DATA SCIENCE

CYCLE 2

1. Create a three dimensional array specifying float data type and print it.

Program code

```
import numpy as np
print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")

depth = int(input("Enter the depth of the array: "))
rows = int(input("Enter the number of rows: "))
columns = int(input("Enter the number of columns: "))
three_dimensional_array = np.random.rand(depth, rows, columns).astype(np.float32)
print("Generated 3D array:")
print(three_dimensional_array)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter the depth of the array: 2
Enter the number of rows: 3
Enter the number of columns: 3
Generated 3D array:
[[[0.9555348  0.87073594 0.23248799]
  [0.84291285 0.65424937 0.27791497]
  [0.38574407 0.5464858  0.28472155]]

 [[0.87548447 0.5153113  0.15746117]
  [0.98902315 0.03388986 0.87948406]
  [0.7322822  0.7008028  0.29085734]]]
```

2. Create a 2 dimensional array (2X3) with elements belonging to complex data type and print it. Also display
    a. the no: of rows and columns
    b. dimension of an array
    c. reshape the same array to 3X2

Program code

```
import numpy as np
print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")

two_dimensional_complex_array = np.array([[1 + 2j, 2 + 3j, 3 + 4j],
                                          [4 + 5j, 5 + 6j, 6 + 7j]], dtype=complex)
print("2D Complex Array:")
print(two_dimensional_complex_array)
num_rows, num_cols = two_dimensional_complex_array.shape
array_dimension = two_dimensional_complex_array.ndim
reshaped_array = two_dimensional_complex_array.reshape(3, 2)

print("\na. Number of rows:", num_rows)
print("   Number of columns:", num_cols)
print("b. Dimension of the array:", array_dimension)
print("c. Reshaped array (3x2):")
print(reshaped_array)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
2D Complex Array:
[[1.+2.j 2.+3.j 3.+4.j]
 [4.+5.j 5.+6.j 6.+7.j]]

a. Number of rows: 2
   Number of columns: 3
b. Dimension of the array: 2
c. Reshaped array (3x2):
[[1.+2.j 2.+3.j]
 [3.+4.j 4.+5.j]
 [5.+6.j 6.+7.j]]
```

3. Familiarize with the functions to create
a. an uninitialized array
b. array with all elements as 1,
c. all elements as 0

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
uninitialized_array = np.empty((3, 3))
print("Uninitialized Array:")
print(uninitialized_array)

ones_array = np.ones((2, 4))  # You can specify the shape you want
print("Array with All Elements as 1:")
print(ones_array)

zeros_array = np.zeros((3, 2))  # You can specify the shape you want
print("Array with All Elements as 0:")
print(zeros_array)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Uninitialized Array:
[[1.49501340e-316 0.00000000e+000 6.90756464e-310]
 [6.90756459e-310 6.90755593e-310 6.90755593e-310]
 [6.90756469e-310 6.90755593e-310 3.95252517e-322]]
Array with All Elements as 1:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
Array with All Elements as 0:
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

4. Create an one dimensional array using **arange** function containing 10 elements.

   Display

   a. First 4 elements
   b. Last 6 elements
   c. Elements from index 2 to 7

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
my_array = np.arange(10)
first_4_elements = my_array[:4]
print("a. First 4 elements:", first_4_elements)
last_6_elements = my_array[-6:]
print("b. Last 6 elements:", last_6_elements)
elements_2_to_7 = my_array[2:8]
print("c. Elements from index 2 to 7:", elements_2_to_7)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
a. First 4 elements: [0 1 2 3]
b. Last 6 elements: [4 5 6 7 8 9]
c. Elements from index 2 to 7: [2 3 4 5 6 7]
```

5. Create an 1D array with **arange** containing first 15 even numbers as elements

a. Elements from index 2 to 8 with step 2(also demonstrate the same using slice function)
b. Last 3 elements of the array using negative index
c. Alternate elements of the array
d. Display the last 3 alternate elements

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
even_numbers = np.arange(2, 31, 2)
subset_a = even_numbers[2:9:2]
subset_a_slice = even_numbers[2:9:2]
print("a. Elements from index 2 to 8 with step 2:")
print(subset_a)
print("Slice Function Equivalent:")
print(subset_a_slice)

last_three = even_numbers[-3:]
print("\nb. Last 3 elements of the array using negative index:")
print(last_three)

alternate_elements = even_numbers[::2]
```

```
print("\nc. Alternate elements of the array:")
print(alternate_elements)

last_three_alternate = even_numbers[-3::2]
print("\nd. Last 3 alternate elements:")
print(last_three_alternate)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
a. Elements from index 2 to 8 with step 2:
[ 6 10 14 18]
Slice Function Equivalent:
[ 6 10 14 18]

b. Last 3 elements of the array using negative index:
[26 28 30]

c. Alternate elements of the array:
[ 2  6 10 14 18 22 26 30]

d. Last 3 alternate elements:
[26 30]
```

