Presentation Template

M. Rajesh Kumar Reddy
EE24BTECH11043
Dept. of Electrical Engg.,
IIT Hyderabad.

November 5, 2024

1. Problem

2. Solution
   - Section Formula

3. Plot
   - Plot- Using Python-code

4. Codes
   - C-code
   - Python code

Problem Statement

Find the point which divides the line segment joining the points $\mathbf{P}\left(7,-6\right)$ and $\mathbf{Q}\left(3,4\right)$ in the ratio $1:2$ internally and the quadrant in which it lies using section formula.

Section Formula

Let the point which divides **P** and **Q** in the ratio 1 : 2 be **R**. By using Section Formula

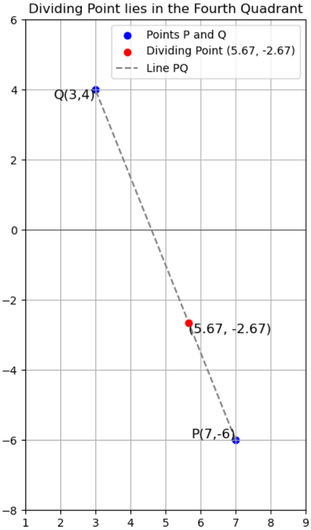
$$\mathbf{R} = \frac{2 \times \mathbf{P} + 1 \times \mathbf{Q}}{1+2} \tag{3.1}$$

$$\mathbf{R}\begin{pmatrix} x \\ y \end{pmatrix} = \frac{2}{3}\begin{pmatrix} 7 \\ -6 \end{pmatrix} + \frac{1}{3}\begin{pmatrix} 3 \\ 4 \end{pmatrix} \tag{3.2}$$

$$\mathbf{R}\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 17 \\ -8 \end{pmatrix} \tag{3.3}$$

$$\mathbf{R}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{17}{3} \\ \frac{-8}{3} \end{pmatrix} \tag{3.4}$$

Plot Points in Graph



C-code

C-code shown below is used to find the point **R** and the quadrant in which it lies.

```c
           #include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "libs/matfun.h"
#include "libs/geofun.h"

// Function to calculate the quadrant based on coordinates
const char* find_quadrant(double x, double y) {
    if (x > 0 && y > 0)
        return "First Quadrant";
    else if (x < 0 && y > 0)
        return "Second Quadrant";
    else if (x < 0 && y < 0)
        return "Third Quadrant";
    else if (x > 0 && y < 0)
        return "Fourth Quadrant";
    else if (x == 0 && y == 0)
```

```
        return "Origin";
    else if (x == 0)
        return "Y-Axis";
    else
        return "X-Axis";
}

int main() {
    // Points P (7, -6) and Q (3, 4)
    double x1 = 7, y1 = -6;
    double x2 = 3, y2 = 4;

    // Ratio m1 : m2 = 1 : 2
    double m1 = 1, m2 = 2;

    // Create matrices for P and Q
    int m = 2, n = 1;
    double **P = createMat(m, n);
    double **Q = createMat(m, n);
    P[0][0] = x1;
```

```
P[1][0] = y1;
Q[0][0] = x2;
Q[1][0] = y2;

// Calculate the point that divides PQ in the ratio 1:2
    using Matadd and Matscale
double **dividing_point = Matadd(Matscale(P, m, n, m2),
    Matscale(Q, m, n, m1), m, n);
dividing_point[0][0] /= (m1 + m2);
dividing_point[1][0] /= (m1 + m2);

// Coordinates of the dividing point
double x = dividing_point[0][0];
double y = dividing_point[1][0];

// Find the quadrant
const char* quadrant = find_quadrant(x, y);

// Write the result to a text file
FILE *fptr = fopen("dividing_point.txt", "w");
```

```
    if (fptr == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fprintf(fptr, "The point that divides the line segment PQ in
        the ratio 1:2 is: (%lf, %lf)\n", x, y);
    fprintf(fptr, "The point lies in the: %s\n", quadrant);

    fclose(fptr);

    // Free allocated memory
    freeMat(P, m);
    freeMat(Q, m);
    freeMat(dividing_point, m);

    printf("Coordinates and quadrant successfully written to
        'dividing_point.txt'\n");

    return 0;
}
```

Python code

Python code reads the point and plots in graph

```python
import matplotlib.pyplot as plt

# Function to read the dividing point from the text file
def read_dividing_point(file_path):
    with open(file_path, ’r’) as file:
        lines = file.readlines()
        # Extract the coordinates from the first line
        coord_line = lines[0].strip().split(": ")[1]
        x, y = map(float, coord_line.strip("()").split(", "))
        return x, y

# Read the dividing point from the file
dividing_point_file = "dividing_point.txt"
x_div, y_div = read_dividing_point(dividing_point_file)

# Original points P and Q
x1, y1 = 7, -6 # Point P
x2, y2 = 3, 4 # Point Q
```

```python
# Function to determine the quadrant of the point
def find_quadrant(x, y):
    if x > 0 and y > 0:
        return "First Quadrant"
    elif x < 0 and y > 0:
        return "Second Quadrant"
    elif x < 0 and y < 0:
        return "Third Quadrant"
    elif x > 0 and y < 0:
        return "Fourth Quadrant"
    elif x == 0 and y == 0:
        return "Origin"
    elif x == 0:
        return "Y-Axis"
    else:
        return "X-Axis"

# Determine the quadrant of the dividing point
quadrant = find_quadrant(x_div, y_div)
```

```
# Plotting the points and the line segment
plt.figure(figsize=(8, 8))
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

# Plot points P and Q
plt.scatter([x1, x2], [y1, y2], color='blue', label='Points P and Q')

# Plot the dividing point
plt.scatter(x_div, y_div, color='red', label=f'Dividing Point ({x_div:.2f}, {y_div:.2f})', zorder=5)

# Draw line PQ
plt.plot([x1, x2], [y1, y2], color='gray', linestyle='--', label='Line PQ')

# Add labels to the points
plt.text(x1, y1, f'P({x1},{y1})', fontsize=12, verticalalignment='bottom', horizontalalignment='right')
plt.text(x2, y2, f'Q({x2},{y2})', fontsize=12, verticalalignment=
    'top', horizontalalignment='right')

plt.text(x_div, y_div, f'({x_div:.2f}, {y_div:.2f})', fontsize
    =12, verticalalignment='top', horizontalalignment='left')

# Display the quadrant
plt.title(f"Dividing Point lies in the {quadrant}")

# Set the x and y limits
plt.xlim(min(x1, x2, x_div) - 2, max(x1, x2, x_div) + 2)
plt.ylim(min(y1, y2, y_div) - 2, max(y1, y2, y_div) + 2)

# Add grid, legend, and show the plot
plt.grid(True)
plt.legend()
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```