Homework 1

(Due: February 14, 2025)

This homework assignment will get you started with programming in Python + Jupyter Notebook. Submit to gradescope a pdf file of the problem descriptions + your solution.

Question 1: 10 points. Write a Python program that solves for all positive integer pairs, i.e., a, $b \geq 0$,

$$\sqrt{a} + \sqrt{b} = \sqrt{n} \tag{1}$$

where n = 2025.

Hint: 2025 happens to be a perfect square, with the prime factorization of $3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 5$. You can use this fact and a little bit of number theory to see that there will only be 46 solutions (i.e., (a,b) pairs).

Question 2: 10 points. A square of sheet metal having side length 2L cm has four pieces cut out symmetrically from the corners as shown in Figure 1. Assuming that L is a constant and L > 2x, then the remaining metal can be folded into a pyramid having volume:

$$\text{Volume}(x) = \frac{4x^2}{3}\sqrt{(L^2 - 2Lx)} \quad \text{cm}^3. \tag{2}$$

The maximum volume occurs when x = 2L/5 cm.

Write a Python program that sets a value for length L = 10 cm, and then systematically computes and print volumes for appropriate values of x. Organize your output into a tidy table, e.g., something like:

```
--- Pyramid: L = 10.0 cm
+--------+---------------+
| X (cm) | Volume (cm^3) |
+--------+---------------+
|  0.00  |          0.00 |
|  0.50  |          3.16 |
|  1.00  |         11.93 |
|  1.50  |         25.10 |
|  2.00  |         41.31 |
|  2.50  |         58.93 |
```

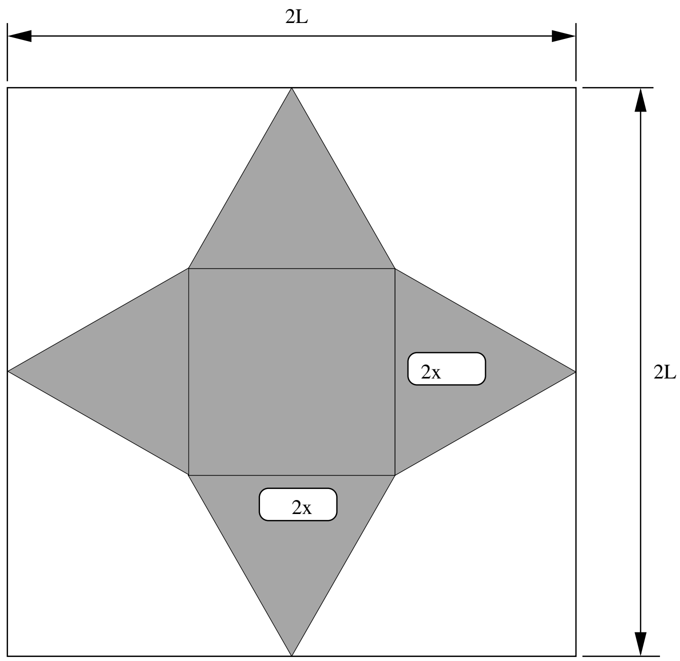


Figure 1: Sheetmetal schematic for a folded pyramid.

