2.3.15

BEERAM MADHURI - EE25BTECH11012

September 2025

Question

The scalar product of the vector $\hat{i} + \hat{j} + \hat{k}$ with the unit vector along the sum of vectors $2\hat{i} + 4\hat{j} - 5\hat{k}$ and $\lambda\hat{i} + 2\hat{j} + 3\hat{k}$ is equal to one. Find the value of λ.

given data

let **A**, **B** and **C** be the vectors, such that:

Variable	value
A	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$
B	$\begin{pmatrix} 2 \\ 4 \\ -5 \end{pmatrix}$
C	$\begin{pmatrix} \lambda \\ 2 \\ 3 \end{pmatrix}$

Table: Variables used

finding Scalar product of **A** with unit vector along B+ C:

given,

$$\frac{\mathbf{A}^\top(\mathbf{B}+\mathbf{C})}{\|\mathbf{B}+\mathbf{C}\|} = 1 \tag{1}$$

$$\mathbf{A}^\top(\mathbf{B}+\mathbf{C}) = \|\mathbf{B}+\mathbf{C}\| \tag{2}$$

squaring on both sides:

$$(\mathbf{A}^\top(\mathbf{B}+\mathbf{C}))^2 = \|\mathbf{B}+\mathbf{C}\|^2 \tag{3}$$

$$(\mathbf{A}^\top\mathbf{B}+\mathbf{A}^\top\mathbf{C})^2 = (\mathbf{B}+\mathbf{C})^\top(\mathbf{B}+\mathbf{C}) \tag{4}$$

Substituting the values of **A**,**B** and **C**:

$$\left(\begin{pmatrix}1 & 1 & 1\end{pmatrix}\begin{pmatrix}2\\4\\-5\end{pmatrix}+\begin{pmatrix}1 & 1 & 1\end{pmatrix}\begin{pmatrix}\lambda\\2\\3\end{pmatrix}\right)^2 = \begin{pmatrix}2+\lambda & 6 & -2\end{pmatrix}\begin{pmatrix}2+\lambda\\6\\-2\end{pmatrix} \tag{5}$$

$$(\lambda+6)^2 = \lambda^2+4\lambda+44 \tag{6}$$

$$\lambda^2+36+12\lambda = \lambda^2+4\lambda+44 \tag{7}$$

$$8\lambda = 8 \tag{8}$$

$$\lambda = 1 \tag{9}$$

Hence value of λ is 1.

Python Code

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

# --- 1. Define vectors with the calculated lambda = 1 ---
lambda_val = 1
A = np.array([1, 1, 1])
B = np.array([2, 4, -5])
C = np.array([lambda_val, 2, 3])
```

Python Code

```
# Calculate the resultant vectors ---
# Sum vector S = B + C
S = B + C
# Unit vector s_hat along S
s_hat = S / np.linalg.norm(S)
```

Python Code

```
# Set up the 3D plot ---
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
origin = [0, 0, 0]

# Plot all vectors from the origin ---
ax.quiver(*origin, *A, color='red', label=r'$\vec{A}$')
ax.quiver(*origin, *B, color='blue', label=r'$\vec{B}$')
ax.quiver(*origin, *C, color='green', label=r'$\vec{C}$ (with $\
    lambda=1$)')
ax.quiver(*origin, *S, color='purple', label=r'Sum Vector $\vec{S
    } = \vec{B} + \vec{C}$')
ax.quiver(*origin, *s_hat, color='orange', label=r'Unit Vector $\
    hat{s}$')
```

Python Code

```
# Add text labels near the vector tips for clarity ---
ax.text(A[0], A[1], A[2], 'A', color='red', fontsize=12)
ax.text(B[0], B[1], B[2], 'B', color='blue', fontsize=12)
ax.text(C[0], C[1], C[2], 'C', color='green', fontsize=12)
ax.text(S[0], S[1], S[2], 'S', color='purple', fontsize=12)
ax.text(s_hat[0]*1.5, s_hat[1]*1.5, s_hat[2]*1.5, '', color='
    orange', fontsize=14)
```

Python Code

