4.11.7

Jnanesh Sathisha karmar - EE25BTECH11029

September 29,2025

Question

Find the equation of the planes passing through the intersection of the planes $\mathbf{r}.\left(3\hat{i}+6\hat{j}\right)+12=0$ and $\mathbf{r}.\left(3\hat{i}-\hat{j}+4\hat{k}\right)=0$ are at a unit distance from the origin.

Equation

Given details
Plane 1:

$$\mathbf{r}.\left(3\hat{i}+6\hat{j}\right)+12=0 \tag{1}$$

$$\begin{pmatrix}3 & 6 & 0\end{pmatrix}\mathbf{r}+12=0 \tag{2}$$

$$\text{dividing the equation with 3:} \tag{3}$$

$$\begin{pmatrix}1 & 2 & 0\end{pmatrix}\mathbf{r}+4=0 \tag{4}$$

$$\Rightarrow \mathbf{n}_1^\top\mathbf{r}+d_1=0 \tag{5}$$

$$\text{Therefore :} \tag{6}$$

$$\mathbf{n_1}^\top=\begin{pmatrix}1 & 2 & 0\end{pmatrix} \text{ and } d_1=4 \tag{7}$$

Plane 2:

$$\mathbf{r}.\left(3\hat{i}-\hat{j}+4\hat{k}\right)=0 \tag{8}$$

$$\begin{pmatrix}3 & -1 & 4\end{pmatrix}\mathbf{r}+0=0 \tag{9}$$

$$\text{similarly:} \tag{10}$$

$$\mathbf{n_2}^\top=\begin{pmatrix}3 & -1 & 4\end{pmatrix} \text{ and } d_2=0 \tag{11}$$

Theoretical Solution

plane passing through the intersection of the two given planes can be represented as a linear combination of their equations.

$$\left(\mathbf{n_1}^\top + d_1\right) + \lambda\left(\mathbf{n_2}^\top + d_2\right) = 0 \tag{12}$$

Rearranging them:

$$\left(\mathbf{n_1}^\top + \lambda\mathbf{n_2}^\top\right) + (d_1 + \lambda d_2) = 0 \tag{13}$$

Substituting the values we get:

$$\left(\begin{pmatrix}1 & 2 & 0\end{pmatrix} + \lambda\begin{pmatrix}3 & -1 & 4\end{pmatrix}\right) + (4 + \lambda.0) = 0 \tag{14}$$

$$\begin{pmatrix}1+3\lambda & 2-\lambda & 4\lambda\end{pmatrix}\mathbf{r} + 4 = 0 \tag{15}$$

This is the general equation for any plane passing through the line of intersection. Let's call the new normal vector $\mathbf{n}^\top = \begin{pmatrix}1+3\lambda & 2-\lambda & 4\lambda\end{pmatrix}$ and the constant $d = 4$

Theoretical Solution

Since the plane is at a unit distance from the origin

$$1 = \frac{|d|}{\|\mathbf{n}\|} \tag{16}$$

$$1 = \frac{4}{\sqrt{(1+3\lambda)^2 + (2-\lambda)^2 + (4\lambda)^2}} \tag{17}$$

$$26\lambda^2 + 2\lambda - 11 = 0 \tag{18}$$

On solving the equation we get:

$$\lambda = \frac{-2 \pm 2\sqrt{287}}{52} = \frac{-1 \pm \sqrt{287}}{26} \tag{19}$$

Theoretical Solution

This gives us two possible values for λ, which means there are two planes that satisfy the given conditions.
The final plane equations are:

$$\mathbf{r}\left(\left(23+3\sqrt{287}\right)\hat{i}+\left(53-\sqrt{287}\right)\hat{j}+\left(4\sqrt{287}-4\right)\hat{k}\right)+104=0 \quad (20)$$

$$\mathbf{r}\left(\left(23-3\sqrt{287}\right)\hat{i}+\left(53+\sqrt{287}\right)\hat{j}-\left(4+4\sqrt{287}\right)\hat{k}\right)+104=0 \quad (21)$$

