2.10.71

EE25BTECH11043 - Nishid Khandagre

September 27, 2025

Question

If the vectors $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ are not coplanar, then prove that the vector $\left(\mathbf{a} \times \mathbf{b}\right) \times \left(\mathbf{c} \times \mathbf{d}\right) + \left(\mathbf{a} \times \mathbf{c}\right) \times \left(\mathbf{d} \times \mathbf{b}\right) + \left(\mathbf{a} \times \mathbf{d}\right) \times \left(\mathbf{b} \times \mathbf{c}\right)$ is parallel to $\mathbf{a}$.

Theoretical Solution

The vector triple product $\mathbf{A} \times \left(\mathbf{B} \times \mathbf{C}\right)$ can be written as:

$$\mathbf{A} \times \left(\mathbf{B} \times \mathbf{C}\right) = \mathbf{B}\left(\mathbf{A}^\top \mathbf{C}\right) - \mathbf{C}\left(\mathbf{A}^\top \mathbf{B}\right) \tag{1}$$

Also, we know

$$\left(\mathbf{A} \times \mathbf{B}\right) \times \mathbf{C} = -\mathbf{C} \times \left(\mathbf{A} \times \mathbf{B}\right) \tag{2}$$

Theoretical Solution

$$\left(\mathbf{A} \times \mathbf{B}\right) \times \left(\mathbf{C} \times \mathbf{D}\right) = \left(\mathbf{A}^\top \left(\mathbf{C} \times \mathbf{D}\right)\right) \mathbf{B} - \left(\mathbf{B}^\top \left(\mathbf{C} \times \mathbf{D}\right)\right) \mathbf{A} \tag{3}$$

By using (3)

$$\left(\mathbf{a} \times \mathbf{b}\right) \times \left(\mathbf{c} \times \mathbf{d}\right) = \left(\mathbf{a}^\top \left(\mathbf{c} \times \mathbf{d}\right)\right) \mathbf{b} - \left(\mathbf{b}^\top \left(\mathbf{c} \times \mathbf{d}\right)\right) \mathbf{a} \tag{4}$$

$$= \left[\mathbf{a}\ \mathbf{c}\ \mathbf{d}\right] \mathbf{b} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right] \mathbf{a} \tag{5}$$

Theoretical Solution

$$
\begin{aligned}
\left(\mathbf{a} \times \mathbf{c}\right) \times \left(\mathbf{d} \times \mathbf{b}\right) &= \left(\mathbf{a}^\top \left(\mathbf{d} \times \mathbf{b}\right)\right)\mathbf{c} - \left(\mathbf{c}^\top \left(\mathbf{d} \times \mathbf{b}\right)\right)\mathbf{a} && (6)\\
&= \left[\mathbf{a}\ \mathbf{d}\ \mathbf{b}\right]\mathbf{c} - \left[\mathbf{c}\ \mathbf{d}\ \mathbf{b}\right]\mathbf{a} && (7)\\
&= -\left[\mathbf{a}\ \mathbf{b}\ \mathbf{d}\right]\mathbf{c} + \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} && (8)
\end{aligned}
$$

Theoretical Solution

$$\left(\mathbf{a} \times \mathbf{d}\right) \times \left(\mathbf{b} \times \mathbf{c}\right) = \left(\mathbf{a}^\top \left(\mathbf{b} \times \mathbf{c}\right)\right) \mathbf{d} - \left(\mathbf{d}^\top \left(\mathbf{b} \times \mathbf{c}\right)\right) \mathbf{a} \tag{9}$$

$$= \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right] \mathbf{d} - \left[\mathbf{d}\ \mathbf{b}\ \mathbf{c}\right] \mathbf{a} \tag{10}$$

$$= \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right] \mathbf{d} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right] \mathbf{a} \tag{11}$$

