Presentation - Matgeo

Aryansingh Sonaye
AI25BTECH11032
EE1030 - Matrix Theory

September 9, 2025

Problem Statement

If

$$\mathbf{a} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \tag{1.1}$$

find $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$.

Description of Variables used

Input variable	**Value**
a	$\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$
b	$\begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$
c	$\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}$

Table

Theoretical Solution

We are asked to compute:

$$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}). \tag{2.1}$$

Step 1 — Vectors as column matrices

$$\mathbf{a} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}. \tag{2.2}$$

Step 2 — Form the Gram matrix

The Gram matrix is

$$G = \begin{pmatrix} \mathbf{a}^T\mathbf{a} & \mathbf{a}^T\mathbf{b} & \mathbf{a}^T\mathbf{c} \\ \mathbf{b}^T\mathbf{a} & \mathbf{b}^T\mathbf{b} & \mathbf{b}^T\mathbf{c} \\ \mathbf{c}^T\mathbf{a} & \mathbf{c}^T\mathbf{b} & \mathbf{c}^T\mathbf{c} \end{pmatrix}. \tag{2.3}$$

Theoretical Solution

Compute each entry:

$$\mathbf{a}^T\mathbf{a} = 14, \qquad \mathbf{b}^T\mathbf{b} = 6, \qquad \mathbf{c}^T\mathbf{c} = 14, \tag{2.4}$$

$$\mathbf{a}^T\mathbf{b} = 3, \qquad \mathbf{b}^T\mathbf{c} = 1, \qquad \mathbf{c}^T\mathbf{a} = 13. \tag{2.5}$$

Thus,

$$G = \begin{pmatrix} 14 & 3 & 13 \\ 3 & 6 & 1 \\ 13 & 1 & 14 \end{pmatrix}. \tag{2.6}$$

Step 3 — Gram determinant identity

We know

$$\det(G) = \left(\det([\mathbf{a}\ \mathbf{b}\ \mathbf{c}])\right)^2 \tag{2.7}$$

$$= \left(\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})\right)^2. \tag{2.8}$$

Theoretical Solution

Direct computation gives

$$\det(G) = 100. \tag{2.9}$$

Hence

$$|\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})| = \sqrt{100} = 10. \tag{2.10}$$

Step 4 — Find the sign

Form the matrix

$$A = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}] = \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & 1 \\ 3 & 1 & 2 \end{pmatrix}. \tag{2.11}$$

Then

$$\det(A) = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}). \tag{2.12}$$

Theoretical Solution

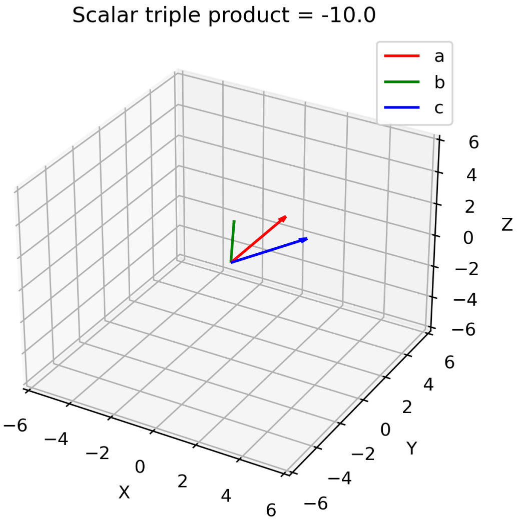
Compute:

$$\det(A) = -10. \tag{2.13}$$

Final Answer

$$\boxed{\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = -10} \tag{2.14}$$

Plot



Code - C

```
#include <stdio.h>

// Dot product of two 3D vectors
double dot_product(double a[3], double b[3]) {
    return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
}

// Build Gram matrix from three 3D vectors
void gram_matrix(double a[3], double b[3], double c[3], double G[3][3])
    {
    G[0][0] = dot_product(a,a);
    G[0][1] = dot_product(a,b);
    G[0][2] = dot_product(a,c);

    G[1][0] = dot_product(b,a);
    G[1][1] = dot_product(b,b);
    G[1][2] = dot_product(b,c);
```

Code - C

```
    G[2][0] = dot_product(c,a);
    G[2][1] = dot_product(c,b);
    G[2][2] = dot_product(c,c);
}

// Determinant of a 3x3 matrix
double det3(double M[3][3]) {
    return M[0][0]*(M[1][1]*M[2][2] - M[1][2]*M[2][1])
        - M[0][1]*(M[1][0]*M[2][2] - M[1][2]*M[2][0])
        + M[0][2]*(M[1][0]*M[2][1] - M[1][1]*M[2][0]);
}

}
```

