12.754

BEERAM MADHURI - EE25BTECH11012

October 2025

Question

Let $\mathbf{Q} = \begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix}$ be a 2×2 matrix. Which one of the following statements is **TRUE**?

a) $\mathbf{Q}$ is equal to its transpose.
b) $\mathbf{Q}$ is equal to its inverse.
c) $\mathbf{Q}$ is full rank.
d) $\mathbf{Q}$ has linearly dependent columns.

finding the properties of Q:

given

$$\mathbf{Q} = \begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix} \tag{1}$$

a)

$$\mathbf{Q}^\top = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \tag{2}$$

$$\mathbf{Q} \neq \mathbf{Q}^\top \tag{3}$$

b)

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

$$\text{If } \mathbf{Q} = \mathbf{Q}^{-}1 \text{ then } \mathbf{Q}^2 = I \tag{5}$$

$$\mathbf{Q}^2 = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \tag{6}$$

$$= \begin{pmatrix} -3 & 4 \\ -4 & -3 \end{pmatrix} \neq \mathbf{I} \tag{7}$$

c)

$$\mathbf{Q} = \begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix} \tag{8}$$

Using Row reduction:-

$$\begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix} \xrightarrow{R_2 - 2(R_1)} \begin{pmatrix} 1 & -2 \\ 0 & -3 \end{pmatrix} \tag{9}$$

$$rank = 2 \tag{10}$$

$\therefore$ **Q** is a full rank Matrix.

d) Columns of $\mathbf{Q}$ are linearly dependent if

$$\mathbf{c_1} = \lambda \mathbf{c_2} \quad (\lambda \neq 0) \tag{11}$$

where $\mathbf{c_1}$ = first column of $\mathbf{Q}$
$\mathbf{c_2}$ = second column of $\mathbf{Q}$.

$$\mathbf{c_1} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \mathbf{c_2} = \begin{pmatrix} -2 \\ 1 \end{pmatrix} \tag{12}$$

$$\mathbf{c_1} \neq \lambda \mathbf{c_2} \text{ for any } \lambda \neq 0 \tag{13}$$

$\therefore$ columns of $\mathbf{Q}$ are linearly independent.

$\therefore$ Option C is correct.

Python Code

```
import numpy as np
from numpy.linalg import inv, det, LinAlgError

def solve_matrix_problem():
    """
    Analyzes the matrix Q and verifies the given statements.
    """
    # Define the 2x2 matrix Q as a NumPy array
    Q = np.array([[1, -2],
                  [2, 1]])

    print(f"Given Matrix Q:\n{Q}\n")
```

Python Code

```
# --- a) Check if Q is equal to its transpose ---
print("a) Checking if Q is equal to its transpose...")
# In NumPy, the .T attribute gets the transpose
Q_T = Q.T
print(f"Transpose Q^T:\n{Q_T}")
# np.array_equal safely checks if two arrays are element-wise
    equal
is_equal_to_transpose = np.array_equal(Q, Q_T)
print(f"Result: Statement (a) is {is_equal_to_transpose}.\n")
```

Python Code

```python
# --- b) Check if Q is equal to its inverse ---
print("b) Checking if Q is equal to its inverse...")
try:
    # np.linalg.inv calculates the inverse
    Q_inv = inv(Q)
    print(f"Inverse Q^-1:\n{np.round(Q_inv, 2)}") # Round for cleaner display
    is_equal_to_inverse = np.array_equal(Q, Q_inv)
    print(f"Result: Statement (b) is {is_equal_to_inverse}.\n")
except LinAlgError:
    # This block runs if the matrix has no inverse (is singular)
    print("Inverse does not exist.")
    print("Result: Statement (b) is False.\n")
```