Theoretical Solution

On adding :

$$\left(\mathbf{a} \times \mathbf{b}\right) \times \left(\mathbf{c} \times \mathbf{d}\right) + \left(\mathbf{a} \times \mathbf{c}\right) \times \left(\mathbf{d} \times \mathbf{b}\right) + \left(\mathbf{a} \times \mathbf{d}\right) \times \left(\mathbf{b} \times \mathbf{c}\right) \tag{12}$$

$$= \left[\mathbf{a}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{b} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} - \left[\mathbf{a}\ \mathbf{b}\ \mathbf{d}\right]\mathbf{c} + \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} + \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right]\mathbf{d} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} \tag{13}$$

$$= \left[\mathbf{a}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{b} - \left[\mathbf{a}\ \mathbf{b}\ \mathbf{d}\right]\mathbf{c} + \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right]\mathbf{d} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} \tag{14}$$

Theoretical Solution

Using the expansion of a vector $\mathbf{a}$ in terms of non-coplanar vectors $\mathbf{b}, \mathbf{c}, \mathbf{d}$:

$$\mathbf{a}\left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right] = \left[\mathbf{a}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{b} + \left[\mathbf{a}\ \mathbf{d}\ \mathbf{b}\right]\mathbf{c} + \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right]\mathbf{d} \tag{15}$$

Rearranging the scalar triple products on the right:

$$\mathbf{a}\left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right] = \left[\mathbf{a}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{b} - \left[\mathbf{a}\ \mathbf{b}\ \mathbf{d}\right]\mathbf{c} + \left[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\right]\mathbf{d} \tag{16}$$

Theoretical Solution

Substitute this into the sum:

$$= \mathbf{a}\left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right] - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} \tag{17}$$

$$= \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} - \left[\mathbf{b}\ \mathbf{c}\ \mathbf{d}\right]\mathbf{a} \tag{18}$$

$$= \mathbf{0} \tag{19}$$

The resultant vector is $\mathbf{0}$. A zero vector is considered parallel to any vector. Thus, the given vector expression is parallel to $\mathbf{a}$.

C Code

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

// Function to calculate the cross product of two 3D vectors.
void cross_product(double ax, double ay, double az,
                   double bx, double by, double bz,
                   double *rx, double *ry, double *rz) {
    *rx = ay * bz - az * by;
    *ry = az * bx - ax * bz;
    *rz = ax * by - ay * bx;
}
```

C Code

```c
// Function to calculate the dot product of two 3D vectors.
double dot_product(double ax, double ay, double az,
                   double bx, double by, double bz) {
    return ax * bx + ay * by + az * bz;
}
// Helper function to calculate a term of the form (V1 x V2) x (
    V3 x V4).
void calculate_vector_quad_cross(
    double v1x, double v1y, double v1z,
    double v2x, double v2y, double v2z,
    double v3x, double v3y, double v3z,
    double v4x, double v4y, double v4z,
    double *term_rx, double *term_ry, double *term_rz) {
```

C Code

```c
// Helper function to calculate a term of the form (V1 x V2) x (
    V3 x V4).
void calculate_vector_quad_cross(
    double v1x, double v1y, double v1z,
    double v2x, double v2y, double v2z,
    double v3x, double v3y, double v3z,
    double v4x, double v4y, double v4z,
    double *term_rx, double *term_ry, double *term_rz) {

    // Calculate V1 x V2
    double v1xv2_x, v1xv2_y, v1xv2_z;
    cross_product(v1x, v1y, v1z, v2x, v2y, v2z, &v1xv2_x, &
        v1xv2_y, &v1xv2_z);
```

C Code

```c
// Calculate V3 x V4
double v3xv4_x, v3xv4_y, v3xv4_z;

cross_product(v3x, v3y, v3z, v4x, v4y, v4z, &v3xv4_x, &
    v3xv4_y, &v3xv4_z);

// Calculate (V1 x V2) x (V3 x V4)
cross_product(v1xv2_x, v1xv2_y, v1xv2_z, v3xv4_x, v3xv4_y,
    v3xv4_z,
term_rx, term_ry, term_rz);
```

C Code

