5.4.37

EE25BTECH11043 - Nishid Khandagre

October 1, 2025

Question

Using elementary transformations, find the inverse of the following matrix.

$$\begin{pmatrix} 1 & 1 & -2 \\ 2 & 1 & -3 \\ 5 & 4 & -9 \end{pmatrix}$$

Theoretical Solution

Let the given matrix be $\mathbf{A}$:

$$A = \begin{pmatrix} 1 & 1 & -2 \\ 2 & 1 & -3 \\ 5 & 4 & -9 \end{pmatrix} \tag{1}$$

To find $\mathbf{A}^{-1}$, we augment the matrix $\mathbf{A}$ with the identity matrix $\mathbf{I}$:

$$\left(\begin{array}{ccc|ccc} 1 & 1 & -2 & 1 & 0 & 0 \\ 2 & 1 & -3 & 0 & 1 & 0 \\ 5 & 4 & -9 & 0 & 0 & 1 \end{array}\right) \tag{2}$$

Theoretical Solution

Apply elementary row operations:

$$R_2 \to R_2 - 2R_1$$
$$R_3 \to R_3 - 5R_1$$

$$\left(\begin{array}{ccc|ccc} 1 & 1 & -2 & 1 & 0 & 0 \\ 0 & -1 & 1 & -2 & 1 & 0 \\ 0 & -1 & 1 & -5 & 0 & 1 \end{array}\right) \tag{3}$$

Theoretical Solution

Then

$$R_3 \to R_3 - R_2$$

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

Theoretical Solution

Since in the left block the last row is all zeros $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$. So the left block cannot be converted into Identity matrix.

Also the left block has rank<3, So the left block is singular.

Therefore, the inverse of the matrix does not exist.

C Code

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

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

C Code

```
// Function to calculate the cofactor of a 3x3 matrix
void calculateCofactor(double matrix[3][3], double cofactor
    [3][3]) {
    // Cofactor for element (0,0)
    cofactor[0][0] = (matrix[1][1] * matrix[2][2] - matrix[1][2]
        * matrix[2][1]);
    // Cofactor for element (0,1)
    cofactor[0][1] = -(matrix[1][0] * matrix[2][2] - matrix[1][2]
         * matrix[2][0]);
    // Cofactor for element (0,2)
    cofactor[0][2] = (matrix[1][0] * matrix[2][1] - matrix[1][1]
        * matrix[2][0]);

    // Cofactor for element (1,0)
    cofactor[1][0] = -(matrix[0][1] * matrix[2][2] - matrix[0][2]
         * matrix[2][1]);
```

C Code

```
    // Cofactor for element (1,1)
    cofactor[1][1] = (matrix[0][0] * matrix[2][2] - matrix[0][2]
        * matrix[2][0]);
    // Cofactor for element (1,2)
    cofactor[1][2] = -(matrix[0][0] * matrix[2][1] - matrix[0][1]
         * matrix[2][0]);

    // Cofactor for element (2,0)
    cofactor[2][0] = (matrix[0][1] * matrix[1][2] - matrix[0][2]
        * matrix[1][1]);
    // Cofactor for element (2,1)
    cofactor[2][1] = -(matrix[0][0] * matrix[1][2] - matrix[0][2]
         * matrix[1][0]);
    // Cofactor for element (2,2)
    cofactor[2][2] = (matrix[0][0] * matrix[1][1] - matrix[0][1]
        * matrix[1][0]);
}
```

C Code

```c
// Function to transpose a matrix
void transposeMatrix(double matrix[3][3], double transpose[3][3])
     {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
}
// Main function to find the inverse of a 3x3 matrix
int findInverse(double matrix[3][3], double inverse[3][3]) {
    double det = calculateDeterminant(matrix);
    // If determinant is zero, inverse does not exist
    if (fabs(det) < 1e-9) {
        printf(Error: Determinant is zero. Inverse does not exist
            .\n);
        return 1; // Indicate error
    }
```

C Code

```
    double cofactor_matrix[3][3];
    calculateCofactor(matrix, cofactor_matrix);

    double adjoint_matrix[3][3];
    transposeMatrix(cofactor_matrix, adjoint_matrix);

    // Calculate the inverse: A_inv = (1/det) * adj(A)
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            inverse[i][j] = adjoint_matrix[i][j] / det;
        }
    }
    return 0;
}
```

Python Code using C shared library

```
import ctypes
import numpy as np

# Load the shared library (assuming code11.so is compiled)
lib_matrix = ctypes.CDLL(./code11.so)

# Define the argument types and return type for the C function
lib_matrix.findInverse.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64, shape=(3, 3), flags=
        'C_CONTIGUOUS'),
    np.ctypeslib.ndpointer(dtype=np.float64, shape=(3, 3), flags=
        'C_CONTIGUOUS')
]
lib_matrix.findInverse.restype = ctypes.c_int
```

Python Code using C shared library

```
# The matrix from the problem:
original_matrix = np.array([
    [1.0, 1.0, -2.0],
    [2.0, 1.0, -3.0],
    [5.0, 4.0, -9.0]
], dtype=np.float64)

# Create an empty numpy array to store the result of the inverse
inverse_matrix_result = np.zeros((3, 3), dtype=np.float64)

print(Original Matrix:)
print(original_matrix)

# Call the C function to find the inverse
success_code = lib_matrix.findInverse(original_matrix,
    inverse_matrix_result)
```

Python Code using C shared library

```
if success_code == 0:
    print(\nThe inverse of the matrix is:)
    print(inverse_matrix_result)

    # Verification (A * A_inv should be close to identity matrix)
    print(\nVerification (Original * Inverse):)
    print(np.dot(original_matrix, inverse_matrix_result))
else:
    print(\nCould not find the inverse as the matrix is singular
        (determinant is zero).)
```

Direct Python Code

```python
import numpy as np

def get_submatrix(matrix, skip_row, skip_col):

    Returns a submatrix by skipping a specified row and column.

    submatrix = []
    for r_idx, row in enumerate(matrix):
        if r_idx == skip_row:
            continue
        new_row = []
        for c_idx, val in enumerate(row):
            if c_idx == skip_col:
                continue
            new_row.append(val)
        submatrix.append(new_row)
    return np.array(submatrix)
```

Direct Python Code

```
def determinant_2x2(matrix):

    Calculates the determinant of a 2x2 matrix.

    return matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1,
        0]

def determinant_3x3(matrix):

    Calculates the determinant of a 3x3 matrix.

    det = (matrix[0, 0] * (matrix[1, 1] * matrix[2, 2] - matrix
        [1, 2] * matrix[2, 1]) -
          matrix[0, 1] * (matrix[1, 0] * matrix[2, 2] - matrix[1,
              2] * matrix[2, 0]) +
          matrix[0, 2] * (matrix[1, 0] * matrix[2, 1] - matrix[1,
              1] * matrix[2, 0]))
    return det
```

Direct Python Code

```
def cofactor_matrix_3x3(matrix):

    Calculates the cofactor matrix for a 3x3 matrix.

    cofactor_mat = np.zeros((3, 3), dtype=float)
    for r in range(3):
        for c in range(3):
            sub = get_submatrix(matrix, r, c)
            minor = determinant_2x2(sub)
            cofactor_mat[r, c] = ((-1)**(r + c)) * minor
    return cofactor_mat

def inverse_matrix_3x3(matrix):

    Finds the inverse of a 3x3 matrix.
    A_inv = (1/det(A)) * adj(A)
```

Direct Python Code

```
    if matrix.shape != (3, 3):
        raise ValueError(Input matrix must be 3x3.)

    det = determinant_3x3(matrix)

    if abs(det) < 1e-9:
        print(Error: Determinant is zero or very close to zero.
            Inverse does not exist.)
        return None

    cofactor_mat = cofactor_matrix_3x3(matrix)
    adjoint_mat = cofactor_mat.T
    inverse = (1 / det) * adjoint_mat
    return inverse
```

Direct Python Code

```
# The matrix from the problem:
original_matrix = np.array([
    [1.0, 1.0, -2.0],
    [2.0, 1.0, -3.0],
    [5.0, 4.0, -9.0]
], dtype=float)

print(Original Matrix:)
print(original_matrix)

inverse_result = inverse_matrix_3x3(original_matrix)
```

Direct Python Code

```
if inverse_result is not None:
    print(\nThe inverse of the matrix is:)
    print(inverse_result)

    # Verification: Multiply original matrix by its inverse
    print(\nVerification (Original * Inverse):)
    print(np.dot(original_matrix, inverse_result))
```