5.4.31

M Chanakya Srinivas- EE25BTECH11036

Question: Using elementary row transformations, find the inverse of

$$A = \begin{pmatrix} 1 & 2 \\ 4 & 2 \end{pmatrix}.$$

Method

The inverse of a non-singular matrix A can be found using the augmented form

$$\left(nn\middle|\mathrm{A} \quad \mathrm{I}\right) \xrightarrow{\text{row operations}} \left(nn\middle|\mathrm{I} \quad \mathrm{A}^{-1}\right).$$

This is known as the **Gauss–Jordan elimination method**.

Step-by-Step Solution

$$\left(\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 4 & 2 & 0 & 1 \end{array}\right) \quad \text{Initial augmented matrix} \tag{1}$$

$$R_2 \leftarrow R_2 - 4R_1 :$$

$$\left(\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & -6 & -4 & 1 \end{array}\right) \tag{2}$$

$$R_2 \leftarrow -\frac{1}{6}R_2 :$$

$$\left(\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 1 & \frac{2}{3} & -\frac{1}{6} \end{array}\right) \tag{3}$$

$$R_1 \leftarrow R_1 - 2R_2 :$$

$$\left(\begin{array}{cc|cc} 1 & 0 & -\frac{1}{3} & \frac{1}{3} \\ 0 & 1 & \frac{2}{3} & -\frac{1}{6} \end{array}\right) \tag{4}$$

Inverse of A

From the final augmented matrix, the left block is I, so the right block is

$$A^{-1} = \begin{pmatrix} -\frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & -\frac{1}{6} \end{pmatrix}.$$

C Code

```c
#include <stdio.h>

// Function to compute inverse of a 2x2 matrix
// Input: double A[2][2]
// Output: double Inv[2][2]
// Returns: 0 if successful, -1 if singular
int inverse2x2(const double A[2][2], double Inv[2][2]) {
    double det = A[0][0]*A[1][1] - A[0][1]*A[1][0];

    if (det == 0.0) {
        return -1; // singular matrix
    }

    double invDet = 1.0 / det;
```

C Code

```
    Inv[0][0] = A[1][1] * invDet;
    Inv[0][1] = -A[0][1] * invDet;
    Inv[1][0] = -A[1][0] * invDet;
    Inv[1][1] = A[0][0] * invDet;

    return 0;
}

// For testing purpose (can be removed when used as .so)
#ifdef TEST_MAIN
```

C Code

```
int main() {
    double A[2][2] = { {1, 2}, {4, 2} };
    double Inv[2][2];

    if (inverse2x2(A, Inv) == 0) {
        printf(Inverse matrix:\n);
        printf(%lf %lf\n, Inv[0][0], Inv[0][1]);
        printf(%lf %lf\n, Inv[1][0], Inv[1][1]);
    } else {
        printf(Matrix is singular!\n);
    }

    return 0;
}
#endif
```

Python code through shared output

```
import ctypes
import numpy as np

# Load the shared library
lib = ctypes.CDLL(./libmatrix.so)

# Define function signature: int inverse2x2(const double A[2][2],
    double Inv[2][2])
lib.inverse2x2.argtypes = [
    (ctypes.c_double * 2) * 2, # input matrix
    (ctypes.c_double * 2) * 2 # output matrix
]
lib.inverse2x2.restype = ctypes.c_int
```

Python code through shared output

```
def inverse2x2(A):

    Call the C function to compute inverse of 2x2 matrix.
    A: numpy array (2x2)
    Returns: numpy array (2x2) or None if singular

    A_c = ((ctypes.c_double * 2) * 2)()
    Inv_c = ((ctypes.c_double * 2) * 2)()

    # Fill input matrix
    for i in range(2):
        for j in range(2):
            A_c[i][j] = A[i, j]

    # Call C function
    status = lib.inverse2x2(A_c, Inv_c)
    if status != 0:
        return None # singular
```

```
    # Convert back to numpy
    Inv = np.zeros((2, 2))
    for i in range(2):
        for j in range(2):
            Inv[i, j] = Inv_c[i][j]
    return Inv

# ---- Example Usage without plotting ----
m_values = np.linspace(0.1, 5, 50)
determinants = []
inverse_norms = []

for m in m_values:
    A = np.array([[1, m], [4, 2]], dtype=float) # parametric
        family of matrices
    det = np.linalg.det(A)
    Inv = inverse2x2(A)
```

Python code through shared output

```
    determinants.append(det)
    if Inv is not None:
        inverse_norms.append(np.linalg.norm(Inv))
    else:
        inverse_norms.append(np.nan)

# Print results
for i, m in enumerate(m_values):
    print(fm = {m:.2f}, det(A) = {determinants[i]:.4f}, ||A^-1||
        = {inverse_norms[i]})
```

only Python code

```
import numpy as np

# ---- Example Usage ----
m_values = np.linspace(0.1, 5, 50)
determinants = []
inverse_norms = []

for m in m_values:
    A = np.array([[1, m], [4, 2]], dtype=float) # parametric 2x2
        matrix
    det = np.linalg.det(A)
```

only Python code

```python
    try:
        Inv = np.linalg.inv(A)
        norm_inv = np.linalg.norm(Inv)
    except np.linalg.LinAlgError:
        Inv = None
        norm_inv = np.nan

    determinants.append(det)
    inverse_norms.append(norm_inv)

# Print results
for i, m in enumerate(m_values):
    print(fm = {m:.2f}, det(A) = {determinants[i]:.4f}, ||A^-1|| = {inverse_norms[i]:.4f})
```