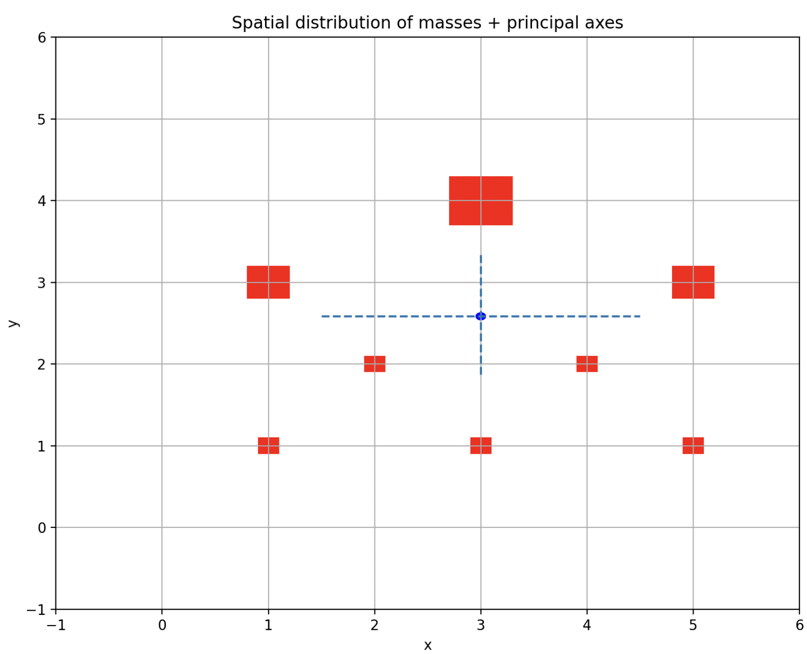


Figure 2: Two-dimensional grid of masses + principal axes.

```
|  3.00  |         75.89 |
|  3.50  |         89.46 |
|  4.00  |         95.41 |
|  4.50  |         85.38 |
|  5.00  |          0.00 |
+--------+---------------+
```

Python has a package called prettytable (i.e., pip3 install prettytable) which you might find useful. A small test program for pretty tables can be found in: python-code.d/basics/TestPrettyTable01.py.

Question 3: 10 points. Figure 2 shows a two-dimensional grid of masses. If the total number of point masses is denoted by N, then the total mass of the grid, M, is given by

$$M = \sum_{i=1}^{N} m_i \tag{3}$$

The coordinates of the grid centroid, $(\bar{x}, \bar{y})$, are defined by:

$$M\bar{x} = \sum_{i=1}^{N} x_i \cdot m_i \quad \text{and} \quad M\bar{y} = \sum_{i=1}^{N} y_i \cdot m_i \tag{4}$$

The moments of inertia about the x- and y-axes are given by:

$$I_{xx} = \sum_{i=1}^{N} y_i^2 \cdot m_i \quad \text{and} \quad I_{yy} = \sum_{i=1}^{N} x_i^2 \cdot m_i \tag{5}$$

respectively. Similarly the cross moment of inertia is given by

$$I_{xy} = \sum_{i=1}^{N} x_i \cdot y_i \cdot m_i \tag{6}$$

With solutions to equations 4 - 6 in hand, the corresponding moments of inertia about the centroid are given by the parallel axes theorem (Google: parallel axis theorem moments of inertia). Finally, the orientation of the principle axes are given by

$$\tan(2\theta) = \left[\frac{2I_{xy}}{I_{xx} - I_{yy}}\right] \tag{7}$$

Now suppose that the (x,y) coordinates and masses are stored in two arrays;

```
mass  = np.array( [ 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 3.0, 2.0 ] );

coord = np.array( [ ( 1.0, 1.0 ),
                    ( 2.0, 2.0 ),
                    ( 3.0, 1.0 ),
                    ( 4.0, 2.0 ),
                    ( 5.0, 1.0 ),
                    ( 5.0, 3.0 ),
                    ( 3.0, 4.0 ),
                    ( 1.0, 3.0 ) ] );
```

Write a Python program to evaluate equations 3 – 7, and create a plot in Python similar to Figure 2. Add the centroid and principal axes (drawn with the appropriate orientation) to your plot.

Question 4: 10 points. Figure 3 shows the cross-section of a T-shaped beam (also called T-beam). Reinforced concrete T-beams are commonly found in buildings and highway bridges.

