2.5.3

Sai Krishna Bakki - EE25BTECH11049

Question

Show that the points (-2, 3), (8, 3), and (6, 7) are the vertices of a right-angled triangle.

Equation

The condition for two sides to be perpendicular :

$$\mathbf{n_1^T n_2} = 0$$

Theoretical Solution

Given

$$\mathbf{A} = \begin{pmatrix} -2 \\ 3 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}, \mathbf{C} = \begin{pmatrix} 6 \\ 7 \end{pmatrix} \tag{1}$$

$$\mathbf{B} - \mathbf{A} = \begin{pmatrix} 10 \\ 0 \end{pmatrix} \tag{2}$$

$$\mathbf{C} - \mathbf{B} = \begin{pmatrix} -2 \\ 4 \end{pmatrix} \tag{3}$$

$$\mathbf{C} - \mathbf{A} = \begin{pmatrix} 8 \\ 4 \end{pmatrix} \tag{4}$$

Theoretical Solution

For a right angle, the dot product of two sides must be zero,

$$(\mathbf{C}-\mathbf{A})^T(\mathbf{C}-\mathbf{B}) = (-2)(8)+(4)(4) = 0 \tag{5}$$

$$(\mathbf{C}-\mathbf{A})^T(\mathbf{B}-\mathbf{A}) = (10)(8)+(0)(4) = 80 \neq 0 \tag{6}$$

$$(\mathbf{B}-\mathbf{A})^T(\mathbf{C}-\mathbf{B}) = (-2)(10)+(4)(0) = -20 \neq 0 \tag{7}$$

Hence $\triangle ABC$ is right angled at $\mathbf{C}$.

C Code

```
double distSq(int x1, int y1, int x2, int y2) {
    long long dx = x2 - x1;
    long long dy = y2 - y1;
    return (double)(dx * dx + dy * dy);
}


int isRightAngled(int x1, int y1, int x2, int y2, int x3, int y3)
    {
  // Calculate the square of the lengths of the three sides
  double d1_sq = distSq(x1, y1, x2, y2);
  double d2_sq = distSq(x2, y2, x3, y3);
  double d3_sq = distSq(x3, y3, x1, y1);
```

C Code

```c
    if (d1_sq == 0 || d2_sq == 0 || d3_sq == 0) {
        return 0;
    }

    if ((d1_sq + d2_sq == d3_sq) ||
        (d1_sq + d3_sq == d2_sq) ||
        (d2_sq + d3_sq == d1_sq)) {
        return 1; // It is a right-angled triangle
    }

    return 0; // It is not a right-angled triangle
}
```

Python Code Through Shared Output

```
import sys
import os
import ctypes
import numpy as np
import numpy.linalg as LA
import scipy.linalg as SA
import matplotlib.pyplot as plt

# --- Helper functions (to make the script self-contained) ---

def line_gen(A, B, n=10):
    Generates n points on a line segment between points A and B.
    return np.array([np.linspace(A[0], B[0], n), np.linspace(A
        [1], B[1], n)])

def label_pts(points, labels):
    Adds text labels to a plot for each point.
    for i, label in enumerate(labels):
```

Python Code Through Shared Output

```
        plt.text(points[0, i] + 0.2, points[1, i] + 0.2, label,
            fontsize=12)

# --- C Library Integration ---

def check_triangle_with_c_lib(p1, p2, p3):

    Loads the C shared library and calls the isRightAngled
        function.

    try:
        # Determine library name based on OS
        lib_name = 'libtriangle.so' if sys.platform.startswith(('
            linux', 'darwin')) else 'triangle.dll'
        lib_path = os.path.join(os.path.dirname(os.path.abspath(
            __file__)), lib_name)

        triangle_lib = ctypes.CDLL(lib_path)
```

Python Code Through Shared Output

```

# Define the C function signature for type safety
isRightAngled = triangle_lib.isRightAngled
isRightAngled.argtypes = [ctypes.c_int, ctypes.c_int,
    ctypes.c_int, ctypes.c_int, ctypes.c_int, ctypes.c_int
    ]
isRightAngled.restype = ctypes.c_int

# Call the C function
result = isRightAngled(p1[0], p1[1], p2[0], p2[1], p3[0],
     p3[1])

# Print the result
print(- * 50)
print(--- C Library Verification ---)
if result == 1:
```

Python Code Through Shared Output

```
            print(fC function returned: 1)
            print(fConclusion: The points form a right-angled triangle.)
        else:
            print(fC function returned: 0)
            print(fConclusion: The points DO NOT form a right-angled triangle.)
        print(- * 50)

    except OSError as e:
        print(fError: Could not load the shared library '{lib_name}'.)
        print(Please compile 'triangle_checker.c' first.)
        print( Linux/macOS: gcc -shared -o libtriangle.so -fPIC triangle_checker.c)
        print( Windows: gcc -shared -o triangle.dll triangle_checker.c)
        print(fDetails: {e})
```