```
// Main function to calculate the full expression:
// (a  b)  (c  d) + (a  c)  (d  b) + (a  d)  (b  c)
void calculate_full_expression(
    double ax, double ay, double az,
    double bx, double by, double bz,
    double cx, double cy, double cz,
    double dx, double dy, double dz,
    double *result_x, double *result_y, double *result_z) {

    double term1_x, term1_y, term1_z;
    double term2_x, term2_y, term2_z;
    double term3_x, term3_y, term3_z;
```

C Code

```
    // Term 1: (a x b) x (c x d)
    calculate_vector_quad_cross(ax, ay, az, bx, by, bz,
                                cx, cy, cz, dx, dy, dz,
                                &term1_x, &term1_y, &term1_z);

    // Term 2: (a x c) x (d x b)
    calculate_vector_quad_cross(ax, ay, az, cx, cy, cz,
                                dx, dy, dz, bx, by, bz,
                                &term2_x, &term2_y, &term2_z);
```

C Code

```
    // Term 3: (a x d) x (b x c)
    calculate_vector_quad_cross(ax, ay, az, dx, dy, dz,
                                bx, by, bz, cx, cy, cz,
                                &term3_x, &term3_y, &term3_z);

    // Sum the terms
    *result_x = term1_x + term2_x + term3_x;
    *result_y = term1_y + term2_y + term3_y;
    *result_z = term1_z + term2_z + term3_z;
}
```

C Code

```
// Function to check if three vectors are coplanar (scalar triple
     product is zero).
double scalar_triple_product(double bx, double by, double bz,
                             double cx, double cy, double cz,
                             double dx, double dy, double dz) {
    double cxd_x, cxd_y, cxd_z;
    cross_product(cx, cy, cz, dx, dy, dz, &cxd_x, &cxd_y, &cxd_z)
        ;
    return dot_product(bx, by, bz, cxd_x, cxd_y, cxd_z);
}
```

Python Code (using C shared library)

```python
import ctypes
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load the shared library
lib_vector = ctypes.CDLL(./code5.so)

# Define argument types and return types for C functions
lib_vector.cross_product.argtypes = [
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.POINTER(ctypes.c_double), ctypes.POINTER(ctypes.c_double), ctypes.POINTER(ctypes.c_double)
]
```

Python Code (using C shared library)

```
lib_vector.cross_product.restype = None

lib_vector.dot_product.argtypes = [
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double
]
lib_vector.dot_product.restype = ctypes.c_double

lib_vector.calculate_vector_quad_cross.argtypes = [
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.POINTER(ctypes.c_double), ctypes.POINTER(ctypes.
        c_double), ctypes.POINTER(ctypes.c_double)
]
```

Python Code (using C shared library)

```
lib_vector.calculate_vector_quad_cross.restype = None

lib_vector.calculate_full_expression.argtypes = [
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.POINTER(ctypes.c_double), ctypes.POINTER(ctypes.
        c_double), ctypes.POINTER(ctypes.c_double)
]
lib_vector.calculate_full_expression.restype = None
```

Python Code (using C shared library)

```
lib_vector.scalar_triple_product.argtypes = [
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double
]
lib_vector.scalar_triple_product.restype = ctypes.c_double

# Define arbitrary non-coplanar vectors b, c, d
b_np = np.array([1.0, 0.0, 0.0])
c_np = np.array([0.0, 1.0, 0.0])
d_np = np.array([0.0, 0.0, 1.0])

# Test coplanarity
stp_val = lib_vector.scalar_triple_product(
    b_np[0], b_np[1], b_np[2], c_np[0], c_np[1], c_np[2], d_np
        [0], d_np[1], d_np[2]
)
```

Python Code (using C shared library)

```
print(fScalar Triple Product of b, c, d: {stp_val})
if np.isclose(stp_val, 0.0):
    print(Warning: Vectors b, c, d might be coplanar. Please
        choose different vectors.)
else:
    print(Vectors b, c, d are not coplanar (as required).)