6. Create a 2 Dimensional array with 4 rows and 4 columns.
a. Display all elements excluding the first row
b. Display all elements excluding the last column
c. Display the elements of 1st and 2nd column in 2nd and 3rd row
d. Display the elements of 2nd and 3rd column
e. Display 2nd and 3rd element of 1st row
f. Display the elements from indices 4 to 10 in descending order(use –values)

Program code

```
import numpy as np
```

```
print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
array_2d = np.array([[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12],
 [13, 14, 15, 16]])

print("a. All elements excluding the first row:")
print(array_2d[1:])

print("\nb. All elements excluding the last column:")
print(array_2d[:, :-1])

print("\nc. Elements of 1st and 2nd column in 2nd and 3rd row:")
print(array_2d[1:3, :2])

print("\nd. Elements of 2nd and 3rd column:")
print(array_2d[:, 1:3])

print("\ne. 2nd and 3rd element of 1st row:")
print(array_2d[0, 1:3])

print("\nf. Elements from indices 4 to 10 in descending order:")
print(array_2d.flatten()[4:11][::-1])
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
a. All elements excluding the first row:
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

b. All elements excluding the last column:
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]

c. Elements of 1st and 2nd column in 2nd and 3rd row:
[[ 5  6]
 [ 9 10]]

d. Elements of 2nd and 3rd column:
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]

e. 2nd and 3rd element of 1st row:
[2 3]

f. Elements from indices 4 to 10 in descending order:
[11 10  9  8  7  6  5]
```

7. Create two 2D arrays using array object and
   a. Add the 2 matrices and print it
   b. Subtract 2 matrices
   c. Multiply the individual elements of matrix
   d. Divide the elements of the matrices
   e. Perform matrix multiplication
   f. Display transpose of the matrix
   g. Sum of diagonal elements of a matrix

Program code

```
import numpy as np


print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
matrix1 = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
matrix2 = np.array([[9, 8, 7],
                   [6, 5, 4],
                   [3, 2, 1]])
# a. Add the two matrices and print the result
matrix_sum = matrix1 + matrix2
print("Matrix Addition:")
print(matrix_sum)


# b. Subtract the two matrices and print the result
matrix_diff = matrix1 - matrix2
print("\nMatrix Subtraction:")
print(matrix_diff)
```

```
# c. Multiply the individual elements of the matrices
elementwise_product = matrix1 * matrix2
print("\nElement-wise Multiplication:")
print(elementwise_product)

# d. Divide the elements of the matrices (assuming no division by zero)
elementwise_division = matrix1 / matrix2
print("\nElement-wise Division:")
print(elementwise_division)

# e. Perform matrix multiplication (dot product)
matrix_product = np.dot(matrix1, matrix2)
print("\nMatrix Multiplication:")
print(matrix_product)

# f. Display the transpose of a matrix
matrix1_transpose = np.transpose(matrix1)
print("\nTranspose of Matrix 1:")
print(matrix1_transpose)

# g. Calculate the sum of diagonal elements of a matrix
diagonal_sum = np.trace(matrix1)
print("\nSum of Diagonal Elements of Matrix 1:", diagonal_sum)
```

<u>Output</u>

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Matrix Addition:
[[10 10 10]
 [10 10 10]
 [10 10 10]]

Matrix Subtraction:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]