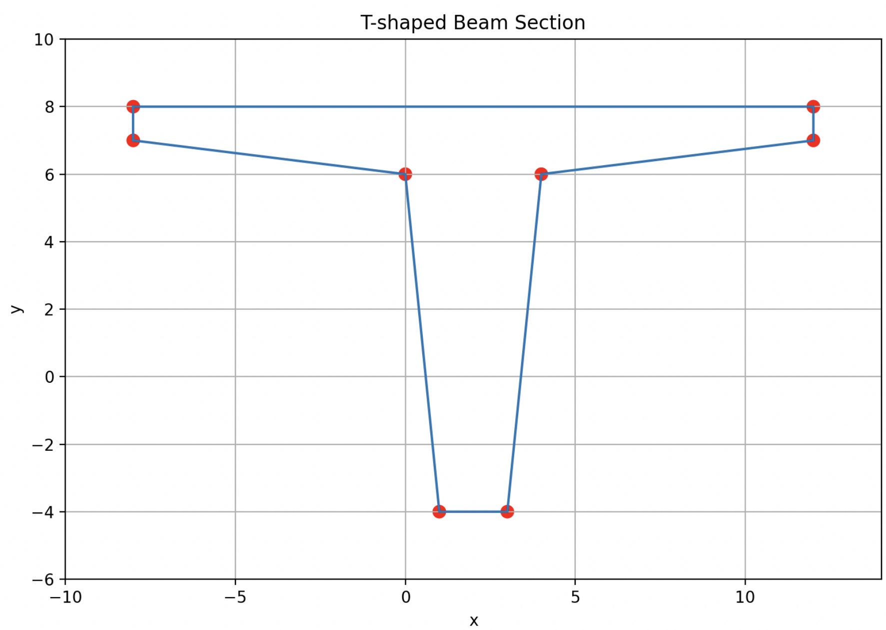


Figure 3: T-shaped beam cross section.

Under service load conditions, T-beams are expected to behave elastically, with very small displacements and no long-term damage. From a mechanics standpoint, the associated elastic analysis procedures require a knowledge of the section area and centroid, and moments of inertia. The purpose of this question is to take a first step toward the development of python code that will compute these section properties automatically. Later on (i.e., homeworks 2 and 3) we will step things up a bit by adding holes to the cross section, and modeling the whole cross section as an object.

Getting Started. The T-beam shown in Figure 3 has (x, y) coordinates stored as two columns of a numpy array:

```
coord = np.array( [ (  -8.0,  8.0 ),
                    (  12.0,  8.0 ),
                    (  12.0,  7.0 ),
                    (   4.0,  6.0 ),
                    (   3.0, -4.0 ),
                    (   1.0, -4.0 ),
                    (   0.0,  6.0 ),
                    (  -8.0 , 7.0 ) ] );
```

Write a Python program that will:

1. Compute and print the minimum and maximum polygon coordinates in both the x and y directions.
2. Compute and print the minimum and maximum distance of the polygon vertices from the coordinate system origin.
3. Create a plot of the T-beam similar to Figure 3.
4. Write functions perimeter() and area() to compute the perimeter and area of the T-beam, respectively.

Hints. For Parts 1 and 2, use the `max()` and `min()` methods in Python. One way of creating Figure 3 is to draw the vertices as circle objects (i.e., from matplotlib.patches import Circle) and the edges as objects of type Line2D (i.e., from matplotlib.lines import Line2D). To compute the perimeter and area of the T-beam, use the fact that the vertices have been specified in a clockwise manner. You should be able to systematically walk around the perimeter of the T-beam and compute the required values of interest.

Question 5: 10 points. Write a Python program that will compute and print a list of (x, y) pairs for:

$$y(x) = \left[\frac{(x^3 - 16x)}{(x-4)(x+5)\sin(x)} \right] \tag{8}$$

over the range $-10 \le x \le 10$ and in intervals of 0.25. You should find that $y(0)$ and $y(4)$ evaluate to not-a-number (NaN), and that $y(-5)$ evaluates to positive infinity.

Python 3 provides remarkably good builtin support for handling of run-time errors. Create a plot of $y(x)$ vs x – you should find that errors will be automatically handled within the matplotlib.pyplot environment.

ENCE 201

February 14, 2025

Problem 1

[4]:
```
import math


n = 2025
sqrt_n = int(math.sqrt(n))


solutions = []


for x in range(sqrt_n + 1):
    y = sqrt_n - x
    a = x**2
    b = y**2
    solutions.append((a, b))


output = "Solutions (a, b) such that sqrt(a) + sqrt(b) = sqrt(2025):\n"
for sol in solutions:
    output += f"{sol}\n"

output += f"\nTotal number of solutions: {len(solutions)}\n"




with open("output.txt", "w") as f:
    f.write(output)
```

```
Solutions (a, b) such that sqrt(a) + sqrt(b) = sqrt(2025):
(0, 2025)
(1, 1936)
(4, 1849)
(9, 1764)
(16, 1681)
(25, 1600)
(36, 1521)
(49, 1444)
```

```
(64, 1369)
(81, 1296)
(100, 1225)
(121, 1156)
(144, 1089)
(169, 1024)
(196, 961)
(225, 900)
(256, 841)
(289, 784)
(324, 729)
(361, 676)
(400, 625)
(441, 576)
(484, 529)
(529, 484)
(576, 441)
(625, 400)
(676, 361)
(729, 324)
(784, 289)
(841, 256)
(900, 225)
(961, 196)
(1024, 169)
(1089, 144)
(1156, 121)
(1225, 100)
(1296, 81)
(1369, 64)
(1444, 49)
(1521, 36)
(1600, 25)
(1681, 16)
(1764, 9)
(1849, 4)
(1936, 1)
(2025, 0)

Total number of solutions: 46
```