# Define an arbitrary vector a
a_np = np.array([2.0, 3.0, 4.0])

# Ctypes doubles to hold the result
result_x = ctypes.c_double()
result_y = ctypes.c_double()
result_z = ctypes.c_double()
```

Python Code (using C shared library)

```
# Call the C function to calculate the full expression
lib_vector.calculate_full_expression(
    a_np[0], a_np[1], a_np[2], b_np[0], b_np[1], b_np[2],
    c_np[0], c_np[1], c_np[2], d_np[0], d_np[1], d_np[2],
    ctypes.byref(result_x), ctypes.byref(result_y), ctypes.byref(
        result_z)
)

calculated_vector_np = np.array([result_x.value, result_y.value,
    result_z.value])

print(f\nVector a: {a_np})
print(fCalculated expression vector: {calculated_vector_np})

is_parallel = False
```

Python Code (using C shared library)

```
if np.allclose(calculated_vector_np, np.array([0,0,0])):
    is_parallel = True
elif np.allclose(a_np, np.array([0,0,0])):
    is_parallel = True
else:
    cross_prod_check = np.cross(a_np, calculated_vector_np)
    if np.allclose(cross_prod_check, np.array([0, 0, 0])):
        is_parallel = True
        if np.linalg.norm(a_np) > 1e-9:
            k = np.dot(calculated_vector_np, a_np) / np.dot(a_np, a_np)
            print(fScalar multiple k: {k:.4f})
            print(fCheck: k * a = {k * a_np})
```

Python Code (using C shared library)

```
if is_parallel:
    print(\nResult: The calculated vector is parallel to vector '
        a'.)
else:
    print(\nResult: The calculated vector is NOT parallel to
        vector 'a'.)
# --- Plotting the vectors in 3D ---
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
origin = [0, 0, 0]

ax.quiver(*origin, a_np[0], a_np[1], a_np[2], color='blue',
    arrow_length_ratio=0.1, label='Vector a')
ax.text(a_np[0], a_np[1], a_np[2], 'a', color='blue')
```

Python Code (using C shared library)

```
ax.quiver(*origin, b_np[0], b_np[1], b_np[2], color='red',
    arrow_length_ratio=0.2, label='Vector b')
ax.text(b_np[0], b_np[1], b_np[2], 'b', color='red')
ax.quiver(*origin, c_np[0], c_np[1], c_np[2], color='green',
    arrow_length_ratio=0.2, label='Vector c')
ax.text(c_np[0], c_np[1], c_np[2], 'c', color='green')

ax.quiver(*origin, d_np[0], d_np[1], d_np[2], color='purple',
    arrow_length_ratio=0.2, label='Vector d')
ax.text(d_np[0], d_np[1], d_np[2], 'd', color='purple')
scale_factor = 0.5
calculated_vector_scaled = calculated_vector_np * scale_factor
```

Python Code (using C shared library)

```
ax.quiver(*origin,
          calculated_vector_scaled[0], calculated_vector_scaled[1], calculated_vector_scaled[2],
          color='black', arrow_length_ratio=0.05, label='Resultant Vector (scaled)')
ax.set_xlim([-5, 5])
ax.set_ylim([-5, 5])
ax.set_zlim([-5, 5])
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.legend()
plt.grid(True)
plt.savefig(fig1.png)
plt.show()
```

Python Code: Direct

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def vector_triple_product_expansion(A, B, C):

    Calculates the vector triple product A x (B x C) using the identity:
    A x (B x C) = (A . C) B - (A . B) C

    return np.dot(A, C) * B - np.dot(A, B) * C
```

Python Code: Direct

```python
def prove_parallel_to_a(a, b, c, d):

    Proves that (a x b) x (c x d) + (a x c) x (d x b) + (a x d) x (b x c)
    is parallel to a, given that b, c, d are not coplanar.

    # Term 1: (a x b) x (c x d)
    term1 = np.dot(np.cross(a, b), d) * c - np.dot(np.cross(a, b), c) * d
    # Term 2: (a x c) x (d x b)
    term2 = np.dot(np.cross(a, c), b) * d - np.dot(np.cross(a, c), d) * b

    # Term 3: (a x d) x (b x c)
    term3 = np.dot(np.cross(a, d), c) * b - np.dot(np.cross(a, d), b) * c
```