Element-wise Multiplication:
[[ 9 16 21]
 [24 25 24]
 [21 16  9]]

Element-wise Division:
[[0.11111111 0.25       0.42857143]
 [0.66666667 1.         1.5       ]
 [2.33333333 4.         9.        ]]

Matrix Multiplication:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]

Transpose of Matrix 1:
[[1 4 7]
 [2 5 8]
 [3 6 9]]

Sum of Diagonal Elements of Matrix 1: 15
```

8. Demonstrate the use of insert() function in 1D and 2D array

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
arr_1d = np.array([1, 2, 3, 4, 5])

# User input for value and index
value_to_insert = int(input("Enter the value to insert: "))
index_to_insert = int(input(f"Enter the index to insert {value_to_insert} at: "))

# Using insert() to insert the value at the specified index
new_arr_1d = np.insert(arr_1d, index_to_insert, value_to_insert)

print("Original 1D Array:")
print(arr_1d)

print(f"\nArray after inserting {value_to_insert} at index {index_to_insert}:")
print(new_arr_1d)

# Creating a 2D array
arr_2d = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

# User input for values and index
values_to_insert = list(map(int, input("Enter values to insert in the format 'x y z':
").split()))
index_to_insert = int(input(f"Enter the index to insert {values_to_insert} at: "))

# Using insert() to insert the values at the specified index along axis 0
new_arr_2d = np.insert(arr_2d, index_to_insert, values_to_insert, axis=0)

print("\nOriginal 2D Array:")
print(arr_2d)

print(f"\nArray after inserting {values_to_insert} at index {index_to_insert} along axis
0:")
print(new_arr_2d)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter the value to insert: 5
Enter the index to insert 5 at: 1
Original 1D Array:
[1 2 3 4 5]

Array after inserting 5 at index 1:
[1 5 2 3 4 5]
Enter values to insert in the format 'x y z': 2 3 5
Enter the index to insert [2, 3, 5] at: 2

Original 2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Array after inserting [2, 3, 5] at index 2 along axis 0:
[[1 2 3]
 [4 5 6]
 [2 3 5]
 [7 8 9]]
```

9. Demonstrate the use of diag() function in 1D and 2D array.(use both square matrix and matrix with different dimensions)

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")

arr_1d = np.array(list(map(int, input("Enter elements for 1D array separated by spaces: ").split())))

# Using diag() (should be empty for 1D array)
```

```
diag_1d = np.diag(arr_1d)
print("\nOriginal 1D Array:")
print(arr_1d)
print("\nDiagonal Elements (empty for 1D array):")
print(diag_1d)

# User input for a square 2D array
n = int(input("Enter the size of the square matrix: "))
square_matrix = np.array([list(map(int, input().split())) for _ in range(n)])

# Using diag() to extract diagonal elements of the square matrix
diag_square_matrix = np.diag(square_matrix)
print("\nOriginal Square Matrix:")
print(square_matrix)
print("\nDiagonal Elements of Square Matrix:")
print(diag_square_matrix)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter elements for 1D array separated by spaces: 1 2 3 4

Original 1D Array:
[1 2 3 4]

Diagonal Elements (empty for 1D array):
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
Enter the size of the square matrix: 3
1 2 3
4 5 6
7 8 9