Python Code

```python
# --- c) & d) Check rank and column dependency via the determinant ---
print("c/d) Checking for full rank and column dependency...")
# np.linalg.det calculates the determinant
q_det = det(Q)
print(f"Determinant of Q = {q_det:.2f}")

# A square matrix has full rank if its determinant is non-zero.
# Its columns are linearly dependent if the determinant is zero.
if abs(q_det) > 1e-9: # Use tolerance for floating point comparison
    print("Result: Statement (c) is True (Determinant is non-zero, so Q is of full rank).")
    print("Result: Statement (d) is False (Columns are linearly independent).\n")
```

Python Code

```
else:
        print("Result: Statement (c) is False (Determinant is zero, so Q is not of full rank).")
        print("Result: Statement (d) is True (Columns are linearly dependent).\n")
    print("Conclusion: The only TRUE statement is (c).")

if _name_ == "_main_":
    solve_matrix_problem()
```

C Code

```c
#include <stdio.h>
#include <math.h> // Required for fabs() for floating-point
    comparisons

// --- Function Prototypes ---
void printMatrix(const char* name, double matrix[2][2]);
double determinant(double matrix[2][2]);
void transpose(double in[2][2], double out[2][2]);
int inverse(double in[2][2], double out[2][2]); // Returns 1 on
    success, 0 on failure
int areMatricesEqual(double A[2][2], double B[2][2]);
```

C Code

```
int main() {
    // Define the 2x2 matrix Q
    double Q[2][2] = {{1.0, -2.0}, {2.0, 1.0}};

    printf("Given Matrix Q:\n");
    printMatrix("Q", Q);
    // a) Check if Q is equal to its transpose.
    printf("a) Checking if Q == Q^T ...\n");
    double Q_T[2][2];
    transpose(Q, Q_T);
    printMatrix("Transpose Q^T", Q_T);
    printf("Result: Statement (a) is %s.\n\n", areMatricesEqual(Q
        , Q_T) ? "TRUE" : "FALSE");
```

C Code

```
// b) Check if Q is equal to its inverse.
printf("b) Checking if Q == Q^-1 ...\n");
double Q_inv[2][2];
if (inverse(Q, Q_inv)) { // Check if inverse exists before
    using it
    printMatrix("Inverse Q^-1", Q_inv);
    printf("Result: Statement (b) is %s.\n\n",
        areMatricesEqual(Q, Q_inv) ? "TRUE" : "FALSE");
} else {
    printf("Inverse does not exist.\n");
    printf("Result: Statement (b) is FALSE.\n\n");
}
```

C Code

```c
// c) & d) Check for full rank and column dependency using
    the determinant.
printf("c/d) Checking rank and column dependency...\n");
double det = determinant(Q);
printf("Determinant of Q = %.2f\n", det);
// If determinant is non-zero, it has full rank and
    independent columns.
if (fabs(det) > 1e-9) {
    printf("Result: Statement (c) is TRUE (Determinant is non
        -zero, so Q is of full rank).\n");
    printf("Result: Statement (d) is FALSE (Columns are
        linearly independent).\n\n");
}
```

C Code

```
else {
        printf("Result: Statement (c) is FALSE (Determinant is
            zero, so Q is not of full rank).\n");
        printf("Result: Statement (d) is TRUE (Columns are
            linearly dependent).\n\n");
    }
    printf("Conclusion: The only TRUE statement is (c).\n");

    return 0;
}
```

C Code

```c
void printMatrix(const char* name, double matrix[2][2]) {
    printf("%s = \n", name);
    printf(" | %6.2f %6.2f |\n", matrix[0][0], matrix[0][1]);
    printf(" | %6.2f %6.2f |\n", matrix[1][0], matrix[1][1]);
}

double determinant(double matrix[2][2]) {
    return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix
        [1][0];
}
```

C Code

```c
void transpose(double in[2][2], double out[2][2]) {
    out[0][0] = in[0][0];
    out[0][1] = in[1][0];
    out[1][0] = in[0][1];
    out[1][1] = in[1][1];
}
int inverse(double in[2][2], double out[2][2]) {
    double det = determinant(in);

    // A matrix is invertible if and only if its determinant is
        non-zero.
    if (fabs(det) < 1e-9) {
        return 0; // No inverse exists
    }
```