Python Code: Direct

```
    # Sum of the three terms
    result_vector = term1 + term2 + term3

    return result_vector

# --- Example Usage and Visualization ---
# Define non-coplanar vectors b, c, d
b = np.array([1, 0, 0])
c = np.array([0, 1, 0])
d = np.array([0, 0, 1])

# Define vector a
a = np.array([2, 3, 4])
# Prove the identity
result_vector = prove_parallel_to_a(a, b, c, d)
```

Python Code: Direct

```
# Check if result_vector is parallel to a
cross_product_check = np.cross(result_vector, a)

print(fVector a: {a})
print(fResulting vector: {result_vector})
print(fCross product of result_vector and a (should be close to
    zero for parallel): {cross_product_check})

# Calculate a scalar multiple to show parallelism
scalar_multiple = np.dot(b, np.cross(c, d))
expected_result = scalar_multiple * a

print(fExpected result based on identity [b c d] * a: {
    expected_result})
```

Python Code: Direct

```
# Numerical check for parallelism
is_parallel = np.allclose(np.cross(result_vector, a), [0, 0, 0])
print(fIs the resulting vector parallel to a? {is_parallel})

# --- 3D Plotting ---
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot vectors a, b, c, d
origin = [0, 0, 0]
ax.quiver(*origin, *a, color='red', label='a', arrow_length_ratio
    =0.1)
ax.quiver(*origin, *b, color='blue', label='b',
    arrow_length_ratio=0.2)
ax.quiver(*origin, *c, color='green', label='c',
    arrow_length_ratio=0.2)
```

Python Code: Direct

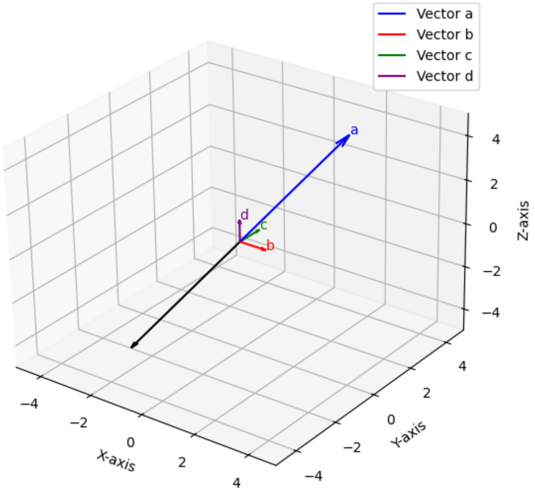
```
ax.quiver(*origin, *d, color='purple', label='d',
    arrow_length_ratio=0.2)
# Plot the resulting vector
ax.quiver(*origin, *result_vector, color='black', label='Result
    Vector', arrow_length_ratio=0.1)

# Set plot limits
max_val = np.max(np.abs(np.array([a, b, c, d, result_vector])))
ax.set_xlim([-max_val, max_val])
ax.set_ylim([-max_val, max_val])
ax.set_zlim([-max_val, max_val])
```

Python Code: Direct

```
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title(Vector Identity Proof: Result Vector Parallel to 'a'
    )
ax.legend()
plt.tight_layout()
plt.savefig(fig2.png)
plt.show()
```

Plot by Python using shared output from C



Plot by Python only

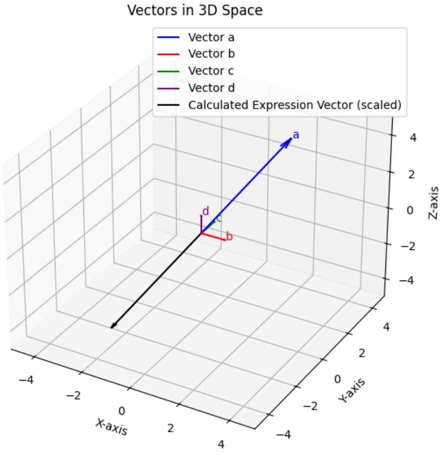


Figure: