1.9.25

M Chanakya Srinivas- EE25BTECH11036

Question

Question 1.9.25: If the point $P(x, y)$ is equidistant from $A(a + b, b - a)$ and $B(a - b, a + b)$, prove that

$$bx = ay$$

Given Data

Define

$$\mathbf{z} = \begin{pmatrix} a \\ b \end{pmatrix}. \tag{1}$$

Then

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \mathbf{z}, \tag{2}$$

$$\mathbf{B} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \mathbf{z}, \tag{3}$$

$$\mathbf{P} = \begin{pmatrix} x \\ y \end{pmatrix}. \tag{4}$$

Notice that

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}^T \quad \Rightarrow \quad \mathbf{B} = \mathbf{A}^T\mathbf{z}. \tag{5}$$

Equidistant Condition

$$\|\mathbf{P} - \mathbf{A}\|^2 = \|\mathbf{P} - \mathbf{B}\|^2 \tag{6}$$

$$(\mathbf{P} - \mathbf{A})^T(\mathbf{P} - \mathbf{A}) = (\mathbf{P} - \mathbf{A}^T)^T(\mathbf{P} - \mathbf{A}^T) \tag{7}$$

Simplification

Using $B = A^T$,

$$2(\mathbf{A}^T - \mathbf{A})^T \mathbf{P} = \mathbf{A}^T \mathbf{A}^T - \mathbf{A}^T \mathbf{A}. \tag{8}$$

But since $\mathbf{A}^T \mathbf{A}$ is symmetric,

$$\mathbf{A}^T \mathbf{A}^T - \mathbf{A}^T \mathbf{A} = 0. \tag{9}$$

Hence

$$(\mathbf{A}^T - \mathbf{A})^T \mathbf{P} = 0. \tag{10}$$

Final Result

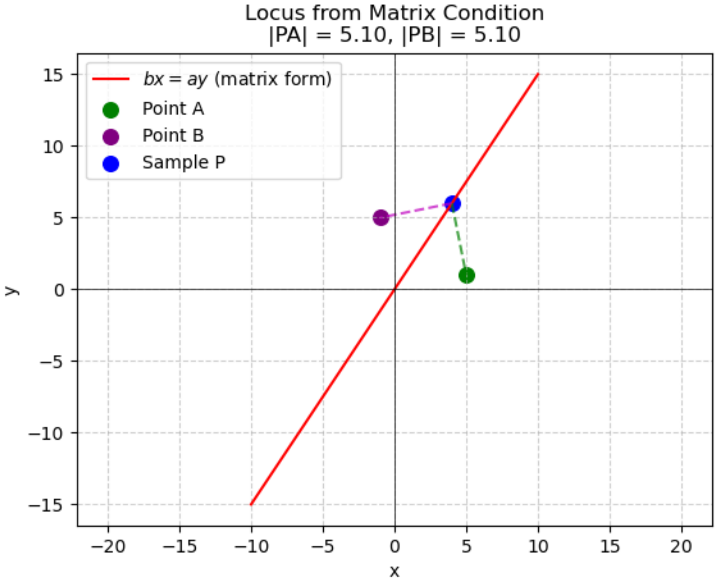
Expanding,

$$(\mathbf{B}-\mathbf{A})^T\mathbf{P} = \left(\begin{pmatrix}0 & -2\\ 2 & 0\end{pmatrix}\mathbf{z}\right)^T\mathbf{P}, \tag{11}$$

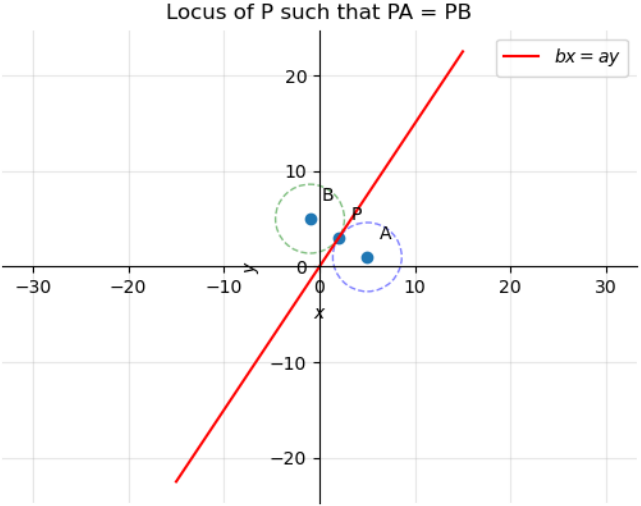
$$= -2bx + 2ay = 0, \tag{12}$$

$$\implies \quad bx = ay \tag{13}$$

Plot by shared output



Direct python code plot



C Code

```
#include <stdio.h>
#include <math.h>

int main() {
    double a, b, x, y;

    // Input parameters
    printf(Enter values of a, b: );
    scanf(%lf %lf, &a, &b);

    printf(Enter point P(x,y): );
    scanf(%lf %lf, &x, &y);

    // Define (B - A)
    double BA[2];
    BA[0] = -2 * b;
    BA[1] = 2 * a;
```

