical problems. The use of Python libraries like Shapely, NumPy, and Matplotlib enables precise calculations and clear visualizations, enhancing both accuracy and comprehension. The examples given in the manuscript have validated the method's applicability to solve complex polygons and emphasize its potential as an educational tool in engineering programs. This integration of computational geometry and programming forwards the algorithmic thinking, which will equip the students and professionals to handle present engineering challenges with confidence.

REFERENCES

[1] V.A. Windarni, A. Setiawan, A. Rahmatalia, COMPARISON OF THE KARNEY POLYGON METHOD AND THE SHOELACE METHOD FOR CALCULATING AREA, MATRIK J. Manajemen, Tek. Inform. Dan Rekayasa Komput. 23 (2023) 39–52. https://doi.org/10.30812/matrik.v23i1.2929.

[2] A. Setiawan, E. Sediyono, CALCULATION OF CENTRAL JAVA PROVINCE REGION AREA USING SHOELACE FORMULA BASED ON THE GADM DATABASE, BAREKENG J. Ilmu Mat. Dan Terap. June 16 (2022) 597–606. https://doi.org/10.1063/5.0103178.

[3] Y. Lee, W. Lim, Shoelace Formula: CONNECTING THE AREA OF A POLYGON AND THE VECTOR CROSS PRODUCT, Math. Teach. 110 (2017) 631–636. https://doi.org/10.5951/mathteacher.110.8.0631.

[4] J. Gechlik, S. High, GAUSS ' S AREA FORMULA FOR IRREGULAR SHAPES, (2024) 12–29.

[5] B. Braden, The Surveyor's Area Formula, Coll. Math. J. 17 (1986) 326–337. https://doi.org/10.1080/07468342.1986.11972974.

[6] B.G. Ryder, M. Lou Soffa, M. Burnett, THE IMPACT OF SOFTWARE ENGINEERING RESEARCH ON MODERN PROGAMMING LANGUAGES, ACM Trans. Softw. Eng. Methodol. 14 (2005) 431–477. https://doi.org/10.1145/1101815.1101818.

[7] P. Dumka, R. Chauhan, D.R. Mishra, F. Shaik, P. Govindaraj, A. Kumar, C. Sonawane, V.I. Velkin, DEVELOPMENT AND IMPLEMENTATION OF A PYTHON FUNCTIONS FOR AUTOMATED CHEMICAL REACTION BALANCING, Indones. J. Electr. Eng. Comput. Sci. 34 (2024) 1557–1565. https://doi.org/10.11591/ijeecs.v34.i3.pp1557-1565.

[8] J. Ranjani, A. Sheela, K. Pandi Meena, COMBINATION OF NUMPY, SCIPY AND MATPLOTLIB/PYLAB-A GOOD ALTERNATIVE METHODOLOGY TO MATLAB-A COMPARATIVE ANALYSIS, in: Proc. 1st Int. Conf. Innov. Inf. Commun. Technol. ICIICT 2019, 2019: pp. 1–5. https://doi.org/10.1109/ICIICT1.2019.8741475.

[9] S. Van Der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: A STRUCTURE FOR EFFICIENT NUMERICAL COMPUTATION, Comput. Sci. Eng. 13 (2011) 22–30. https://doi.org/10.1109/MCSE.2011.37.

[10] P. Dumka, R. Chauhan, A. Singh, G. Singh, D. Mishra, IMPLEMENTATION OF BUCKINGHAM ' S PI THEOREM USING PYTHON, Adv. Eng. Softw. 173 (2022) 103232. https://doi.org/10.1016/j.advengsoft.2022.103232.

[11] P. Mishra, P. Tewari, R. Mishra, Dhananjay, P. Dumka, INTEGRATION BASED ON MONTE CARLO SIMULATION, Int. J. Math. Sci. Comput. 9 (2023) 58–65. https://doi.org/10.5815/ijmsc.2023.03.05.

[12] P. Mishra, P. Tewari, D.R. Mishra, P. Dumka, NUMERICAL MODELLING OF DOUBLE INTEGRATION WITH DIFFERENT DATA SPACING : A PYTHON-BASED APPROACH, 4 (2023) 46–54. https://doi.org/10.30511/MCS.2023.1990951.1115.

[13] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, Am.T. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, SymPy: SYMBOLIC COMPUTING IN PYTHON, PeerJ Comput. Sci. 2017 (2017) 1–27. https://doi.org/10.7717/peerj-cs.103.

[14] M. Cywiak, D. Cywiak, SymPy, in: MULTI-PLATFORM GRAPH. PROGRAM. WITH KIVY BASIC ANAL. PROGRAM. 2D, 3D, Stereosc. Des., Apress, Berkeley, CA, 2021: pp. 173–190. https://doi.org/10.1007/978-1-4842-7113-1_11.

[15] P. Dumka, P.S. Pawar, A. Sauda, G. Shukla, D.R. Mishra, APPLICATION OF HE'S HOMOTOPY AND PERTURBATION METHOD TO SOLVE HEAT TRANSFER EQUATIONS: A PYTHON APPROACH, Adv. Eng. Softw. 170 (2022)

103160. https://doi.org/10.1016/j.advengsoft.2022.103160.
[16] V. Porcu, Matplotlib, in: PYTHON DATA MIN. Quick Syntax Ref., Apress, Berkeley, CA, 2018: pp. 201–234. https://doi.org/10.1007/978-1-4842-4113-4_10.
[17] S. Gillies, THE SHAPELY USER MANUAL, URL Https//Pypi. Org/Project/Shapely (2013).
[18] Y.C. Huei, BENEFITS AND INTRODUCTION TO PYTHON PROGRAMMING FOR FRESHMORE STUDENTS USING INEXPENSIVE ROBOTS, in: Proc. IEEE Int. Conf. Teaching, Assess. Learn. Eng. Learn. Futur. Now, TALE 2014, 2015: pp. 12–17. https://doi.org/10.1109/TALE.2014.7062611.
[19] C. Liu, X. Sun, Z. Li, J. Cui, CALCULATION OF SHIP FLOATING STATE BY QUASI-NEWTON ITERATION METHOD FOR ONBOARD LOADING COMPUTER, Ships Offshore Struct. (2024) 1–10. https://doi.org/10.1080/17445302.2024.2335438.
[20] S.P. Timoshenko, D.H. Young, ENGINEERING MECHANICS: statics, 1937.
[21] E.E. Gdoutos, NUMERICAL METHODS, Courier Corporation, 2020. https://doi.org/10.1007/978-3-030-35098-7_16.