C Code (1) - Function to store the points

```c
#include <stdlib.h>
#include <math.h>

void free_points(float* points) {
    if (points != NULL) {
        free(points);
    }
}
float* generate_plane_1_points(float y_min, float y_max, float
    z_min, float z_max, int num_steps) {
    if (num_steps <= 1) {
        return NULL;
    }
    int total_points = num_steps * num_steps;
    float* points = (float*)malloc(total_points * 3 * sizeof(
        float));
    if (points == NULL) {
        return NULL;
    }
```

C Code (1) - Function to store the points

```c
    float y_step_size = (y_max - y_min) / (num_steps - 1);
    float z_step_size = (z_max - z_min) / (num_steps - 1);
    int index = 0;
    for (int i = 0; i < num_steps; i++) {
        float y = y_min + i * y_step_size;
        for (int j = 0; j < num_steps; j++) {
            float z = z_min + j * z_step_size;
            float x = -2.0f * y - 4.0f;
            points[index++] = x;
            points[index++] = y;
            points[index++] = z;
        }
    }
    return points;
}
float* generate_plane_2_points(float x_min, float x_max, float
    y_min, float y_max, int num_steps) {
    if (num_steps <= 1) {return NULL;}
```

C Code (1) - Function to store the points

```c
    int total_points = num_steps * num_steps;
    float* points = (float*)malloc(total_points * 3 * sizeof(
        float));
    if (points == NULL) {return NULL;}
    float x_step_size = (x_max - x_min) / (num_steps - 1);
    float y_step_size = (y_max - y_min) / (num_steps - 1);
    int index = 0;
    for (int i = 0; i < num_steps; i++) {
        float x = x_min + i * x_step_size;
        for (int j = 0; j < num_steps; j++) {
            float y = y_min + j * y_step_size;
            float z = (-3.0f * x + y) / 4.0f;
            points[index++] = x;
            points[index++] = y;
            points[index++] = z;
        }
    }
    return points;
}
```

C Code (1) - Function to store the points

```c
float* generate_plane_3_points(float x_min, float x_max, float y_min, float y_max, int num_steps) {
    if (num_steps <= 1) {
        return NULL;
    }

    int total_points = num_steps * num_steps;
    float* points = (float*)malloc(total_points * 3 * sizeof(float));
    if (points == NULL) {
        return NULL;
    }

    const float sqrt287 = sqrtf(287.0f);
    const float A = 23.0f + 3.0f * sqrt287;
    const float B = 53.0f - sqrt287;
    const float C = 4.0f * sqrt287 - 4.0f;
    const float D = 104.0f;
```

C Code (1) - Function to store the points

```c
    float x_step_size = (x_max - x_min) / (num_steps - 1);
    float y_step_size = (y_max - y_min) / (num_steps - 1);

    int index = 0;
    for (int i = 0; i < num_steps; i++) {
        float x = x_min + i * x_step_size;
        for (int j = 0; j < num_steps; j++) {
            float y = y_min + j * y_step_size;
            float z = (-A * x - B * y - D) / C;

            points[index++] = x;
            points[index++] = y;
            points[index++] = z;
        }
    }
    return points;
}
```

C Code (1) - Function to store the points

```c
float* generate_plane_4_points(float x_min, float x_max, float
    y_min, float y_max, int num_steps) {
    if (num_steps <= 1) {
        return NULL;
    }

    int total_points = num_steps * num_steps;
    float* points = (float*)malloc(total_points * 3 * sizeof(
        float));
    if (points == NULL) {
        return NULL;
    }

    const float sqrt287 = sqrtf(287.0f);
    const float A = 23.0f - 3.0f * sqrt287;
    const float B = 53.0f + sqrt287;
    const float C = -4.0f - 4.0f * sqrt287;
    const float D = 104.0f;
```