Problem 2

[6]:
```
import math
from prettytable import PrettyTable



L = 10.0
```

```python


table = PrettyTable()
table.title = f"Pyramid: L = {L:.1f} cm"
table.field_names = ["X (cm)", "Volume (cm^3)"]


x = 0.0
while x <= 5.0:

    volume = (4 * x**2 / 3) * math.sqrt(L**2 - 2 * L * x)


    table.add_row([f"{x:.2f}", f"{volume:.2f}"])


    x += 0.5


print(table)


with open("output2.txt", "w") as output_file:
    output_file.write(str(table))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[6], line 2
      1 import math
----> 2 from prettytable import PrettyTable
      4 # Set the value for L
      5 L = 10.0

ModuleNotFoundError: No module named 'prettytable'
```

[8]:
```
!pip install prettytable
```

```
Collecting prettytable
  Using cached prettytable-3.14.0-py3-none-any.whl.metadata (30 kB)
Requirement already satisfied: wcwidth in c:\users\fabio\anaconda3\lib\site-
packages (from prettytable) (0.2.5)
Using cached prettytable-3.14.0-py3-none-any.whl (31 kB)
Installing collected packages: prettytable
Successfully installed prettytable-3.14.0
```

[10]:
```python
import math
from prettytable import PrettyTable


L = 10.0


table = PrettyTable()
table.title = f"Pyramid: L = {L:.1f} cm"
table.field_names = ["X (cm)", "Volume (cm^3)"]


x = 0.0
while x <= 5.0:

    volume = (4 * x**2 / 3) * math.sqrt(L**2 - 2 * L * x)


    table.add_row([f"{x:.2f}", f"{volume:.2f}"])


    x += 0.5


print(table)


with open("output2.txt", "w") as output_file:
    output_file.write(str(table))
```

```
+------------------------+
|  Pyramid: L = 10.0 cm  |
+--------+---------------+
| X (cm) | Volume (cm^3) |
+--------+---------------+
|  0.00  |      0.00     |
|  0.50  |      3.16     |
|  1.00  |     11.93     |
|  1.50  |     25.10     |
|  2.00  |     41.31     |
|  2.50  |     58.93     |
|  3.00  |     75.89     |
|  3.50  |     89.46     |
|  4.00  |     95.41     |
|  4.50  |     85.38     |
|  5.00  |      0.00     |
+--------+---------------+
```

