Date __/__/__

Name : Anshuman Gaonsindhe
BITS ID No: 2023ab05150
Section : 4.

Ans 1 a) Code Snippet for REF.

```
def perform_ref(matrix):
    # Converts a matrix to ref with pivot
    # normalization
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    Lead = 0

    for r in range(num_rows):
        if lead >= num_cols:
            break.

        i = r.
        While matrix[i][lead] == 0:
            i += 1
            if i == num_rows:
                i = r
                Lead = lead + 1
                if Lead == num_cols:
                    break
        swap_rows(matrix, i, r) # Pivoting fun^n
        pivot_value = matrix[r][lead]
        for i in range(len(matrix[r])):
            matrix[r][i] /= pivot_value
```

//vijeta//

Date / /

Name : Anshuman Gaonsindhe
Bits ID : 2023ab05150
Section : Sec 4.

1@ continued...

```
        for j in range(r+1, num_rows):
            if matrix[j][lead] != 0:
                add_multiple_of_row_to_row
                    (matrix, r, j, -matrix[j][lead])

        lead += 1

# Pivoting function, swaps two rows of matrix

def swap_rows(matrix, row1, row2):
    matrix[row1], matrix[row2]
        = matrix[row2], matrix[row1]

# Elimination function

def add_multiple_of_row_to_row(matrix,
        source_row, target_row, scalar):

    for i in range(len(matrix[target_row])):
        matrix[target_row][i] += scalar *
            matrix[source_row][i]
```

Input

Matrix A:

$$\begin{bmatrix} 3 & 1 & 4 & 6 & 8 \\ 8 & 6 & 5 & 1 & 8 \\ 8 & 6 & 8 & 6 & 1 \\ 2 & 7 & 2 & 7 & 6 \end{bmatrix}$$

Vector b:

$$\begin{bmatrix} 2 & -1 & 2 & 4 \end{bmatrix}$$

Output

REF :

$$\begin{bmatrix} 1 & 0 & 1 & 2 & 2 & 0 \\ 0 & 1 & -1 & -4 & -3 & -1 \\ 0 & 0 & 1 & -2 & -2 & 1 \\ 0 & 0 & 0 & 3 & 3 & 0 \end{bmatrix}$$

Date / /

Name: Anshuman Gaonsindhe
Bits ID: 2023 ab 05150
Section: 4.

1(a) continued...

Code Snippet for RREF.

```
def perform-rref(matrix):
# Converts a matrix to rref.
    perform_ref(matrix) # Perform REF first

    num-rows = len(matrix)

    for r in range(num_rows -1, -1, -1):
        for j in range(r):
            if matrix[j][r] != 0:
                add-multiple-of-row-to-row(matrix,
                                 r, j, -matrix[j][r])

        if matrix[r][r] != 0
            scale-row(matrix, r, 1/matrix[r][r])

# Scale the pivot to 1

def scale-row(matrix, row, scalar):
    ~~for (i~~
    for i in range(len(matrix[row])):
        matrix[row][i] *= scalar
```

Output -

RREF: $\begin{bmatrix} 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 1 & 0 & 0 & -2 & 0 \\ 0 & 0 & 1 & 0 & -7 & 0 \\ 0 & 0 & 0 & 1 & 3 & 0 \end{bmatrix}$

Date / /

Name: Anshuman Gaonsindhe
Bits ID: 2023ab05150
Section: 4.

Ans 1b) Code Snippet for pivot, non-pivot, particular solⁿ and solution to $Ax = 0$.

```
# Identify Pivot Columns on RREF matrix

def identify_pivot_columns(matrix)
    pivot_cols = [ ]
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if matrix[i][j] == 1 and
                    all(matrix[k][j] == 0 for k in range(i)):
                pivot_cols.append(j)
                break

    non_pivot_cols = [i for i in range(len(matrix[0]))
                        if i not in pivot_cols ]

    return pivot_cols, non_pivot_cols.
```

Input

Matrix A:

$$\begin{bmatrix} 9 & 5 & 4 & 1 & 1 \\ 10 & 1 & 10 & 3 & 7 \\ 4 & 6 & 9 & 1 & 9 \\ 8 & 6 & 4 & 7 & 1 \end{bmatrix}$$

Vector b:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

RREF A|b:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Output

Pivot Columns:
[0 1 2 3]

Non Pivot Cols:
[4, 5]

Date / /

Name : Anshuman Gaonsindhe
Bits Id : 2023ab05150
Section : 4

1(b) continued.

```
# Funⁿ for Particular solⁿ, homogenous solⁿ

def find_particular_solution(rref_matrix):
    solution = [0] * (len(rref_matrix[0]) - 1)
    for i in range(len(rref_matrix)):
        if 1 in rref_matrix[i]: # Check if leading 1
            solution[rref_matrix[i].index(1)]
                        = rref_matrix[i][-1]

    return solution

def find_homogenous_solution(rref_matrix):
    cols = len(matrix[0])
    pivot_colums, non_pivot_columns = identify_pivot_columns
                                      (rref_matrix)
    solutions = []
    for col in non_pivot_columns:
        solution = np.zeros((cols - 1, 1))
        if col < len(solution):
            solution[col] = 1
        for row, pivot_col in enumerate(pivot_columns):
            if pivot_col < len(solution):
                solution[pivot_col, 0] = - matrix[row][col]
        solutions.append(solution.flatten().tolist())
    return solutions
```

Output :
Particular Solution : [-0.153, 0.229, 0.200, 0.436, 0]

Date / /

Name : Anshuman Gaonsindhe
Bits ID : 2023 ab 05 150
Section : 4

1 (b) continues

Solutions to $Ax = 0$:

$$\begin{bmatrix} 0.433 & -0.076 & -1.151 & 0.084 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 0.153 & -0.229 & -0.200 & \cancel{-0.200} \; -0.436 & 0.0 \end{bmatrix}$$

Ans 1 (c) Outputs for a random 5 X 7 Matrix.

Input:

$$\text{Matrix } A: \begin{bmatrix} 2 & 6 & 8 & 3 & 5 & 4 & 7 \\ 1 & 3 & 8 & 1 & 6 & 2 & 10 \\ 3 & 8 & 7 & 4 & 5 & 5 & 6 \\ 2 & 7 & 3 & 6 & 1 & 7 & 9 \\ 3 & 7 & 5 & 1 & 5 & 1 & 9 \end{bmatrix}$$

$$\text{Vector } b: \begin{bmatrix} -1 \\ 2 \\ 3 \\ -2 \\ 4 \end{bmatrix}$$

Output:

$$\text{REF}: \begin{bmatrix} 1 & 3 & 4 & 1 & 2 & 2 & 3 & 0 \\ 0 & 1 & 5 & 0 & 2 & 1 & 4 & -.4 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 11 & 7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 7 & 3 \end{bmatrix}$$

Date ___/___/___

Name: Anshuman Gaonsindhe
Bits ID: 2023ab05150
Section: 4

1(c) continues

$$\text{RREF}: \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -26 & 9 \\ 0 & 1 & 0 & 0 & 0 & 0 & 11 & -4 \\ 0 & 0 & 1 & 0 & 0 & 0 & -4 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 7 & 3 \end{bmatrix}$$

Pivot Columns: [0, 1, 2, 3, 4]
(0 based Indexing)

Non-Pivot Columns: [5, 6, 7]

Particular Solution:
[9.108 -4.594 -1.729 2.351 3.027 0 0]

Solution to $Ax = 0$.
[-0.432 0.378 -0.081 -1.405 -0.108 1.0 0.0]

[26.891 -11.405 4.729 1.648 -7.027 0.0 1.0]

[-9.108 4.594 1.729 -2.351 -3.027 0.0 0.0]

General Solⁿ
[9.108, -4.594, -1.729, 2.351, 3.027, 0, 0]
[-0.432, 0.378, -0.081, -1.405, -0.180, 1.0, 0.0]
[26.891, -11.405, 4.729, 1.648, -7.027, 0.0, 0.0]
[-9.108, 4.594, 1.729, -2.351, -3.027, 0.0, 0.0]

Note: Substitute the solutions in $Ax=b$ & $Ax=0$, we will satisfy that the solⁿ are systems of linear eqⁿ

Date / /

Name : Anshuman Gaonsindhe
Bits Id : 2023ab05150
Section : 4.

Ans 2 (a) Code Snippet for elementry matrix & A=LU

```
# Generate Elementry Matrix

def generate-elementry-matrix(n, i, j, factor)
    E = [[1 if p==q else 0 for q is range(n)]
            for p is range(n)]

    E[j][i] = factor
    return E

# LU Decomposition

def lu-decomposition(A):
    n = len(A)
    L = [[0 for _ is range(n)] for _ is range(n)]
    U = [row.copy() for row is A]

    for i is range(n):
        for j is range(i+1, n):
            factor = U[j][i] / U[i][i]
            L[j][i] = factor
    # Applying Elementry Operations.
        E = generate elementry-matrix(n, i, j, -factor)

        U = multiply_matrices(E, U)
```

Date / /

Name: Anshuman Gaonsindhe
Bits Id :- 2023ab05150
Section :- 4.

2(a) continues

```
    for i in range(n):
        L[i][i] = 1

    return L, U.

# function to Multiply Matrices

def multiply_matrices(A, B):
    result = [[0 for _ in range(len(B[0]))]
              for _ in range(len(A))]

    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    return result.

# function to verify A = LU
def verify_lu_decomposition(A, L, U):
    reconstructed_A = multiply_matrices(L, U)
    for i in range(len(A)):
        for j in range(len(A[0])):
            if abs(A[i][j] - reconstructed_A[i][j]
                                > 1e-8:
                return False
    return True
```

Date / /

Name: Anshuman Gaonsindhe
Bits Id : 2023 ab 05150
Section : 4.

2 (a) continues.

Input (Random Matrix)

Matrix A: $\begin{bmatrix} 2 & 4 & 7 \\ 8 & 3 & 1 \\ 4 & 2 & 8 \end{bmatrix}$

Output

Elementry Matrix : 1
$\begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Elementry Matrix : 2
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix}$

Elementry Matrix : 3
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -0.46 & 1 \end{bmatrix}$

Lower Triangular Matrix, L : $\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 2 & 0.46 & 1 \end{bmatrix}$

Upper Triangular Matrix U : $\begin{bmatrix} 2 & 4 & 7 \\ 0 & -13 & -27 \\ 0 & 0 & 6.46 \end{bmatrix}$

LU : $\begin{bmatrix} 2 & 4 & 7 \\ 8 & 3 & 1 \\ 4 & 2 & 8 \end{bmatrix}$

$\boxed{A = LU}$

Date ___/___/___

Name: Anshuman Gaonsindhe
Bits Id: 2023 ab 05150
Section: 4.

Ans 2(b) Code Snippet for Cholesky's Decompⁿ

```
# Function to calculate Cholesky's Decompⁿ

def cholesky_decomposition(A):
    n = len(A)
    L = np.zeros((n, n))

    for i in range(n):
        for j in range(i+1):
            if i == j:
                L[i][j] = np.sqrt(A[i, i] - np.sum
                          (L[i, :j]**2))

            else:
                L[i, j] = (A[i, j] - np.sum(L[i, :j] *
                          L[j, :j])) / L[j, j]

    return L

# Function to verify cholesky's decomposition
def verify_cholesky_decomposition(A, L):
    reconstructed_A = np.dot(L, L.T)
    return np.allclose(A, reconstructed_A)
```

Input

Matrix A: $\begin{bmatrix} 75 & 67 & 52 \\ 67 & 65 & 63 \\ 52 & 63 & 90 \end{bmatrix}$

Lower Triangular Matrix L

$\begin{bmatrix} 8.66 & 0 & 0 \\ 7.73 & 2.26 & 0 \\ 6.00 & 7.29 & 0.86 \end{bmatrix}$

$\boxed{A = LL^T = \text{True}}$

vijeta

Date __/__/__

Name : Anshuman Gaonsindhe
Bits Id : 2023 ab 05150
Section : 4.

Ans 2 (c) Code Snippet for QR Decomposition

```
def qr_decomposition(A):
    m, n = A.shape
    Q = np.zeros((m, n))
    R = np.zeros((n, n))

    for j in range(n):
        v = A[:, j]

        for i in range(j):
            R[i, j] = np.dot(Q[:, i], A[:, j])
            v = v - R[i, j] * Q[:, i]

        R[j, j] = np.linalg.norm(v)
        Q[:, j] = v / R[j, j]

    return Q, R.
```

Input

Matrix A : $\begin{bmatrix} 8 & 5 & 4 \\ 3 & 4 & 6 \\ 9 & 2 & 1 \\ 9 & 8 & 9 \end{bmatrix}$

~~Matrix Q :~~ $\cancel{\begin{bmatrix} 0.52 & 0.03 & -0.63 \\ 0.03 & & \\ & & \\ & & \end{bmatrix}}$

Matrix Q : $\begin{bmatrix} 0.52 & 0.03 & -0.63 \\ 0.19 & 0.45 & 0.67 \\ 0.58 & -0.71 & 0.37 \\ 0.58 & 0.53 & -0.03 \end{bmatrix}$

Matrix R : $\begin{bmatrix} 15.32, & 9.2, & 9.13 \\ 0, & 4.8, & 6.9 \\ 0, & 0, & 1.5 \end{bmatrix}$

Verification QR = A : True

Date / /

Name: Anshuman Gaonsindhe
Bits Id: 2023 ab 05150
Section: 4.

Ans 2 (d) Random 5 x 4 Matrix

Matrix A:
$$\begin{bmatrix} 0.614 & 0.337 & 0.246 & 0.770 \\ 0.102 & 0.522 & 0.357 & 0.552 \\ 0.712 & 0.528 & 0.839 & 0.304 \\ 0.371 & 0.773 & 0.855 & 0.993 \\ 0.453 & 0.346 & 0.731 & 0.681 \end{bmatrix}$$

Matrix Q (Orthogonal):

$$\begin{bmatrix} -0.552 & 0.300 & 0.703 & -0.329 \\ -0.092 & -0.652 & 0.310 & 0.136 \\ -0.639 & 0.140 & -0.224 & 0.721 \\ -0.333 & 0.677 & -0.098 & -0.189 \\ -0.407 & 0.074 & -0.590 & -0.562 \end{bmatrix}$$

Matrix R: Upper Triangular.
$$\begin{bmatrix} -1.113 & -0.971 & -1.289 & -1.280 \\ 0 & -0.663 & -0.565 & -0.707 \\ 0 & 0 & -0.420 & 0.144 \\ 0 & 0 & 0 & -0.530 \end{bmatrix}$$

Diagonal Elements Of R.

$$\begin{bmatrix} -1.113 & -0.663 & -0.420 & -0.530 \end{bmatrix}$$

vijeta

Date / /

Name: Anshuman Gaonsindhe
Bits Id: 2023ab05150
Section: 4

2 (d) continues.

Observations on O/P:

- Orthoganal Matrix (Q)
  - The columns of Q form an orthoganal basis. The dot product of any two columns is approx. zero indicating orthogonality.

- Upper Triangular Matrix (R)
  - R is an upper triangular matrix with all the entries below the main diagonal being zero.

- Diagonal Elements of R:
  - The diagonal elements of R represent the scale or magnitude of the corresponding columns of the original matrix.
  - All diagonal elements are non-zero, indicating that the original matrix has linearly independent columns.
  - In Example, the output confirms that the 5×4 mat has been successfully decomposed into an Orthoganal Matrix Q & Upper Triangular matrix R.
  - The non-zero diagonal elements of R indicate the scale of original columns and the negative sign in the diagonal elements is common due to Graham-Schmidt process & doesn't affect Orthogonality of Q.