5.13.53

INDHIRESH S - EE25BTECH11027

29 September, 2025

Question

Given

$$\begin{aligned} 2x - y + 2z &= 2 \\ x - 2y + z &= -4 \\ x + y + \lambda z &= 4 \end{aligned}$$

then the value of λ such that the given system of equation has NO solution, is

1. 3
2. 1
3. 0
4. -3

Equation I

The given equation can be combined as:

$$\mathbf{A}\mathbf{x} = \mathbf{C} \tag{1}$$

$$\begin{pmatrix} 2 & -1 & 2 \\ 1 & -2 & 1 \\ 1 & 1 & \lambda \end{pmatrix} \mathbf{x} = \begin{pmatrix} 2 \\ -4 \\ 4 \end{pmatrix} \tag{2}$$

Where,

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 2 \\ 1 & -2 & 1 \\ 1 & 1 & \lambda \end{pmatrix} \quad and \quad \mathbf{C} = \begin{pmatrix} 2 \\ -4 \\ 4 \end{pmatrix} \tag{3}$$

Theoretical Solution

Now forming the augmented matrix:

$$[\mathbf{A}|\mathbf{C}] = \left(\begin{array}{ccc|c} 2 & -1 & 2 & 2 \\ 1 & -2 & 1 & -4 \\ 1 & 1 & \lambda & 4 \end{array}\right) \tag{4}$$

$$\left(\begin{array}{ccc|c} 2 & -1 & 2 & 2 \\ 1 & -2 & 1 & -4 \\ 1 & 1 & \lambda & 4 \end{array}\right) \xleftrightarrow[R_1 \leftarrow R_1 - 2R_2]{R_3 \leftarrow R_3 - R_2} \left(\begin{array}{ccc|c} 0 & 3 & 0 & 10 \\ 1 & -2 & 1 & -4 \\ 1 & 1 & \lambda & 4 \end{array}\right) \tag{5}$$

$$\left(\begin{array}{ccc|c} 0 & 3 & 0 & 10 \\ 1 & -2 & 1 & -4 \\ 1 & 1 & \lambda & 4 \end{array}\right) \xleftrightarrow{R_3 \leftarrow R_3 - R_2} \left(\begin{array}{ccc|c} 0 & 3 & 0 & 10 \\ 1 & -2 & 1 & -4 \\ 0 & 3 & \lambda - 1 & 8 \end{array}\right) \tag{6}$$

Theoretical solution

$$\left(\begin{array}{ccc|c} 0 & 3 & 0 & 10 \\ 1 & -2 & 1 & -4 \\ 0 & 3 & \lambda-1 & 8 \end{array}\right) \xleftrightarrow{R_3 \leftarrow R_3 - R_1} \left(\begin{array}{ccc|c} 0 & 3 & 0 & 10 \\ 1 & -2 & 1 & -4 \\ 0 & 0 & \lambda-1 & -2 \end{array}\right) \tag{7}$$

Given that the system of equation has NO solution . So,

$$\lambda = 1 \tag{8}$$

C Code

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

// Function to perform Gaussian elimination on a 3x4 augmented
    matrix [A|B]
// Returns 0 for No Solution (Inconsistent), 1 for Solvable (
    Unique/Infinite).
int solve_system_c(double matrix[3][4]) {
    const int N = 3; // Number of variables
    int i, j, k;
    double factor;
    const double TOLERANCE = 1e-9; // Tolerance for floating
        point zero checks
```

C Code

```c
    // --- Forward Elimination ---
    for (i = 0; i < N; i++) {
        // Find pivot (for robustness, a full pivot search would
            be better, but we assume simple pivots)
        if (fabs(matrix[i][i]) < TOLERANCE) {
            for (k = i + 1; k < N; k++) {
                if (fabs(matrix[k][i]) > TOLERANCE) {
                    // Swap R_i and R_k
                    for (j = i; j < N + 1; j++) {
                        double temp = matrix[i][j];
                        matrix[i][j] = matrix[k][j];
                        matrix[k][j] = temp;
                    }
                    break;
                }
            }
```

C Code

```
        if (fabs(matrix[i][i]) < TOLERANCE) continue; // Skip if
            no good pivot found
        }

        // Eliminate
        for (k = i + 1; k < N; k++) {
            factor = matrix[k][i] / matrix[i][i];
            for (j = i; j < N + 1; j++) {
                matrix[k][j] -= factor * matrix[i][j];
            }
        }
    }
```

```c
        // --- Check for Inconsistency (No Solution) ---
    // Look for a row [0 0 0 | k] where k != 0
    if (fabs(matrix[N-1][N-1]) < TOLERANCE && fabs(matrix[N-1][N
        ]) > TOLERANCE) {
        return 0; // No Solution
    }

    return 1; // Solvable (Could be unique or infinite, but not '
        no solution')
}
```

Python Code

```
import ctypes
import os
import numpy as np
from subprocess import run, PIPE
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# --- Step 1: Compile the C code ---
C_FILE = 'system.c'
os.name == 'posix'
LIB_FILE = './system.so'
compile_cmd = ['gcc', '-shared', '-o', LIB_FILE, C_FILE, '-fPIC',
     '-lm']

print(fAttempting to compile C file: {C_FILE})
compile_result = run(compile_cmd, stderr=PIPE)
```

Python Code

