12.234

BEERAM MADHURI - EE25BTECH11012

October 2025

Question-12.234

Consider the set of vectors in three-dimensional real vector space $\mathbb{R}^3$,

$$S = \{(1,1,1),(1,-1,1),(1,1,-1)\}.$$

Which one of the following statements is true?

a) S is not a linearly independent set.

b) S is a basis for $\mathbb{R}^3$.

c) The vectors in S are orthogonal.

d) An orthogonal set of vectors cannot be generated from S.

given data

let the vectors in S be:

Point	Vector
$\mathbf{v_1}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$
$\mathbf{v_2}$	$\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$
$\mathbf{v_3}$	$\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$

Table: Variables used

finding the properties of S:

Let A be the matrix with its columns as vectors of S

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \tag{1}$$

these vectors are linearly independent if and only if

$$\det(A) \neq 0 \tag{2}$$

$$det(A) = 1(0) - 1(-2) + 1(2) \tag{3}$$
$$= 4 \neq 0 \tag{4}$$

$\therefore$ Vectors are linearly independent
$\therefore$ Since there are 3 linearly independent vectors in $\mathbb{R}^3$
they form a basis for $\mathbb{R}^3$
Let the vector be $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$.

$$\mathbf{v_1^T v_2} \neq 0 \tag{5}$$
$$\mathbf{v_1^T v_3} \neq 0 \tag{6}$$
$$\mathbf{v_2^T v_3} \neq 0 \tag{7}$$

$\therefore$ These vectors are not orthogonal
Applying Gram-Schmidt process :
let the orthogonal vectors be $\mathbf{u_1}, \mathbf{u_2}, \mathbf{u_3}$ generated from $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$

$$\mathbf{u_1} = \mathbf{v_1} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{8}$$

$$\mathbf{u_2} = \mathbf{v_2} - (\mathbf{u_1^T v_2})\hat{\mathbf{u}}_1 \tag{9}$$

$$\mathbf{u_2} = \mathbf{v_2} - \left(\frac{\mathbf{v_2}^T \mathbf{u_1}}{\mathbf{u_1}^T \mathbf{u_1}}\right) \mathbf{u_1} \tag{10}$$

$$= \begin{pmatrix} 2/3 \\ -4/3 \\ 2/3 \end{pmatrix} \tag{11}$$

$$\mathbf{u_3} = \mathbf{v_3} - (\hat{\mathbf{u}}_2^T \mathbf{v_3})\hat{\mathbf{u}}_2 \tag{12}$$

$$= \mathbf{v_3} - \left(\frac{\mathbf{u_2}^T \mathbf{v_3}}{\mathbf{u_2}^T \mathbf{u_2}}\right)\mathbf{u_2} \tag{13}$$

$$= \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} \tag{14}$$

$$\mathbf{u_1}^T \mathbf{u_2} = 0 \tag{15}$$

$$\mathbf{u_2}^T \mathbf{u_3} = 0 \tag{16}$$

$$\mathbf{u_1}^T \mathbf{u_3} = 0 \tag{17}$$

$\therefore$ an orthogonal set of vectors can be generated from S.

$\therefore$ Options b and d are correct.

Python Code

```python
import numpy as np

# Define the vectors
v1 = np.array([1, 1, 1])
v2 = np.array([1, -1, 1])
v3 = np.array([1, 1, -1])

# Form the matrix with given vectors as columns
A = np.column_stack((v1, v2, v3))
```

Python Code

```python
# 1. Check Linear Independence using determinant
det_A = np.linalg.det(A)
if abs(det_A) > 1e-9:
    print("S is a linearly independent set.")
else:
    print("S is not a linearly independent set.")
```

Python Code

```python
# 2. Check if S is a basis for R
if A.shape == (3,3) and abs(det_A) > 1e-9:
    print("S is a basis for R.")
else:
    print("S is not a basis for R.")

# 3. Check if vectors are orthogonal
def is_orthogonal(u, v):
    return np.dot(u, v) == 0
```