C Code (1) - Function to store the points

```c
    float x_step_size = (x_max - x_min) / (num_steps - 1);
    float y_step_size = (y_max - y_min) / (num_steps - 1);

    int index = 0;
    for (int i = 0; i < num_steps; i++) {
        float x = x_min + i * x_step_size;
        for (int j = 0; j < num_steps; j++) {
            float y = y_min + j * y_step_size;
            float z = (-A * x - B * y - D) / C;

            points[index++] = x;
            points[index++] = y;
            points[index++] = z;
        }
    }
    return points;
}
```

Python Code - Using Shared Object

```
import ctypes
import numpy as np
import matplotlib.pyplot as plt
NUM_STEPS = 50
PLOT_RANGE = 10.0
plane_lib = ctypes.CDLL(./planes.so)

float_ptr = ctypes.POINTER(ctypes.c_float)

plane_lib.free_points.argtypes = [float_ptr]
plane_lib.free_points.restype = None
plane_lib.generate_plane_1_points.argtypes = [ctypes.c_float,
    ctypes.c_float, ctypes.c_float, ctypes.c_float, ctypes.c_int]
plane_lib.generate_plane_1_points.restype = float_ptr

plane_lib.generate_plane_2_points.argtypes = [ctypes.c_float,
    ctypes.c_float, ctypes.c_float, ctypes.c_float, ctypes.c_int]
plane_lib.generate_plane_2_points.restype = float_ptr
```

Python Code - Using Shared Object

```
plane_lib.generate_plane_3_points.argtypes = [ctypes.c_float,
    ctypes.c_float, ctypes.c_float, ctypes.c_float, ctypes.c_int]
plane_lib.generate_plane_3_points.restype = float_ptr

plane_lib.generate_plane_4_points.argtypes = [ctypes.c_float,
    ctypes.c_float, ctypes.c_float, ctypes.c_float, ctypes.c_int]
plane_lib.generate_plane_4_points.restype = float_ptr



def get_plane_data(c_function, *args):
    points_ptr = None
    points_data = None
    total_points = NUM_STEPS * NUM_STEPS
```

Python Code - Using Shared Object

```python
    try:
        points_ptr = c_function(*args)
        if not points_ptr:
            raise MemoryError("C function failed to allocate memory or returned NULL.")
        points_np_view = np.ctypeslib.as_array(points_ptr, shape=(total_points * 3,))
        points_data = np.copy(points_np_view)
    finally:
        if points_ptr:
            plane_lib.free_points(points_ptr)
    return points_data
print("Generating points for all planes...")
plane1_data_flat = get_plane_data(
    plane_lib.generate_plane_1_points,
    -PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS
)
```

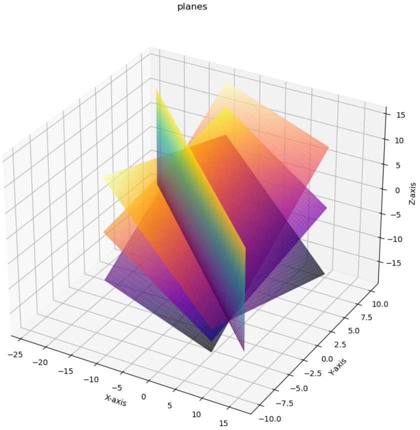
Python Code - Using Shared Object

```
plane2_data_flat = get_plane_data(
    plane_lib.generate_plane_2_points,
    -PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
plane3_data_flat = get_plane_data(
    plane_lib.generate_plane_3_points,
    -PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
plane4_data_flat = get_plane_data(
    plane_lib.generate_plane_4_points,
    -PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
print(f"Generated {plane1_data_flat.shape[0] // 3} points per
    plane and freed C memory.")
def reshape_for_plot(flat_data):
    points = flat_data.reshape(NUM_STEPS * NUM_STEPS, 3)
    X = points[:, 0].reshape(NUM_STEPS, NUM_STEPS)
    Y = points[:, 1].reshape(NUM_STEPS, NUM_STEPS)
    Z = points[:, 2].reshape(NUM_STEPS, NUM_STEPS)
    return X, Y, Z
```

