**LAB CYCLE 2**

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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

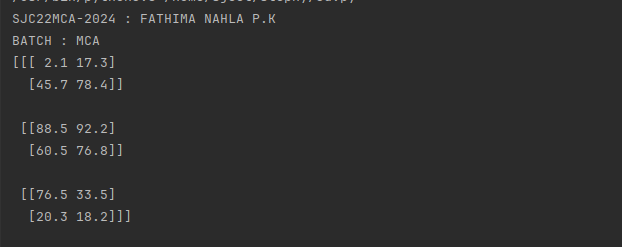
print("BATCH : MCA")

import numpy as np

arr=np.array([[[2.1,17.3], [45.7, 78.4]], [[88.5, 92.2], [60.5, 76.8]],[[76.5,33.5],[20.3,18.2]]],dtype='float')

print(arr)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

x = np.array([[1+2j, 2+3j, 3+4j], [4+1j, 5+2j, 6+3j]], dtype=complex)

print (x)

rows,columns=x.shape

print("number of rows",rows)

print("number of columns",columns)

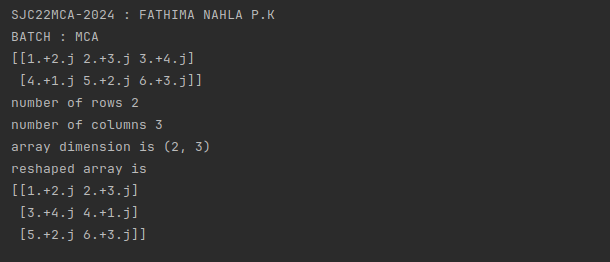
print("array dimension is",x.shape)

reshape=x.reshape(3,2)

print("reshaped array is")

print(reshape)

OUTPUT:



3. Familiarize with the functions to create

a) an uninitialized array

b) array with all elements as 1,

c) all elements as 0

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

uninitialized\_array = np.empty(shape=(2, 3))

print("Uninitialized Array:")

print(uninitialized\_array)

ones\_array = np.ones(shape=(2, 3))

print("Array with All Elements as 1:")

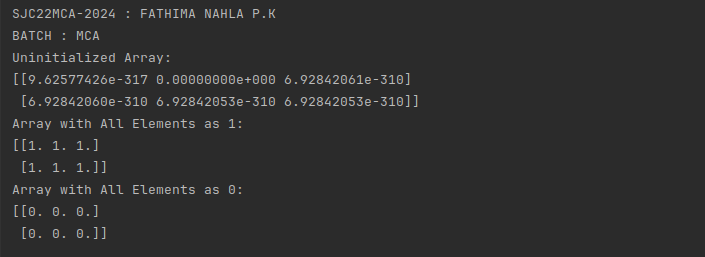
print(ones\_array)

zeros\_array = np.zeros(shape=(2, 3))

print("Array with All Elements as 0:")

print(zeros\_array)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

one\_dimensional\_array = np.arange(10)

first\_4\_elements = one\_dimensional\_array[:4]

last\_6\_elements = one\_dimensional\_array[-6:]

elements\_2\_to\_7 = one\_dimensional\_array[2:8]

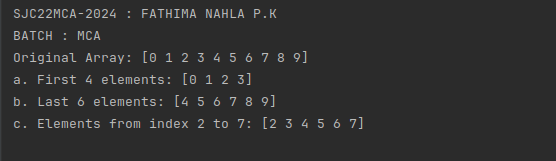
print("Original Array:", one\_dimensional\_array)

print("a. First 4 elements:", first\_4\_elements)

print("b. Last 6 elements:", last\_6\_elements)

print("c. Elements from index 2 to 7:", elements\_2\_to\_7)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

array=np.arange(2,32,2)

print(array)

step\_size=array[2:9:2]

print("a.index 2 to 8 with step 2:",step\_size)

result\_slice=array[2:9:2]

print("a.1 using slice function:",result\_slice)

last\_3\_elements = array[-3:]

print("b. Last 3 elements:", last\_3\_elements)

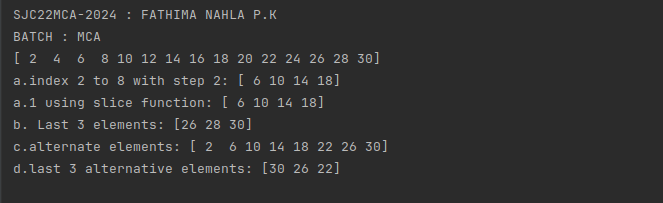
result=array[::2]

print("c.alternate elements:",result)

result=array[-1::-2][:3]

print("d.last 3 alternative elements:",result)