Python Code

```
print("Dot products:")
print("v1v2 =", np.dot(v1, v2))
print("v1v3 =", np.dot(v1, v3))
print("v2v3 =", np.dot(v2, v3))
if is_orthogonal(v1, v2) and is_orthogonal(v1, v3) and
     is_orthogonal(v2, v3):
    print("The vectors in S are orthogonal.")
else:
    print("The vectors in S are not orthogonal.")
```

Python Code

```python
# 4. Generate an orthogonal set using Gram-Schmidt process
def gram_schmidt(vectors):
    ortho = []
    for v in vectors:
        for u in ortho:
            v = v - np.dot(v, u) / np.dot(u, u) * u
        ortho.append(v)
    return ortho
orthogonal_set = gram_schmidt([v1, v2, v3])
print("\nOrthogonal set generated using Gram-Schmidt:")
for vec in orthogonal_set:
    print(vec)
```

C Code

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

// Function to calculate determinant of 3x3 matrix
float determinant(float a[3][3]) {
    float det;
    det = a[0][0]*(a[1][1]*a[2][2] - a[1][2]*a[2][1])
        - a[0][1]*(a[1][0]*a[2][2] - a[1][2]*a[2][0])
        + a[0][2]*(a[1][0]*a[2][1] - a[1][1]*a[2][0]);
    return det;
}
```

C Code

```c
// Function to compute dot product of two 3D vectors
float dot_product(float a[3], float b[3]) {
    return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
}
int main() {
    float v1[3] = {1, 1, 1};
    float v2[3] = {1, -1, 1};
    float v3[3] = {1, 1, -1};
```

C Code

```c
    // Form matrix with vectors as columns
    float A[3][3] = {
        {v1[0], v2[0], v3[0]},
        {v1[1], v2[1], v3[1]},
        {v1[2], v2[2], v3[2]}
    };
    // 1. Check linear independence using determinant
    float det = determinant(A);
    printf("Determinant = %.2f\n", det);
```

C Code

```c
    if (fabs(det) > 1e-6)
        printf("S is a linearly independent set.\n");
    else
        printf("S is NOT a linearly independent set.\n");
    // 2. Check if it is a basis for R^3
    if (fabs(det) > 1e-6)
        printf("S is a basis for R^3.\n");
    else
        printf("S is NOT a basis for R^3.\n");
```

C Code

```c
    // 3. Check orthogonality
    float d12 = dot_product(v1, v2);
    float d13 = dot_product(v1, v3);
    float d23 = dot_product(v2, v3);

    printf("\nDot products:\n");
    printf("v1v2 = %.2f\n", d12);
    printf("v1v3 = %.2f\n", d13);
    printf("v2v3 = %.2f\n", d23);
```

C Code

```c
if (fabs(d12) < 1e-6 && fabs(d13) < 1e-6 && fabs(d23) < 1e-6)
    printf("Vectors are orthogonal.\n");
else
    printf("Vectors are NOT orthogonal.\n");
// 4. Based on results, print the correct option
printf("\nCorrect statement:\n");
if (fabs(det) > 1e-6 && !(fabs(d12) < 1e-6 && fabs(d13) < 1e
    -6 && fabs(d23) < 1e-6))
    printf("Option (b): S is a basis for R^3.\n");
```

C Code

```c
    else if (fabs(det) < 1e-6)
        printf("Option (a): S is not a linearly independent set.\
            n");
    else if (fabs(d12) < 1e-6 && fabs(d13) < 1e-6 && fabs(d23) <
        1e-6)
        printf("Option (c): The vectors in S are orthogonal.\n");
    else
        printf("Option (d): An orthogonal set cannot be generated
             from S.\n");
    return 0;}
```

Python and C Code