Code - Python(with shared C code)

The code to obtain the required plot is

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

# --- Load compiled C library ---
lib = ctypes.CDLL("./libgram.so")

# Define ctypes array types
DoubleArray3 = ctypes.c_double * 3
DoubleMatrix3 = (DoubleArray3) * 3

# Function signatures
lib.dot_product.argtypes = [DoubleArray3, DoubleArray3]
lib.dot_product.restype = ctypes.c_double
```

Code - Python(with shared C code)

```
lib.gram_matrix.argtypes = [DoubleArray3, DoubleArray3, DoubleArray3,
    DoubleMatrix3]
lib.det3.argtypes = [DoubleMatrix3]
lib.det3.restype = ctypes.c_double

# --- Define vectors ---
a = np.array([2.0, 1.0, 3.0])
b = np.array([-1.0, 2.0, 1.0])
c = np.array([3.0, 1.0, 2.0])

# Convert to C arrays
A = DoubleArray3(*a)
B = DoubleArray3(*b)
C = DoubleArray3(*c)
```

Code - Python(with shared C code)

```
# --- Step 1: Build Gram matrix ---
G = DoubleMatrix3()
lib.gram_matrix(A, B, C, G)

# --- Step 2: Compute det(G) ---
detG = lib.det3(G)
print("det(G) =", detG)

# --- Step 3: Magnitude of scalar triple product ---
magnitude = math.sqrt(detG)
print("|a . (b x c)| =", magnitude)
```

Code - Python(with shared C code)

```
# --- Step 4: Compute sign using det(A) ---
A_mat = np.column_stack((a, b, c)) # matrix [a b c]
sign_val = np.linalg.det(A_mat) # NumPy to check sign
scalar_triple = math.copysign(magnitude, sign_val)
print("a . (b x c) =", scalar_triple)

# Image generation
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

def draw_vec(v, color, label):
    ax.quiver(0, 0, 0, v[0], v[1], v[2],
              color=color, arrow_length_ratio=0.1, label=label)
```

Code - Python(with shared C code)

```
# Draw just the vectors
draw_vec(a, 'r', 'a')
draw_vec(b, 'g', 'b')
draw_vec(c, 'b', 'c')

# Set axes limits
lim = 6
ax.set_xlim([-lim, lim])
ax.set_ylim([-lim, lim])
ax.set_zlim([-lim, lim])

# Labels
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.legend()
```

Code - Python(with shared C code)

```
plt.title(f"Scalar triple product = {scalar_triple}")
plt.savefig("gram_triple_product.png", dpi=300)
plt.show()
```

Code - Python only

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

# --- Define vectors ---
a = np.array([2.0, 1.0, 3.0])
b = np.array([-1.0, 2.0, 1.0])
c = np.array([3.0, 1.0, 2.0])

# --- Step 1: Build Gram matrix ---
G = np.array([
    [np.dot(a, a), np.dot(a, b), np.dot(a, c)],
    [np.dot(b, a), np.dot(b, b), np.dot(b, c)],
    [np.dot(c, a), np.dot(c, b), np.dot(c, c)]
])
```

Code - Python only

```
print("Gram matrix:\n", G)

# --- Step 2: Compute det(G) ---
detG = np.linalg.det(G)
print("det(G) =", detG)

# --- Step 3: Magnitude of scalar triple product ---
magnitude = math.sqrt(detG)
print("|a . (b x c)| =", magnitude)

# --- Step 4: Compute sign using det(A) ---
A_mat = np.column_stack((a, b, c)) # matrix [a b c]
sign_val = np.linalg.det(A_mat)
scalar_triple = math.copysign(magnitude, sign_val)

print("a . (b x c) =", scalar_triple)
```

Code - Python only

```
# Image Generation
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

def draw_vec(v, color, label):
    ax.quiver(0, 0, 0, v[0], v[1], v[2],
              color=color, arrow_length_ratio=0.1, label=label)

# Draw the vectors
draw_vec(a, 'r', 'a')
draw_vec(b, 'g', 'b')
draw_vec(c, 'b', 'c')
```

Code - Python only

```
# Set axes limits
lim = 6
ax.set_xlim([-lim, lim])
ax.set_ylim([-lim, lim])
ax.set_zlim([-lim, lim])

# Labels
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.legend()

plt.title(f"Scalar triple product = {scalar_triple}")
plt.savefig("gram_triple_product_python.png", dpi=300)
plt.show()
```