C Code

```
    // Define P
    double P[2];
    P[0] = x;
    P[1] = y;

    // Compute dot product (B - A)^T P
    double dot = BA[0]*P[0] + BA[1]*P[1];

    printf(Matrix form condition:\n);
    printf((B - A)^T P = %lf\n, dot);

    if (fabs(dot) < 1e-6)
        printf(=> Point lies on locus (bx = ay)\n);
    else
        printf(=> Point does NOT lie on locus\n);

    return 0;
}
```

Python Code through shared output

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

# Parameters
a, b = 2, 3 # You can change values here

# Points
A = np.array([a+b, b-a])
B = np.array([a-b, a+b])
P = np.array([4, 6]) # sample point

# Locus line: bx = ay -> y = (b/a) x (if a != 0)
x_vals = np.linspace(-10, 10, 400)
if a != 0:
    y_vals = (b/a) * x_vals
else:
    x_vals = np.full_like(x_vals, 0)
    y_vals = np.linspace(-10, 10, 400)
```

Python Code through shared output

```
# Plot locus line
plt.plot(x_vals, y_vals, 'r-', label=r'$bx = ay$ (matrix form)')

# Plot points A, B, and P
plt.scatter(*A, color='green', s=70, label='Point A')
plt.scatter(*B, color='purple', s=70, label='Point B')
plt.scatter(*P, color='blue', s=70, label='Sample P')

# Draw distances PA and PB
plt.plot([P[0], A[0]], [P[1], A[1]], 'g--', alpha=0.7)
plt.plot([P[0], B[0]], [P[1], B[1]], 'm--', alpha=0.7)

# Distance check
dist_PA = np.linalg.norm(P - A)
dist_PB = np.linalg.norm(P - B)

plt.title(fLocus from Matrix Condition\n|PA| = {dist_PA:.2f}, |PB| = {dist_PB:.2f})
plt.xlabel(x)
```

Python Code through shared output

```
plt.ylabel(y)
plt.axhline(0, color='black', columnwidth=0.5)
plt.axvline(0, color='black', columnwidth=0.5)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.axis(equal)
plt.show()
```

Direct Python code

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt

# local imports
from line.funcs import *
from conics.funcs import circ_gen

# Given values
a, b = 2, 3
A = np.array(([a+b, b-a])).reshape(-1,1) # (5,-3)
B = np.array(([a-b, a+b])).reshape(-1,1) # (-1,7)
```

Direct Python code

```
# Choose a point P on line bx = ay
xP = 2
yP = (b*xP)/a
P = np.array(([xP, yP])).reshape(-1,1)

# Distances
dist_PA = LA.norm(P - A)
dist_PB = LA.norm(P - B)

# Generate locus line bx = ay
xx = np.linspace(-15,15,400)
yy = (b*xx)/a

# Plot locus line
plt.plot(xx, yy, 'r', label=r'$bx=ay$ (matrix form)')

# Plot A, B, P
tri_coords = np.block([[A,B,P]])
plt.scatter(tri_coords[0,:], tri_coords[1,:], s=60)
```

Direct Python code

```
# Labels
vert_labels = ['A','B','P']
for i, txt in enumerate(vert_labels):
    plt.annotate(txt,
                 (tri_coords[0,i], tri_coords[1,i]),
                 textcoords=offset points,
                 xytext=(10,10),
                 ha='center')

# Draw dashed lines PA and PB
plt.plot([A[0,0], P[0,0]], [A[1,0], P[1,0]], 'g--', alpha=0.7)
plt.plot([B[0,0], P[0,0]], [B[1,0], P[1,0]], 'm--', alpha=0.7)

# Circles centered at A and B with radius |PA|
circleA = plt.Circle(A.flatten(), dist_PA, color='blue', fill=
    False, linestyle='--', alpha=0.4)
circleB = plt.Circle(B.flatten(), dist_PB, color='green', fill=
    False, linestyle='--', alpha=0.4)
```

Direct Python code

```
ax = plt.gca()
ax.add_patch(circleA)
ax.add_patch(circleB)

# Axes through origin
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')

plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title(fLocus of P such that PA = PB\n|PA|={dist_PA:.2f}, |PB|={dist_PB:.2f})
plt.legend()
plt.grid(True, alpha=0.3)
plt.axis('equal')
plt.show()
```