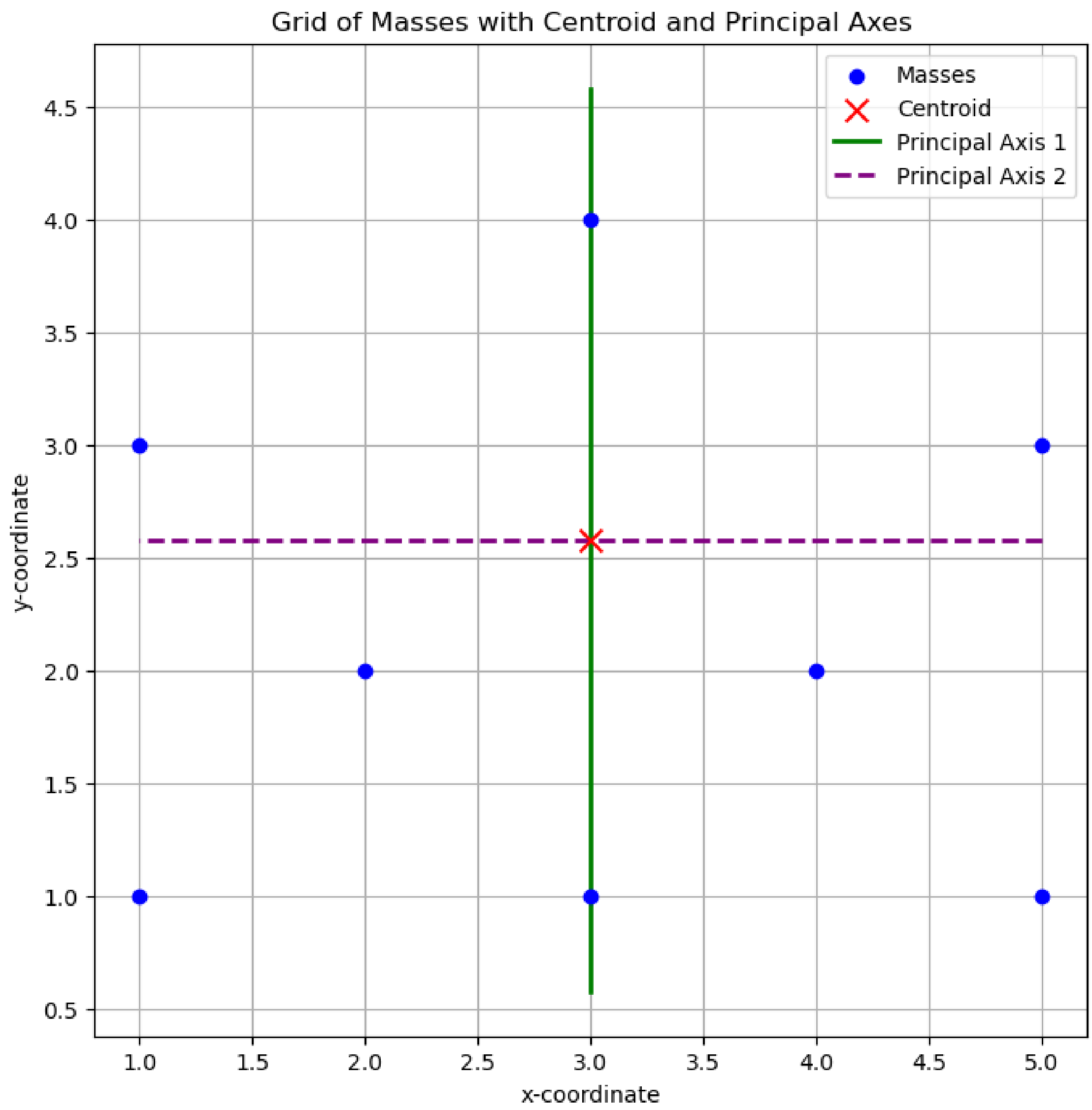
Problem 3

[22]:
```python
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import math


mass = np.array([1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 3.0, 2.0])
coord = np.array([
    (1.0, 1.0),
    (2.0, 2.0),
    (3.0, 1.0),
    (4.0, 2.0),
    (5.0, 1.0),
    (5.0, 3.0),
    (3.0, 4.0),
    (1.0, 3.0)
])


N = len(mass)


x = coord[:, 0]
y = coord[:, 1]


M = np.sum(mass)


x_bar = np.sum(x * mass) / M
y_bar = np.sum(y * mass) / M


Ixx = np.sum((y**2) * mass)
Iyy = np.sum((x**2) * mass)
Ixy = np.sum(x * y * mass)


Ixx_c = Ixx - M * y_bar**2
Iyy_c = Iyy - M * x_bar**2
Ixy_c = Ixy - M * x_bar * y_bar



theta = 0.5 * math.atan2(2 * Ixy_c, (Ixx_c - Iyy_c))
theta_deg = math.degrees(theta)
```

```
print("Computed Properties for the Grid of Masses")
print("==========================================")
print(f"Total number of point masses (N): {N}")
print(f"Total mass (M): {M:.2f}")
print(f"Centroid (x̄, ȳ): ({x_bar:.2f}, {y_bar:.2f})")
print("")
print("Moments about the coordinate axes:")
print(f"Ixx: {Ixx:.2f}")
print(f"Iyy: {Iyy:.2f}")
print(f"Ixy: {Ixy:.2f}")
print("")
print("Moments about the centroid:")
print(f"Ixx_c: {Ixx_c:.2f}")
print(f"Iyy_c: {Iyy_c:.2f}")
print(f"Ixy_c: {Ixy_c:.2f}")
print("")
print("Principal Axes Orientation:")
print(f"Theta (radians): {theta:.4f}")
print(f"Theta (degrees): {theta_deg:.2f}")


fig, ax = plt.subplots(figsize=(8, 8))


ax.scatter(x, y, color='blue', label='Masses', zorder=5)