```
    if compile_result.returncode != 0:
     print(C Compilation Error:)
     print(compile_result.stderr.decode())
     exit()
print(fSuccessfully compiled {C_FILE} to {LIB_FILE})

# --- Step 2: Load the compiled library and define types ---
try:
    lib = ctypes.CDLL(os.path.abspath(LIB_FILE)) # Use absolute
        path for robustness
except OSError as e:
    print(fError loading {LIB_FILE}. Error: {e})
    exit()
```

Python Code

```
    # Define C types for the 3x4 array argument (double matrix
        [3][4])
c_double_ptr = ctypes.POINTER(ctypes.c_double)
lib.solve_system_c.argtypes = [c_double_ptr]
lib.solve_system_c.restype = ctypes.c_int

# --- Step 3: Prepare the data for the system with lambda = 1 ---
# Original Augmented Matrix [A|B] for lambda=1
# R1: 2x - y + 2z = 2
# R2: x - 2y + z = -4
# R3: x + y + 1z = 4
matrix_np = np.array([
    [2.0, -1.0, 2.0, 2.0],
    [1.0, -2.0, 1.0, -4.0],
    [1.0, 1.0, 1.0, 4.0]
], dtype=np.float64)
```

Python Code

```
    # Create a copy to pass to C, as C function might modify it
        in-place
matrix_for_c = matrix_np.copy()
matrix_c_ptr = matrix_for_c.ctypes.data_as(c_double_ptr)

# --- Step 4: Call the C function and interpret the result ---
print(\n--- Running C Code (Gaussian Elimination) ---)
result_code = lib.solve_system_c(matrix_c_ptr)

print(fSystem tested with lambda = 1.)
if result_code == 0:
    print(C Function Output: NO SOLUTION (Inconsistent System))
    plot_required = True
elif result_code == 1:
    print(C Function Output: Solvable (Consistent System))
    plot_required = False # Only plot if there's no solution for
        this problem
```

Python Code

```
   else:
    print(C Function Output: Unknown result code.)
    plot_required = False

print(\nFinal Matrix after C Gaussian Elimination:)
print(matrix_for_c) # Shows the modified matrix from C

# --- Step 5: Plot the graph if 'No Solution' is found ---
if plot_required:
    print(\n--- Generating 3D Plot of Planes ---)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Define the range for x and y
    x_range = np.linspace(-5, 5, 20)
    y_range = np.linspace(-5, 5, 20)
    X, Y = np.meshgrid(x_range, y_range)
```

Python Code

```
# Calculate z for each plane from original equations
# P1: 2x - y + 2z = 2 => z = (2 - 2x + y) / 2
Z1 = (2 - 2*X + Y) / 2

# P2: x - 2y + z = -4 => z = -4 - x + 2y
Z2 = -4 - X + 2*Y

# P3: x + y + z = 4 => z = 4 - x - y
Z3 = 4 - X - Y

# Plot the planes
# Note: plot_surface does not directly support 'label' for
    legend.
```

Python Code

```python
# We will approximate or add dummy plots for legend.
  ax.plot_surface(X, Y, Z1, alpha=0.5, color='cyan', rstride
      =100, cstride=100)
  ax.plot_surface(X, Y, Z2, alpha=0.5, color='red', rstride
      =100, cstride=100)
  ax.plot_surface(X, Y, Z3, alpha=0.5, color='yellow', rstride
      =100, cstride=100)

  # Create dummy plots for legend
  ax.plot([], [], [], color='cyan', label='Plane 1: $2x - y + 2
      z = 2$')
  ax.plot([], [], [], color='red', label='Plane 2: $x - 2y + z
      = -4$')
  ax.plot([], [], [], color='yellow', label='Plane 3: $x + y +
      z = 4$')
```

Python Code

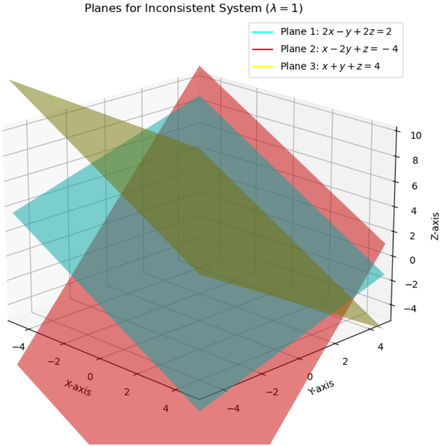
```
# Customize the plot appearance
   ax.set_xlabel('X-axis')
   ax.set_ylabel('Y-axis')
   ax.set_zlabel('Z-axis')
   ax.set_title('Planes for Inconsistent System ($\lambda=1$)')
   ax.legend()

   # Set view to better show the no intersection or triangular
       tunnel
   ax.view_init(elev=20, azim=-45)
   ax.set_xlim([-5, 5])
   ax.set_ylim([-5, 5])
   ax.set_zlim([-5, 10]) # Adjust z-limits as needed for better
       visualization
```

Python Code

```
plt.savefig(/media/indhiresh-s/New Volume/Matrix/ee1030-2025/
    ee25btech11027/MATGEO/5.13.53/figs/figure1.png)
  plt.show()
```

Plot


Planes for Inconsistent System ($\lambda = 1$)
Plane 1: $2x - y + 2z = 2$
Plane 2: $x - 2y + z = -4$
Plane 3: $x + y + z = 4$
X-axis
Y-axis
Z-axis
−4
−2
0
2
4
10
8
6
4
2
0
−2
−4