12.130

BEERAM MADHURI - EE25BTECH11012

October 2025

Question

The linear operation $\mathrm{L}(\mathbf{x})$ is defined by the cross product $\mathrm{L}(\mathbf{x}) = \mathbf{b} \times \mathbf{x}$, where $\mathbf{b} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ and $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ are three dimensional vectors. The 3×3 matrix $\mathbf{M}$ of this operation satisfies $\mathrm{L}(\mathbf{x}) = \mathbf{M}\mathbf{x}$. Then the eigenvalues of $\mathbf{M}$ are

- $0, +1, -1$
- $1, -1, 1$
- $i, -i, 1$
- $i, -i, 0$

Given Data

Point	Vector
$\mathbf{b}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
$\mathbf{x}$	$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$

Table: Variables used

finding the Normal:

Given,

$$L(x) = MX \tag{1}$$

$$L(x) = b \times X \tag{2}$$

Cross product can be written as skew symmetric matrix.

$$b \times X = \begin{pmatrix} 0 & -b_3 & b_2 \\ b_3 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{pmatrix} X \tag{3}$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} X \tag{4}$$

$$\therefore M = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \tag{5}$$

$$\tag{6}$$

finding eigenvalues :-

$$|M - \lambda I| = 0 \tag{7}$$

$$\begin{pmatrix} -\lambda & 0 & 0 \\ 0 & -\lambda & 1 \\ 0 & -1 & -\lambda \end{pmatrix} = 0 \tag{8}$$

$$-\lambda^3 - \lambda = 0 \tag{9}$$

$$-\lambda(\lambda^2 + 1) = 0 \tag{10}$$

$$\lambda_1 = 0 \tag{11}$$

$$\lambda_2 = i \tag{12}$$

$$\lambda_3 = -i \tag{13}$$

Hence Option d is correct.

Python Code

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

# Define the matrix M
M = np.array([
    [0, 0, 1],
    [0, 0, 0],
    [-1, 0, 0]
])
```

Python Code

```python
# Compute eigenvalues
eigenvalues = np.linalg.eigvals(M)
# Print eigenvalues
print("Eigenvalues of M:", eigenvalues)

# Plot eigenvalues in the complex plane
plt.figure(figsize=(6, 6))
plt.scatter(eigenvalues.real, eigenvalues.imag, color='red', s
    =100, label='Eigenvalues')
```

Python Code

```
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.grid(True)
plt.title('Eigenvalues in the Complex Plane')
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.legend()
plt.show()
```

C Code

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

int main() {
    // b = (0, 1, 0)
    // Cross product matrix M for b  x is:
    // | 0 0 1 |
    // | 0 0 0 |
    // | -1 0 0 |
```

C Code

```c
    double M[3][3] = {
        {0, 0, 1},
        {0, 0, 0},
        {-1, 0, 0}
    };

    // Characteristic equation of M:
    // |M - I| = 0 gives (^2 + 1) = 0
    //   = 0, i, -i
```

C Code

```c
    double complex eigen1 = 0.0 + 0.0 * I;
    double complex eigen2 = 0.0 + 1.0 * I;
    double complex eigen3 = 0.0 - 1.0 * I;
    printf("The eigenvalues of matrix M are:\n");
    printf("1 = %.1f + %.1fi\n", creal(eigen1), cimag(eigen1));
    printf("2 = %.1f + %.1fi\n", creal(eigen2), cimag(eigen2));
    printf("3 = %.1f + %.1fi\n", creal(eigen3), cimag(eigen3));
    return 0;
}
```

Python and C Code

```
import numpy as np

def calculate_eigenvalues():
    # b = (0, 1, 0)
    # Cross product matrix M for b  x is:
    # | 0 0 1 |
    # | 0 0 0 |
    # | -1 0 0 |
```

Python and C Code

```
# Define the 3x3 matrix M using a NumPy array
M = np.array([
    [0, 0, 1],
    [0, 0, 0],
    [-1, 0, 0]
])
# Calculate the eigenvalues and (optionally) eigenvectors
# 'w' will contain the eigenvalues (complex numbers)
w, v = np.linalg.eig(M)
# Note: The characteristic equation (^2 + 1) = 0 yields
#  = 0, i, -i. The calculated values should match these.
```

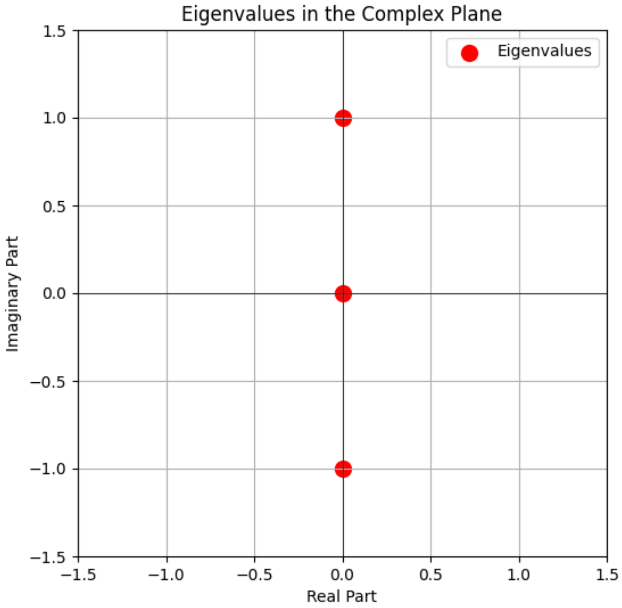
Python and C Code

```
    # Sort the eigenvalues for consistent output order (optional)
    # Sorting a mix of real and complex numbers can be tricky;
    # we'll just print them as they come out for simplicity,
    # which is usually (0+0j), (0+1j), (0-1j) or a permutation.

    print("The eigenvalues of matrix M are:")
    # Use a loop to print each eigenvalue
    for i, eigenvalue in enumerate(w):
```

Python and C Code

```
        # NumPy returns eigenvalues as complex numbers (even if the imaginary part is zero)
        # We can format the output to match the C code style (real + imag*i)
        print(f"{i+1} = {eigenvalue.real:.1f} + {eigenvalue.imag:.1f}i")
# Execute the function
if __name__ == "__main__":
    calculate_eigenvalues()
```


Eigenvalues in the Complex Plane
Eigenvalues
Imaginary Part
Real Part
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5