Python Code - Using Shared Object

```
X1, Y1, Z1 = reshape_for_plot(plane1_data_flat)
X2, Y2, Z2 = reshape_for_plot(plane2_data_flat)
X3, Y3, Z3 = reshape_for_plot(plane3_data_flat)
X4, Y4, Z4 = reshape_for_plot(plane4_data_flat)
print("Plotting the planes...")
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X1, Y1, Z1, cmap='viridis', alpha=0.7)
ax.plot_surface(X2, Y2, Z2, cmap='plasma', alpha=0.7)
ax.plot_surface(X3, Y3, Z3, cmap='inferno', alpha=0.7)
ax.plot_surface(X4, Y4, Z4, cmap='magma', alpha=0.7)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title('planes')
plt.savefig("./figs/planes.png")
subprocess.run(shlex.split('termux-open ../figs/planes.png'))
plt.show()
```

Plot-Using Both C and Python



Python Code

```
import numpy as np
import matplotlib.pyplot as plt
NUM_STEPS = 50
PLOT_RANGE = 10.0
def generate_plane_1_points_py(y_min, y_max, z_min, z_max,
    num_steps):
    y = np.linspace(y_min, y_max, num_steps)
    z = np.linspace(z_min, z_max, num_steps)
    Y, Z = np.meshgrid(y, z)
    X = -2.0 * Y - 4.0
    return X, Y, Z
def generate_plane_2_points_py(x_min, x_max, y_min, y_max,
    num_steps):
    x = np.linspace(x_min, x_max, num_steps)
    y = np.linspace(y_min, y_max, num_steps)
    X, Y = np.meshgrid(x, y)
    Z = (-3.0 * X + Y) / 4.0
    return X, Y, Z
```

Python Code

```
def generate_plane_3_points_py(x_min, x_max, y_min, y_max,
    num_steps):
    sqrt287 = np.sqrt(287.0)
    A = 23.0 + 3.0 * sqrt287
    B = 53.0 - sqrt287
    C = 4.0 * sqrt287 - 4.0
    D = 104.0
    x = np.linspace(x_min, x_max, num_steps)
    y = np.linspace(y_min, y_max, num_steps)
    X, Y = np.meshgrid(x, y)
    Z = (-A * X - B * Y - D) / C
    return X, Y, Z
def generate_plane_4_points_py(x_min, x_max, y_min, y_max,
    num_steps):
    sqrt287 = np.sqrt(287.0)
    A = 23.0 - 3.0 * sqrt287
    B = 53.0 + sqrt287
    C = -4.0 - 4.0 * sqrt287
    D = 104.0
```

Python Code

```
    x = np.linspace(x_min, x_max, num_steps)
    y = np.linspace(y_min, y_max, num_steps)
    X, Y = np.meshgrid(x, y)
    Z = (-A * X - B * Y - D) / C
    return X, Y, Z
print("Generating points for all planes using Python...")

X1, Y1, Z1 = generate_plane_1_points_py(-PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
X2, Y2, Z2 = generate_plane_2_points_py(-PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
X3, Y3, Z3 = generate_plane_3_points_py(-PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)
X4, Y4, Z4 = generate_plane_4_points_py(-PLOT_RANGE, PLOT_RANGE, -PLOT_RANGE, PLOT_RANGE, NUM_STEPS)

print(f"Generated points for {NUM_STEPS*NUM_STEPS} grid locations per plane.")
```

Python Code

```
print("Plotting the planes...")
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(projection='3d')

ax.plot_surface(X1, Y1, Z1, cmap='viridis', alpha=0.7)
ax.plot_surface(X2, Y2, Z2, cmap='plasma', alpha=0.7)
ax.plot_surface(X3, Y3, Z3, cmap='inferno', alpha=0.7)
ax.plot_surface(X4, Y4, Z4, cmap='magma', alpha=0.7)

ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title('Planes')
plt.savefig("./figs/planes2.png")
subprocess.run(shlex.split('termux-open ../figs/planes2.png'))
plt.show()
```

Plot-Using only Python