C Code

```
    double inv_det = 1.0 / det;
    out[0][0] = in[1][1] * inv_det;
    out[0][1] = -in[0][1] * inv_det;
    out[1][0] = -in[1][0] * inv_det;
    out[1][1] = in[0][0] * inv_det;

    return 1; // Success
}
```

C Code

```c
int areMatricesEqual(double A[2][2], double B[2][2]) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            // Use a small tolerance for floating-point comparison
            if (fabs(A[i][j] - B[i][j]) > 1e-9) {
                return 0; // Not equal
            }
        }
    }
    return 1; // Equal
}
```

Python and C Code

```
import ctypes
import math

# Define a 2x2 matrix type using ctypes (array of arrays)
Matrix2x2 = (ctypes.c_double * 2) * 2

def print_matrix(name, matrix):
    print(f"{name} = ")
    for i in range(2):
        print(f" | {matrix[i][0]:6.2f} {matrix[i][1]:6.2f} |")
def determinant(matrix):
    return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix
        [1][0]
```

Python and C Code

```
def transpose(in_matrix, out_matrix):
    out_matrix[0][0] = in_matrix[0][0]
    out_matrix[0][1] = in_matrix[1][0]
    out_matrix[1][0] = in_matrix[0][1]
    out_matrix[1][1] = in_matrix[1][1]

def inverse(in_matrix, out_matrix):
    det = determinant(in_matrix)
    if abs(det) < 1e-9:
        return False
    inv_det = 1.0 / det
```

Python and C Code

```
    out_matrix[0][0] = in_matrix[1][1] * inv_det
    out_matrix[0][1] = -in_matrix[0][1] * inv_det
    out_matrix[1][0] = -in_matrix[1][0] * inv_det
    out_matrix[1][1] = in_matrix[0][0] * inv_det
    return True
def are_matrices_equal(A, B):
    for i in range(2):
        for j in range(2):
            if abs(A[i][j] - B[i][j]) > 1e-9:
                return False
    return True
```

Python and C Code

```
def main():
    # Define matrix Q
    Q = Matrix2x2()
    Q[0][0], Q[0][1] = 1.0, -2.0
    Q[1][0], Q[1][1] = 2.0, 1.0
    print("Given Matrix Q:")
    print_matrix("Q", Q)

    # a) Check if Q == Q^T
    print("a) Checking if Q == Q^T ...")
    Q_T = Matrix2x2()
    transpose(Q, Q_T)
```

Python and C Code

```
    print_matrix("Transpose Q^T", Q_T)
    print(f"Result: Statement (a) is {'TRUE' if are_matrices_equal(Q, Q_T) else 'FALSE'}.\n")
    # b) Check if Q == Q^-1
    print("b) Checking if Q == Q^-1 ...")
    Q_inv = Matrix2x2()
    if inverse(Q, Q_inv):
        print_matrix("Inverse Q^-1", Q_inv)
        print(f"Result: Statement (b) is {'TRUE' if are_matrices_equal(Q, Q_inv) else 'FALSE'}.\n")
```

```python
    else:
        print("Inverse does not exist.")
        print("Result: Statement (b) is FALSE.\n")
    # c) & d) Check full rank and column dependency using determinant
    print("c/d) Checking rank and column dependency...")
    det = determinant(Q)
    print(f"Determinant of Q = {det:.2f}")
    if abs(det) > 1e-9:
        print("Result: Statement (c) is TRUE (Determinant is non-zero, so Q is of full rank).")
        print("Result: Statement (d) is FALSE (Columns are linearly independent).\n")
```

Python and C Code

```python
    else:
        print("Result: Statement (c) is FALSE (Determinant is
            zero, so Q is not of full rank).")
        print("Result: Statement (d) is TRUE (Columns are
            linearly dependent).\n")
    print("Conclusion: The only TRUE statement is (c).")

if __name__ == "__main__":
    main()
```