ax.scatter(x_bar, y_bar, color='red', marker='x', s=100, label='Centroid',␣
↪zorder=6)


range_val = max(x.max() - x.min(), y.max() - y.min()) * 0.5


x1 = x_bar - range_val * math.cos(theta)
x2 = x_bar + range_val * math.cos(theta)
y1 = y_bar - range_val * math.sin(theta)
y2 = y_bar + range_val * math.sin(theta)
ax.plot([x1, x2], [y1, y2], color='green', linestyle='-', linewidth=2,␣
↪label='Principal Axis 1')


theta2 = theta + math.pi/2
x1b = x_bar - range_val * math.cos(theta2)
x2b = x_bar + range_val * math.cos(theta2)
```

```
y1b = y_bar - range_val * math.sin(theta2)
y2b = y_bar + range_val * math.sin(theta2)
ax.plot([x1b, x2b], [y1b, y2b], color='purple', linestyle='--', linewidth=2,
 ↪label='Principal Axis 2')


ax.set_title("Grid of Masses with Centroid and Principal Axes")
ax.set_xlabel("x-coordinate")
ax.set_ylabel("y-coordinate")
ax.legend()
ax.grid(True)
ax.set_aspect('equal', 'box')


plt.savefig("plot3.png", dpi=300)


plt.show()
```

```
Computed Properties for the Grid of Masses
==========================================
Total number of point masses (N): 8
Total mass (M): 12.00
Centroid (x̄, ȳ): (3.00, 2.58)

Moments about the coordinate axes:
Ixx: 95.00
Iyy: 134.00
Ixy: 93.00

Moments about the centroid:
Ixx_c: 14.92
Iyy_c: 26.00
Ixy_c: 0.00

Principal Axes Orientation:
Theta (radians): 1.5708
Theta (degrees): 90.00
```



[2]:

```
sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic
```

```
  Cell In[2], line 1
    sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic
         ^
SyntaxError: invalid syntax
```