OUTPUT:



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 1 st and 2 nd column in 2 nd and 3 rd row

d. Display the elements of 2 nd and 3 rd column

e. Display 2 nd and 3 rd element of 1 st row

f. Display the elements from indices 4 to 10 in descending order(use

–values)

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

twodim\_array = np.array([

[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 16]

])

result=twodim\_array[1:]

print("a.elements excluding the first row:")

print(result)

result=twodim\_array[:,:-1]

print("b.elements excluding the last column:")

print(result)

result=twodim\_array[1:3,0:2]

print("c.elements of 1 st and 2 nd column in 2 nd and 3 rd row")

print(result)

result=twodim\_array[:,1:3]

print("d.elements of 2 nd and 3 rd column")

print(result)

result=twodim\_array[0,1:3]

print("e.2 nd and 3 rd element of 1 st row")

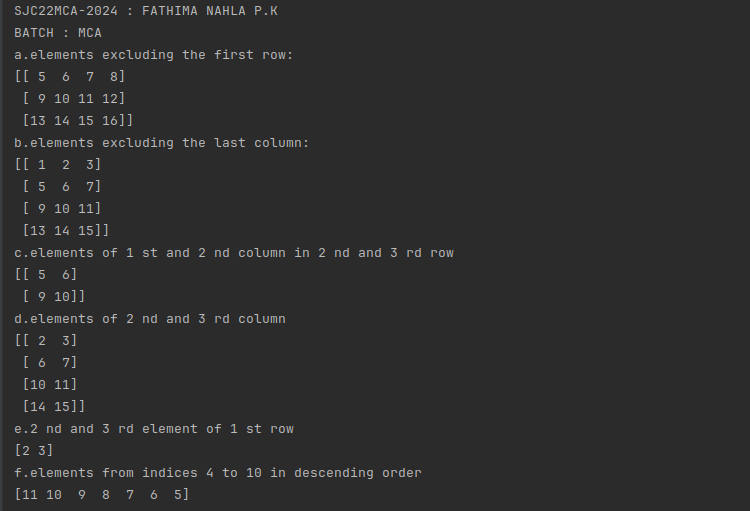
print(result)

result=twodim\_array.flatten()[10:3:-1]

print("f.elements from indices 4 to 10 in descending order")

print(result)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

matrix1 = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

matrix2 = np.array([[9, 8, 7],

[6, 5, 4],

[3, 2, 1]])

result = matrix1 + matrix2

print("Matrix Addition:")

print(result)

result = matrix1 - matrix2

print("\nMatrix Subtraction:")

print(result)

result = matrix1 \* matrix2

print("\nMatrix Multiplication:")

print(result)

result = matrix1 / matrix2

print("\nMatrix Division:")

print(result)

result = np.dot(matrix1, matrix2)

print("\nMatrix Multiplication:")

print(result)

result = matrix1.T

print("\nMatrix Transpose:")

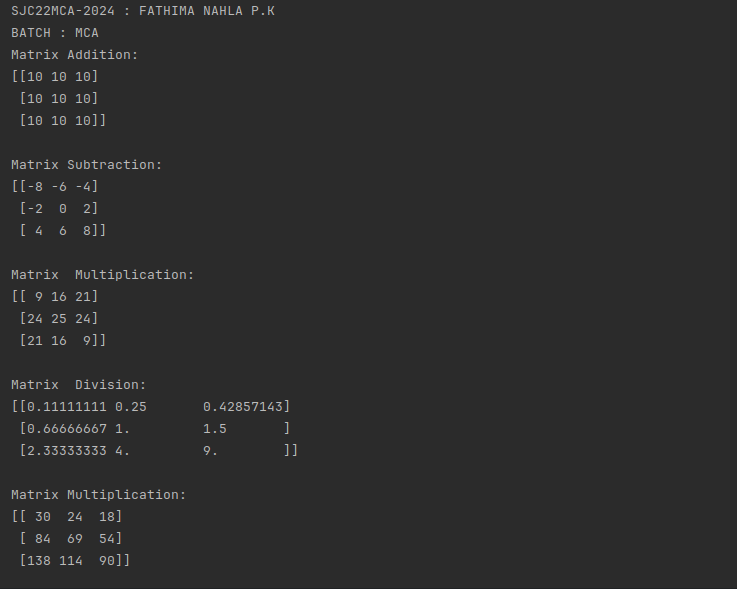
print(result)

diagonal\_sum = np.trace(matrix1)

print("\nSum of Diagonal Elements:")

print(diagonal\_sum)