```
import ctypes
import math
# Define C-like types
c_float = ctypes.c_float
c_float_p = ctypes.POINTER(c_float)
# Define C-like array types
# float[3] is analogous to C float v[3]
FloatArray3 = c_float * 3
# float[3][3] is analogous to C float A[3][3]
FloatMatrix3x3 = (c_float * 3) * 3
```

Python and C Code

```python
# Function to calculate determinant of 3x3 matrix
# Takes a 3x3 array/matrix of c_float
def determinant(a):
    """Calculates the determinant of a 3x3 matrix represented by
        a ctypes array."""
    # Access elements using a[row][col]
    det = a[0][0] * (a[1][1] * a[2][2] - a[1][2] * a[2][1]) \
        - a[0][1] * (a[1][0] * a[2][2] - a[1][2] * a[2][0]) \
        + a[0][2] * (a[1][0] * a[2][1] - a[1][1] * a[2][0])
    return det.value if isinstance(det, c_float) else det
```

Python and C Code

```python
# Function to compute dot product of two 3D vectors
# Takes two 3-element arrays/vectors of c_float
def dot_product(a, b):
    """Computes the dot product of two 3D vectors represented by
        ctypes arrays."""
    # Access elements using a[index]
    result = a[0] * b[0] + a[1] * b[1] + a[2] * b[2]
    return result.value if isinstance(result, c_float) else
        result
```

Python and C Code

```python
# Main execution block
def main():
    # Define vectors using the C-like FloatArray3 type
    v1 = FloatArray3(1.0, 1.0, 1.0)
    v2 = FloatArray3(1.0, -1.0, 1.0)
    v3 = FloatArray3(1.0, 1.0, -1.0)
    # Define the tolerance (epsilon) used for floating-point
        comparisons
    TOLERANCE = 1e-6
```

Python and C Code

```python
# Form matrix A with vectors as columns using the C-like
    FloatMatrix3x3 type
A = FloatMatrix3x3(
    FloatArray3(v1[0], v2[0], v3[0]), # Row 0: x-components
    FloatArray3(v1[1], v2[1], v3[1]), # Row 1: y-components
    FloatArray3(v1[2], v2[2], v3[2]) # Row 2: z-components
)

print("--- Linear Algebra Calculations with ctypes ---")
```

Python and C Code

```python
# 1. Check linear independence using determinant
det = determinant(A)
print(f"Determinant = {det:.2f}")

if abs(det) > TOLERANCE:
    print("S is a linearly independent set.")
else:
    print("S is NOT a linearly independent set.")
```

Python and C Code

```
# 2. Check if it is a basis for R^3
if abs(det) > TOLERANCE:
    print("S is a basis for R^3.")
else:
    print("S is NOT a basis for R^3.")
# 3. Check orthogonality
d12 = dot_product(v1, v2)
d13 = dot_product(v1, v3)
d23 = dot_product(v2, v3)
```

Python and C Code

```python
print("\nDot products:")
print(f"v1v2 = {d12:.2f}")
print(f"v1v3 = {d13:.2f}")
print(f"v2v3 = {d23:.2f}")
is_orthogonal = abs(d12) < TOLERANCE and abs(d13) < TOLERANCE
     and abs(d23) < TOLERANCE
if is_orthogonal:
    print("Vectors are orthogonal.")
else:
    print("Vectors are NOT orthogonal.")
```

Python and C Code

```
    # 4. Based on results, print the correct option
    print("\nCorrect statement:")
    if abs(det) > TOLERANCE and not is_orthogonal:
        print("Option (b): S is a basis for R^3.")
    elif abs(det) < TOLERANCE:
        print("Option (a): S is not a linearly independent set.")
```

Python and C Code

```
    elif is_orthogonal:
        print("Option (c): The vectors in S are orthogonal.")
    else:
        # This else block should theoretically not be reached if
            the previous logic is exhaustive
        print("Option (d): An orthogonal set cannot be generated
            from S.")

if __name__ == "__main__":
    main()
```