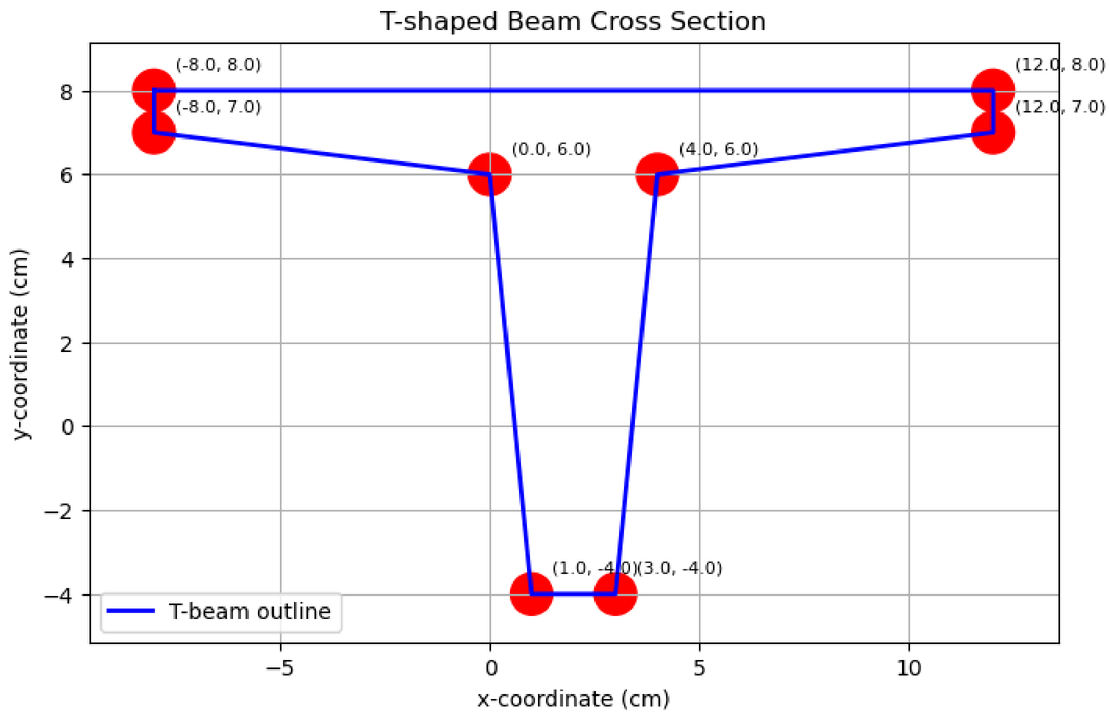
Problem 4

[2]:
```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math


coord = np.array([
    (-8.0, 8.0),
    (12.0, 8.0),
    (12.0, 7.0),
    (4.0, 6.0),
    (3.0, -4.0),
    (1.0, -4.0),
    (0.0, 6.0),
    (-8.0, 7.0)
])


x_coords = coord[:, 0]
y_coords = coord[:, 1]
min_x = np.min(x_coords)
max_x = np.max(x_coords)
min_y = np.min(y_coords)
max_y = np.max(y_coords)

print("Minimum and Maximum Polygon Coordinates:")
print(f"min x: {min_x}, max x: {max_x}")
print(f"min y: {min_y}, max y: {max_y}")



distances = np.sqrt(x_coords**2 + y_coords**2)
min_distance = np.min(distances)
max_distance = np.max(distances)

print("\nMinimum and Maximum Distances from Origin:")
print(f"Minimum distance: {min_distance:.2f}")
print(f"Maximum distance: {max_distance:.2f}")


def perimeter(points):
    """
    Compute the perimeter of a polygon given by an array of points.
    Points are assumed to be in order (clockwise) and the polygon is closed.
    """
    n = len(points)
    perim = 0.0
```

```
    for i in range(n):
        j = (i + 1) % n  # Wrap around to the first point at the end.
        perim += np.linalg.norm(points[j] - points[i])
    return perim

def area(points):
    """
    Compute the area of a polygon using the shoelace formula.
    Points are assumed to be in order (clockwise or counterclockwise).
    """
    n = len(points)
    A = 0.0
    for i in range(n):
        j = (i + 1) % n
        A += points[i, 0] * points[j, 1] - points[j, 0] * points[i, 1]
    return abs(A) / 2

beam_perimeter = perimeter(coord)
beam_area = area(coord)

print("\nT-beam Section Properties:")
print(f"Perimeter: {beam_perimeter:.2f}")
print(f"Area: {beam_area:.2f}")


fig, ax = plt.subplots(figsize=(8, 6))


closed_coord = np.vstack([coord, coord[0]])
ax.plot(closed_coord[:, 0], closed_coord[:, 1], 'b-', lw=2, label='T-beam␣
 ↪outline')


for (x, y) in coord:
    circle = plt.Circle((x, y), 0.5, color='red', fill=True)
    ax.add_patch(circle)
    ax.text(x + 0.5, y + 0.5, f"({x}, {y})", fontsize=8)

ax.set_title("T-shaped Beam Cross Section")
ax.set_xlabel("x-coordinate (cm)")
ax.set_ylabel("y-coordinate (cm)")
ax.legend()
ax.grid(True)
ax.set_aspect('equal', 'box')
plt.show()
```

```
Minimum and Maximum Polygon Coordinates:
min x: -8.0, max x: 12.0
```

```
min y: -4.0, max y: 8.0

Minimum and Maximum Distances from Origin:
Minimum distance: 4.12
Maximum distance: 14.42

T-beam Section Properties:
Perimeter: 60.22
Area: 62.00
```