OUTPUT:





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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])

new\_arr1 = np.insert(arr1, 2, 6)

print("Original 1D Array:")

print(arr1)

print("\n1D Array After Insertion:")

print(new\_arr1)

arr2d = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

new\_arr2d = np.insert(arr2d, 1, [10, 11, 12], axis=0)

new\_arr2d = np.insert(new\_arr2d, 2, [0, 0, 0, 0], axis=1)

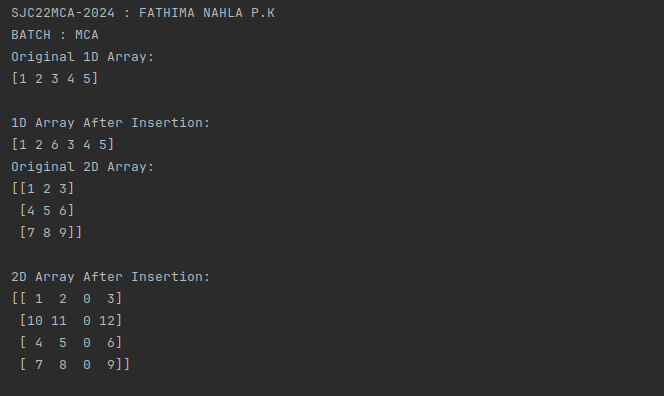
print("Original 2D Array:")

print(arr2d)

print("\n2D Array After Insertion:")

print(new\_arr2d)

OUTPUT:



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

and matrix with different dimensions)

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

print("1D array:")

arr1d=np.array([1, 2, 3, 4, 5])

diagonal\_matrix=np.diag(arr1d)

print("Original 1D Array:")

print(arr1d)

print("Diagonal Matrix:")

print(diagonal\_matrix)

print("2D array:")

arr2=np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

diagonal\_matrix=np.diag(arr2)

print("Original 1D Array:")

print(arr2)

print("Diagonal Matrix:")

print(diagonal\_matrix)

square\_matrix=np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

diagonal\_elements=np.diag(square\_matrix)

print("Original Square Matrix:")

print(square\_matrix)

print("Diagonal Elements (1D Array):")

print(diagonal\_elements)

non\_square\_matrix=np.array([[1, 2, 3],

[4, 5, 6]])

diagonal\_elements=np.diag(non\_square\_matrix)

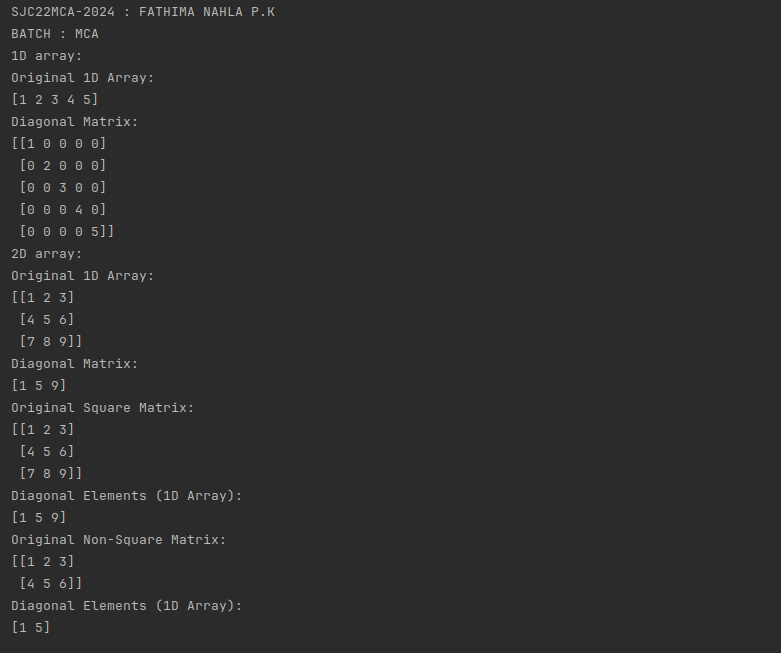
print("Original Non-Square Matrix:")