Python Code Through Shared Output

```
        # Exit if the library isn't found, as the check is
            crucial
        sys.exit(1)


# --- Define Triangle Vertices (as per the problem statement) ---
A = np.array([-2, 3])
B = np.array([8, 3])
C = np.array([6, 7])

# Perform the check using the C library before proceeding
check_triangle_with_c_lib(A, B, C)

# Create the vertex matrix G_v (vertices as columns)
G_v = np.array([A, B, C]).T


# --- Matrix Algebra Calculations (from original script) ---
```

Python Code Through Shared Output

```
# Rotation matrix for normals
R_o = np.array([[0, -1], [1, 0]])

# Direction vector circulant matrix
C_m = SA.circulant([1, 0, -1]).T

# Direction vector Matrix (vectors representing sides B-A, C-B, A
    -C)
G_dir = G_v @ C_m

# Normal vector matrix
G_n = R_o @ G_dir

# Find the line constants for the side equations n.T @ x = c
cmat = np.diag(G_n.T @ G_v).reshape(-1, 1)
# print(Line Matrix [nx, ny, c]:\n, np.block([G_n.T, cmat]))

# Get lengths of the sides
```

Python Code Through Shared Output

```
side_lengths = np.linalg.norm(G_dir, axis=0)
a, b, c = side_lengths[1], side_lengths[2], side_lengths[0] # BC,
     AC, AB
# print(fSide lengths squared: AB^2={c**2:.1f}, BC^2={a**2:.1f},
    AC^2={b**2:.1f})



# ---------------- Plotting ------------------------------

# Generate points for each side
line_AB = line_gen(A, B)
line_BC = line_gen(B, C)
line_CA = line_gen(C, A)

# Setup the plot
plt.figure(figsize=(10, 8))
plt.style.use('seaborn-v0_8-whitegrid')
```

Python Code Through Shared Output

```
# Plot the sides
plt.plot(line_AB[0, :], line_AB[1, :], label='Side AB')
plt.plot(line_BC[0, :], line_BC[1, :], label='Side BC')
plt.plot(line_CA[0, :], line_CA[1, :], label='Side AC')

# Plot and label the vertices
plt.plot(G_v[0, :], G_v[1, :], 'o', color='red', markersize=8)
vert_labels = [f'A({A[0]},{A[1]})', f'B({B[0]},{B[1]})', f'C({C
    [0]},{C[1]})']
label_pts(G_v, vert_labels)
```

```
# Set plot properties
plt.title('Triangle Analysis', fontsize=16)
plt.xlabel('$x$-axis', fontsize=12)
plt.ylabel('$y$-axis', fontsize=12)
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

```python
import numpy as np
import numpy.linalg as LA
import scipy.linalg as SA
import matplotlib.pyplot as plt

# Local imports from separate files
from libs.params import *
from libs.funcs import *

# ----------------- Main Script -------------------------------

# --- Define Triangle Vertices (as per the problem statement) ---
A = np.array([-2, 3])
B = np.array([8, 3])
C = np.array([6, 7])
```

Python Code

```
# Perform the right-angle check using the imported function from
    funcs.py
is_right_angled_python(A, B, C)

# Create the vertex matrix G_v (vertices as columns for matrix
    operations)
G_v = np.array([A, B, C]).T

# --- Matrix Algebra Calculations ---

# Direction vector circulant matrix
C_m = SA.circulant([1, 0, -1]).T

# Direction vector Matrix (vectors representing sides B-A, C-B, A
    -C)
G_dir = G_v @ C_m
```

Python Code

```
# Normal vector matrix (uses R_o imported from params.py)
G_n = R_o @ G_dir

# Find the line constants for the side equations n.T @ x = c
cmat = np.diag(G_n.T @ G_v).reshape(-1, 1)

# ---------------- Plotting ----------------------------

# Generate points for each side (uses line_gen from funcs.py)
line_AB = line_gen(A, B)
line_BC = line_gen(B, C)
line_CA = line_gen(C, A)

# Setup the plot
plt.figure(figsize=(10, 8))
plt.style.use('seaborn-v0_8-whitegrid')

# Plot the sides of the triangle
```

Python Code

```python
# Plot and label the vertices (uses label_pts from funcs.py)
plt.plot(G_v[0, :], G_v[1, :], 'o', color='red', markersize=8)
vert_labels = [f'A({A[0]},{A[1]})', f'B({B[0]},{B[1]})', f'C({C
    [0]},{C[1]})']
label_pts(G_v, vert_labels)

# Set plot properties
plt.title('Triangle Analysis', fontsize=16)
plt.xlabel('$x$-axis', fontsize=12)
plt.ylabel('$y$-axis', fontsize=12)
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

Plot By C code and Python Code