Problem 5

[4]:
```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt


x_vals = np.arange(-10, 10.25, 0.25)



y_vals = (x_vals**3 - 16*x_vals) / (((x_vals - 4) * (x_vals + 5)) * np.
 ↪sin(x_vals))

# Print the (x, y) pairs
print("List of (x, y) pairs:")
```

```
for x, y in zip(x_vals, y_vals):
    print(f"({x:.2f}, {y})")


x0 = x_vals[np.isclose(x_vals, 0)][0]
x4 = x_vals[np.isclose(x_vals, 4)][0]
xm5 = x_vals[np.isclose(x_vals, -5)][0]
y0 = y_vals[np.isclose(x_vals, 0)][0]
y4 = y_vals[np.isclose(x_vals, 4)][0]
ym5 = y_vals[np.isclose(x_vals, -5)][0]
print("\nSpecific values:")
print(f"y(0)  = {y0}")
print(f"y(4)  = {y4}")
print(f"y(-5) = {ym5}")


plt.figure(figsize=(10, 6))
plt.plot(x_vals, y_vals, 'b-', label='y(x)')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.title('Plot of y(x) for Problem 5')
plt.grid(True)
plt.legend()
plt.show()
```

```
C:\Users\fabio\AppData\Local\Temp\ipykernel_46712\2096160647.py:12:
RuntimeWarning: divide by zero encountered in divide
  y_vals = (x_vals**3 - 16*x_vals) / (((x_vals - 4) * (x_vals + 5)) *
np.sin(x_vals))
C:\Users\fabio\AppData\Local\Temp\ipykernel_46712\2096160647.py:12:
RuntimeWarning: invalid value encountered in divide
  y_vals = (x_vals**3 - 16*x_vals) / (((x_vals - 4) * (x_vals + 5)) *
np.sin(x_vals))

List of (x, y) pairs:
(-10.00, -22.057967530675988)
(-9.75, -36.9387248545078)
(-9.50, -154.50349961198125)
(-9.25, 65.71110704730987)
(-9.00, 27.29797473995988)
(-8.75, 17.741169146355876)
(-8.50, 13.686597135756863)
(-8.25, 11.693488300538773)
(-8.00, 10.78139966293437)
(-7.75, 10.625572884400764)
(-7.50, 11.194030127915003)
(-7.25, 12.723199492617342)
(-7.00, 15.982061156919169)
```

```
(-6.75, 23.56912017091354)
(-6.50, 50.35949206593944)
(-6.25, -339.0676806329957)
(-6.00, -42.94679456705287)
(-5.75, -26.396259969379216)
(-5.50, -23.386331584464184)
(-5.25, -30.561119853598512)
(-5.00, inf)
(-4.75, 14.260084889245721)
(-4.50, 4.603438726520859)
(-4.25, 1.5828866048984291)
(-4.00, -0.0)
(-3.75, -1.3121951668288023)
(-3.50, -3.3258906771305417)
(-3.25, -12.873565423309685)
(-3.00, 10.62925109360578)
(-2.75, 4.002970724260695)
(-2.50, 2.50638231833802)
(-2.25, 1.8402101338859387)
(-2.00, 1.466333560392822)
(-1.75, 1.2312558582644855)
(-1.50, 1.0741192545500624)
(-1.25, 0.9659447033916468)
(-1.00, 0.8912963293335908)
(-0.75, 0.8413978860959618)
(-0.50, 0.811155972251912)
(-0.25, 0.7977577305041587)
(0.00, nan)
(0.25, 0.8180182442947405)
(0.50, 0.8532939448364271)
(0.75, 0.908934840163865)
(1.00, 0.990329254815101)
(1.25, 1.1064457511577046)
(1.50, 1.2724181938516121)
(1.75, 1.5150020642842845)
(2.00, 1.8852860062193426)
(2.25, 2.4928955015696213)
(2.50, 3.62033001537714)
(2.75, 6.2756250709377355)
(3.00, 18.601189413810115)
(3.25, -26.3973109185037)
(3.50, -8.80382826299261)
(3.75, -5.811150024527553)
(4.00, nan)
(4.25, -4.235291186079581)
(4.50, -4.1188662289923474)
(4.75, -4.265837360030771)
(5.00, -4.692758457471326)
```

```
(5.25, -5.51590943699095)
(5.50, -7.053020636584436)
(5.75, -10.260340253878299)
(6.00, -19.521270257751304)
(6.25, -171.6268506907756)
(6.50, 27.58824347960161)
(6.75, 13.722060292930323)
(7.00, 9.766815151450604)
(7.25, 8.08931835872845)
(7.50, 7.356076941201287)
(7.75, 7.180942720568883)
(8.00, 7.4640459204930245)
(8.25, 8.267205380070141)
(8.50, 9.856602875545066)
(8.75, 12.987554446853823)
(9.00, 20.27849552111305)
(9.25, 49.46174390110793)
(9.50, -117.69388841915814)
(9.75, -28.445812434016688)
(10.00, -17.156196968303547)

Specific values:
y(0)  = nan
y(4)  = nan
y(-5) = inf
```

[ ]:

```

```