print(non\_square\_matrix)

print("Diagonal Elements (1D Array):")

print(diagonal\_elements)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

matrix\_size = 3

min\_value = 1

max\_value = 10

random\_matrix = np.random.randint(min\_value, max\_value + 1, size=(matrix\_size, matrix\_size))

print("Random Square Matrix:")

print(random\_matrix)

inverse\_matrix = np.linalg.inv(random\_matrix)

print("\nInverse Matrix:")

print(inverse\_matrix)

matrix\_rank = np.linalg.matrix\_rank(random\_matrix)

print("\nRank of the Matrix:", matrix\_rank)

matrix\_determinant = np.linalg.det(random\_matrix)

print("\nDeterminant of the Matrix:", matrix\_determinant)

matrix\_1d = random\_matrix.flatten()

print("\nMatrix as 1D Array:")

print(matrix\_1d)

eigenvalues, eigenvectors = np.linalg.eig(random\_matrix)

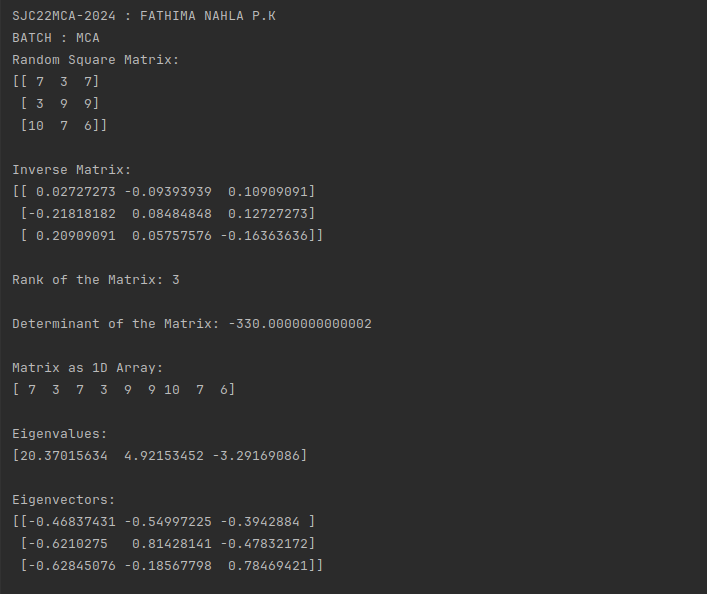
print("\nEigenvalues:")

print(eigenvalues)

print("\nEigenvectors:")

print(eigenvectors)

OUTPUT:



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.

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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

X = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# i) Display the cube of each element of the matrix using different methods

# Using np.multiply()

cubed\_X1 = np.multiply(X, np.multiply(X, X))

# Using the \* operator

cubed\_X2 = X \* X \* X

# Using np.power()

cubed\_X3 = np.power(X, 3)

# Using the \*\* operator

cubed\_X4 = X \*\* 3

print("Cube of each element using np.multiply():")

print(cubed\_X1)

print("\nCube of each element using \* operator:")

print(cubed\_X2)

print("\nCube of each element using np.power():")

print(cubed\_X3)

print("\nCube of each element using \*\* operator:")

print(cubed\_X4)

# ii) Display identity matrix of the given square matrix

identity\_matrix = np.identity(X.shape[0])

print("\nIdentity Matrix of X:")

print(identity\_matrix)

# iii) Display each element of the matrix to different powers

exponents = np.array([[2, 3, 4],

[3, 2, 1],

[0.5, 0.25, 0.1]])

powered\_X = np.power(X, exponents)

print("\nMatrix X to Different Powers:")

print(powered\_X)

# Create matrix Y with the same dimensions as X

Y = np.random.randint(1, 10, size=X.shape)