Original Square Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Diagonal Elements of Square Matrix:
[1 5 9]
```

10. Create a square matrix with random integer values(use randint()) and use appropriate functions to find:
    i) inverse
    ii) rank of matrix
    iii) Determinant
    iv) transform matrix into 1D array
    v) eigen values and vectors

<u>Program code</u>

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
size = int(input("Enter the size of the square matrix: "))
```

```
# User input for the elements of the matrix
print(f"Enter {size}x{size} matrix:")
user_matrix = np.array([list(map(int, input().split())) for _ in range(size)])

print("\nUser Input Matrix:")
print(user_matrix)

# i) Inverse of the matrix
try:
    inverse_matrix = np.linalg.inv(user_matrix)
    print("\nInverse of the Matrix:")
    print(inverse_matrix)
except np.linalg.LinAlgError:
    print("\nMatrix is singular, and its inverse does not exist.")

# ii) Rank of the matrix
rank_matrix = np.linalg.matrix_rank(user_matrix)
print("\nRank of the Matrix:", rank_matrix)

# iii) Determinant of the matrix
determinant_matrix = np.linalg.det(user_matrix)
print("\nDeterminant of the Matrix:", determinant_matrix)

# iv) Transform matrix into a 1D array
flattened_matrix = user_matrix.flatten()
print("\nMatrix as 1D Array:")
print(flattened_matrix)

# v) Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(user_matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter the size of the square matrix: 2
Enter 2x2 matrix:
2 4
5 6

User Input Matrix:
[[2 4]
 [5 6]]

Inverse of the Matrix:
[[-0.75   0.5  ]
 [ 0.625 -0.25 ]]

Rank of the Matrix: 2

Determinant of the Matrix: -7.999999999999998

Matrix as 1D Array:
[2 4 5 6]

Eigenvalues:
[-0.89897949  8.89897949]

Eigenvectors:
[[-0.8097086  -0.50158596]
 [ 0.58683216 -0.86510781]]
```

11. Create a matrix X with suitable rows and columns
    i)Display the cube of each element of the matrix using different methods(use multiply(), *, power(),**)
    ii)Display identity matrix of the given square matrix.
    iii)Display each element of the matrix to different powers.

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))

X = np.empty((rows, cols), dtype=int)
for i in range(rows):
    for j in range(cols):
        X[i, j] = int(input(f"Enter the element at row {i+1}, column {j+1}: "))

cubed_matrix_multiply = X * X * X

cubed_matrix_power = np.power(X, 3)
cubed_matrix_double_asterisk = X ** 3

# Display the results
print("\nCube of each element using multiplication:\n", cubed_matrix_multiply)
print("\nCube of each element using power function:\n", cubed_matrix_power)
print("\nCube of each element using double asterisk (**):\n",
cubed_matrix_double_asterisk)

if rows == cols:
    identity_matrix = np.identity(rows)
    print("\nIdentity matrix of X:\n", identity_matrix)
else:
    print("\nX is not a square matrix, so it doesn't have an identity matrix.")

powers = [2, 3, 4]
powered_matrices = [np.power(X, p) for p in powers]

for i, p in enumerate(powers):
    print(f"\nMatrix raised to the power {p}:\n", powered_matrices[i])
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter the number of rows: 3
Enter the number of columns: 2
Enter the element at row 1, column 1: 1
Enter the element at row 1, column 2: 2
Enter the element at row 2, column 1: 3
Enter the element at row 2, column 2: 4
Enter the element at row 3, column 1: 5
Enter the element at row 3, column 2: 6

Cube of each element using multiplication:
 [[  1   8]
 [ 27  64]
 [125 216]]

Cube of each element using power function:
 [[  1   8]
 [ 27  64]
 [125 216]]

Cube of each element using double asterisk (**):
 [[  1   8]
 [ 27  64]
 [125 216]]

X is not a square matrix, so it doesn't have an identity matrix.
```

```
Matrix raised to the power 2:
 [[ 1  4]
 [ 9 16]
 [25 36]]

Matrix raised to the power 3:
 [[  1   8]
 [ 27  64]
 [125 216]]

Matrix raised to the power 4:
 [[   1   16]
 [  81  256]
 [ 625 1296]]
```

12. Create a matrix Y with same dimension as X and perform the operation X^2+2Y

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))

X = np.empty((rows, cols), dtype=float)
print("Enter first matrix")
for i in range(rows):
    for j in range(cols):
        X[i, j] = float(input(f"Enter the value for X[{i}][{j}]: "))

Y = np.empty((rows, cols), dtype=float)

print("Enter second matrix")
for i in range(rows):
    for j in range(cols):
```

```
        Y[i, j] = float(input(f"Enter the value for Y[{i}][{j}]: "))

    result = X**2 + 2*Y

    print("Matrix X:")
    print(X)
    print("Matrix Y:")
    print(Y)
    print("Result (X^2 + 2Y):")
    print(result)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Enter the number of rows: 2
Enter the number of columns: 2
Enter first matrix
Enter the value for X[0][0]: 4
Enter the value for X[0][1]: 5
Enter the value for X[1][0]: 6
Enter the value for X[1][1]: 7
Enter second matrix
Enter the value for Y[0][0]: 1
Enter the value for Y[0][1]: 2
Enter the value for Y[1][0]: 9
Enter the value for Y[1][1]: 3
Matrix X:
[[4. 5.]
 [6. 7.]]
Matrix Y:
[[1. 2.]
 [9. 3.]]
Result (X^2 + 2Y):
[[18. 29.]
 [54. 55.]]
```