```
# Customize and display the plot ---
ax.set_title('Visualization of Vectors', fontsize=16)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_xlim([-6, 6])
ax.set_ylim([-6, 6])
ax.set_zlim([-6, 6])
ax.legend()
ax.grid(True)
plt.show()
```

C Code

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

/*f(lambda) = (lambda + 6)^2 - (lambda + 2)^2 - 40
  We need f(lambda) = 0 */
double f(double lambda) {
    return (lambda + 6.0)*(lambda + 6.0)
         - (lambda + 2.0)*(lambda + 2.0)
         - 40.0;}
int main(void) {
    double left = -10.0; // lower bound for search
    double right = 10.0; // upper bound for search
    double mid;
    double tol = 1e-8; // desired accuracy
```

C Code

```c
// Bisection method
while ((right - left) > tol) {
    mid = (left + right) / 2.0;
    if (f(mid) == 0.0) break;

    // Root lies where sign changes
    if (f(left) * f(mid) < 0.0)
        right = mid;
    else
        left = mid;
}
```

C Code

```
    double lambda = (left + right) / 2.0;
    printf("Computed value of lambda: %.10f\n", lambda);
// Optional verification: compute the scalar product
    double sx = lambda + 2.0;
    double sy = 6.0;
    double sz = -2.0;
```

C Code

```
    double magnitude = sqrt(sx*sx + sy*sy + sz*sz);
    double scalar_product = (1.0*sx + 1.0*sy + 1.0*sz) /
        magnitude;
    printf("Verification (scalar product) : %.10f\n",
        scalar_product);
   return 0;
}
```

Python and C Code

```python
# File: main.py
import ctypes
import platform
# --- 1. Load the shared library ---
if platform.system() == "Windows":
    lib_path = "./liblambda.dll"
else:
    lib_path = "./liblambda.so"
try:
    lib = ctypes.CDLL(lib_path)
except OSError as e:
    print(f"Error loading library: {e}")
    print("Have you compiled lambda_solver.c?")
    exit()
```

Python and C Code

```
# --- 2. Define function signatures ---
# Signature for the first function: solve_for_lambda()
lib.solve_for_lambda.argtypes = [ctypes.c_double, ctypes.c_double
    , ctypes.c_double]
lib.solve_for_lambda.restype = ctypes.c_double

# Signature for the second function: verify_scalar_product()
lib.verify_scalar_product.argtypes = [ctypes.c_double]
lib.verify_scalar_product.restype = ctypes.c_double
```

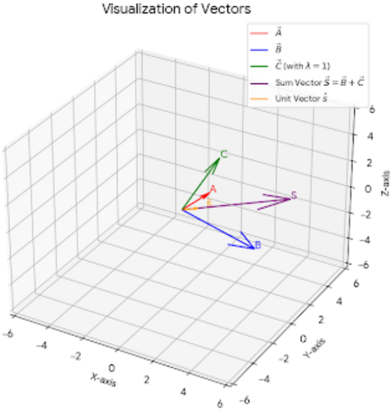
Python and C Code

```python
# --- 3. Prepare input data ---
left_bound = -10.0
right_bound = 10.0
tolerance = 1e-8
# --- 4. Call the C functions ---
# Call the first C function to solve for lambda
computed_lambda = lib.solve_for_lambda(left_bound, right_bound,
    tolerance)
```

Python and C Code

```
# Use the result to call the second C function for verification
scalar_product = lib.verify_scalar_product(computed_lambda)

# --- 5. Print the results ---
print("--- Results from C library ---")
print(f"Computed value of lambda: {computed_lambda:.10f}")
print(f"Verification (scalar product) : {scalar_product:.10f}")
```



Visualization of Vectors
$\vec{A}$
$\vec{B}$
$\vec{C}$ (with $\lambda = 1$)
Sum Vector $\vec{S} = \vec{B} + \vec{C}$
Unit Vector $\hat{s}$
X-axis
Y-axis
Z-axis

Figure: Plot