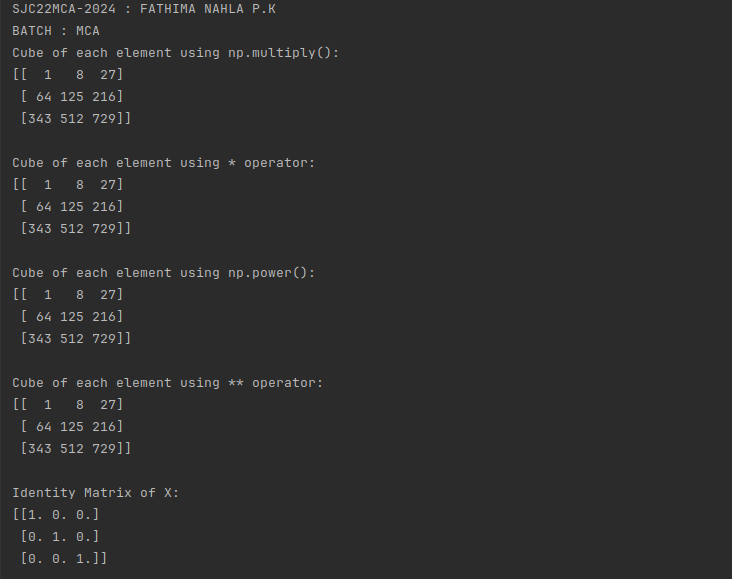
# Perform the operation X^2 + 2Y

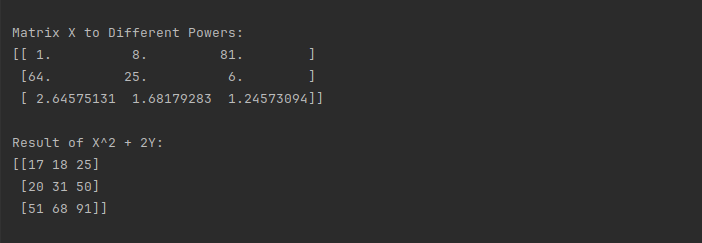
result = np.power(X, 2) + 2 \* Y

print("\nResult of X^2 + 2Y:")

print(result)

OUTPUT:





12. 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.

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

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)

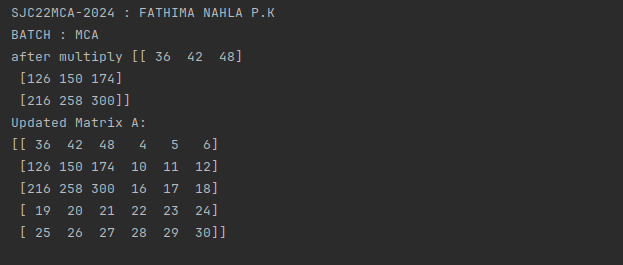
print("after multiply",result)

A[:3, :3] = result

print("Updated Matrix A:")

print(A)

OUTPUT:



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

the 3 matrices.

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

A = np.array([[1, 2, 3],

[4, 5, 6]])

B = np.array([[7, 8],

[9, 10],

[11, 12]])

C = np.array([[13, 14],

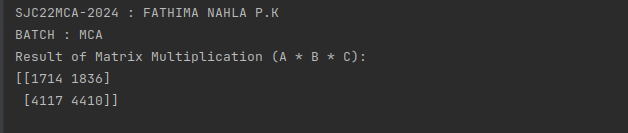
[15, 16]])

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

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

print(result)

OUTPUT:



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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

def symmetric(matrix):

transpose = np.transpose(matrix)

return np.array\_equal(matrix, transpose)

def 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 symmetric(matrix):

print("The matrix is symmetric.")

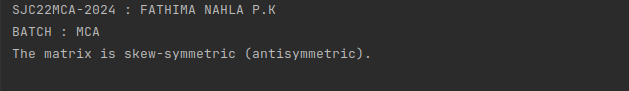
elif skew\_symmetric(matrix):

print("The matrix is skew-symmetric (antisymmetric).")

else:

print("The matrix is neither symmetric nor skew-symmetric.")

OUTPUT:



15. 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.

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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

A = np.array([[2, 1, -2],

[3, 0, 1],

[1, 1, -1]])

b = np.array([-3, 5, 2])

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:



16. 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

CODE:

print("SJC22MCA-2024 : FATHIMA NAHLA P.K")

print("BATCH : MCA")

import numpy as np

# Define the matrix A (replace with your own matrix)

A = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# Perform SVD on matrix A

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

# Reconstruct the original matrix from the three matrices obtained in SVD

reconstructed\_A = np.dot(U, np.dot(np.diag(S), VT))

# Print the original matrix A, SVD components, and the reconstructed matrix

print("Original Matrix A:")

print(A)

print("\nMatrix U:")

print(U)

print("\nSingular Values S:")

print(S)

print("\nMatrix VT (Transpose of V):")

print(VT)

print("\nSVD Reconstructed Matrix A:")

print(reconstructed\_A)

OUTPUT:

