5.4.22

INDHIRESH S - EE25BTECH11027

29 September, 2025

Question

Using elementary transformations, find the inverse of the following matrices

$$\begin{pmatrix} 6 & -3 \\ -2 & 1 \end{pmatrix}$$

Equation I

Let the given matrix be:

$$\mathbf{A} = \begin{pmatrix} 6 & -3 \\ -2 & 1 \end{pmatrix} \tag{1}$$

Now finding the inverse of a matrix by elementary operation.
Now forming the augmented matrix $[\mathbf{A}|\mathbf{I}]$

$$[\mathbf{A}|\mathbf{I}] = \left(\begin{array}{cc|cc} 6 & -3 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{array}\right) \tag{2}$$

$$\left(\begin{array}{cc|cc} 6 & -3 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{array}\right) \xleftrightarrow{R_2 \leftarrow R_2 + \frac{1}{3}R_1} \left(\begin{array}{cc|cc} 6 & -3 & 1 & 0 \\ 0 & 0 & \frac{1}{3} & 1 \end{array}\right) \tag{3}$$

From above we can observe that the rank of the left-side augmented matrix is 1
Therefore the matrix $\mathbf{A}$ is singular and hence the inverse does not exist for the given matrix

C Code

```c
#include <stddef.h> // For size_t

/**
 * @brief Performs the operation: dest_row = dest_row + (factor *
     src_row)
 * This is the core operation for elimination.
 * @param dest_row The row to be modified.
 * @param src_row The row used for the operation.
 * @param factor The multiplication factor.
 * @param size The number of elements in the rows.
 */
void add_scaled_row(double* dest_row, const double* src_row,
    double factor, size_t size) {
    for (size_t i = 0; i < size; i++) {
        dest_row[i] += factor * src_row[i];
    }
}
```

C Code

```c
/**
 * @brief Scales a row by multiplying each element by a factor.
 * Used for normalization (making pivots equal to 1).
 * @param row The row to be scaled.
 * @param factor The scaling factor.
 * @param size The number of elements in the row.
 */
void scale_row(double* row, double factor, size_t size) {
    for (size_t i = 0; i < size; i++) {
        row[i] *= factor;
    }
}
```

Python Code

```
import numpy as np
import ctypes

# Define a pointer type for a double array
DOUBLE_PTR = ctypes.POINTER(ctypes.c_double)

c_lib = ctypes.CDLL('./inverse.so')

# Define argtypes for add_scaled_row(double*, double*, double,
    size_t)
c_lib.add_scaled_row.argtypes = [DOUBLE_PTR, DOUBLE_PTR, ctypes.
    c_double, ctypes.c_size_t]
c_lib.add_scaled_row.restype = None

# Define argtypes for scale_row(double*, double, size_t)
c_lib.scale_row.argtypes = [DOUBLE_PTR, ctypes.c_double, ctypes.
    c_size_t]
c_lib.scale_row.restype = None
```

Python Code

```
def invert_matrix_with_c(A):

    Inverts a 2x2 matrix using Python logic and C row operations.

    if A.shape != (2, 2):
        return Input must be a 2x2 matrix., False

    # 1. Create the 2x4 augmented matrix [A|I]
    I = np.identity(2)
    aug = np.concatenate((A, I), axis=1).astype(np.float64)
    num_cols = aug.shape[1]

    # Get C-compatible pointers to the start of each row's data
    row0_ptr = aug[0].ctypes.data_as(DOUBLE_PTR)
    row1_ptr = aug[1].ctypes.data_as(DOUBLE_PTR)

    # --- Python Logic controlling C Functions ---
```

Python Code

```
# 2. Forward Elimination
    # Python calculates the factor
    factor = -aug[1, 0] / aug[0, 0]
    # C performs the operation: R2 -> R2 + factor * R1
    c_lib.add_scaled_row(row1_ptr, row0_ptr, factor, num_cols)

    # 3. Check for Singularity (in Python)
    if abs(aug[1, 1]) < 1e-9:
        return Matrix is singular; inverse does not exist., False

    # 4. Backward Elimination
    # Python calculates the factor
    factor = -aug[0, 1] / aug[1, 1]
    # C performs the operation: R1 -> R1 + factor * R2
    c_lib.add_scaled_row(row0_ptr, row1_ptr, factor, num_cols)
```

Python Code

```
# 5. Normalization
    # Python calculates the scaling factor for row 0
    scale_factor_r0 = 1.0 / aug[0, 0]
    # C scales the row: R1 -> R1 / aug[0, 0]
    c_lib.scale_row(row0_ptr, scale_factor_r0, num_cols)

    # Python calculates the scaling factor for row 1
    scale_factor_r1 = 1.0 / aug[1, 1]
    # C scales the row: R2 -> R2 / aug[1, 1]
    c_lib.scale_row(row1_ptr, scale_factor_r1, num_cols)

    # 6. Extract the inverse
    inverse = aug[:, 2:]
    return inverse, True
```

Python Code

```
# --- Main execution ---
if __name__ == __main__:
    matrix = np.zeros((2, 2))
    print(Enter the elements of the 2x2 matrix:)
    for i in range(2):
        for j in range(2):
            value = float(input(fEnter element [{i}][{j}]: ))
            matrix[i, j] = value

    print(\nInput Matrix:\n, matrix)
    print(--------------------------------)

    result, success = invert_matrix_with_c(matrix)

    if success:
        print(Inverse Matrix Found:\n, result)
    else:
        print(Result:, result)
```