13. Define matrices A with dimension 5x6 and B with dimension 3x3. Extract a sub matrix of dimension 3x3 from A and multiply it with B**.** Replace the extracted sub matrix in A with the matrix obtained after multiplication

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
A = np.array([[1, 2, 3, 4, 5, 6],
         [7, 8, 9, 10, 11, 12],
         [13, 14, 15, 16, 17, 18],
         [19, 20, 21, 22, 23, 24],
         [25, 26, 27, 28, 29, 30]])

B = np.array([[2, 3, 4],
         [5, 6, 7],
         [8, 9, 10]])

submatrix_A = A[:3, :3]

result = np.dot(submatrix_A, B)

A[:3, :3] = result

# Display the updated matrix A
print("Updated Matrix A:")
print(A)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Updated Matrix A:
[[ 36  42  48   4   5   6]
 [126 150 174  10  11  12]
 [216 258 300  16  17  18]
 [ 19  20  21  22  23  24]
 [ 25  26  27  28  29  30]]
```

14. Given 3 Matrices A, B and C. Write a program to perform matrix multiplication of the 3 matrices.

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
A = np.array([[1, 2, 3],
              [4, 5, 6]])

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])

C = np.array([[13, 14],
              [15, 16]])

result = np.dot(np.dot(A, B), C)

print("Result of Matrix Multiplication (A * B * C):")
print(result)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Result of Matrix Multiplication (A * B * C):
[[1714 1836]
 [4117 4410]]
```

15. Write a program to check whether given matrix is symmetric or Skew Symmetric.

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
def is_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, transpose)

def is_skew_symmetric(matrix):
    transpose = np.transpose(matrix)
    return np.array_equal(matrix, -transpose)

matrix = np.array([[0, 1, -2],
                   [-1, 0, 3],
                   [2, -3, 0]])

if is_symmetric(matrix):
    print("The matrix is symmetric.")
elif is_skew_symmetric(matrix):
    print("The matrix is skew-symmetric (antisymmetric).")
else:
    print("The matrix is neither symmetric nor skew-symmetric.")
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
The matrix is skew-symmetric (antisymmetric).
```

16. Given a matrix-vector equation AX=b. Write a program to find out the value of X using **solve(),** given A and b as below

$X=A^{-1}$ b.

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$$

Note: Numpy provides a function called solve for solving such equations.

Program code

```
import numpy as np


print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
A = np.array([[2, 3, -1],
              [1, 2, 1],
              [3, 1, -2]])
b = np.array([7, 3, 8])
try:
   X = np.linalg.solve(A, b)
   print("Solution X:")
   print(X)
except np.linalg.LinAlgError:
   print("Matrix A is singular. The system of equations may not have a unique solution.")
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Solution X:
[ 2.   0.8 -0.6]
```

17. Write a program to perform the SVD of a given matrix A. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.

Use the function: **numpy.linalg.svd()**

Singular value Decomposition

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.

The SVD of mxn matrix A is given by the formula $A = U\Sigma V^T$

Program code

```
import numpy as np

print("MUHAMMED HISHAM KP , 41 , MCA-2022-24")
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

U, S, Vt = np.linalg.svd(A)

A_hat = U @ np.diag(S) @ Vt

print("Original Matrix A:")
print(A)
print("\nSingular Values:")
print(S)
print("\nReconstructed Matrix A_hat:")
print(A_hat)
```

Output

```
MUHAMMED HISHAM KP , 41 , MCA-2022-24
Original Matrix A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Singular Values:
[1.68481034e+01 1.06836951e+00 4.41842475e-16]

Reconstructed Matrix A_hat:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```