1. 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**

import numpy as np

import numpy as nf

from numpy.linalg import eig

mat = np.random.randint(10, size=(3, 3))

array = nf.random.randint(10, size=(3, 3))

print(mat)

M\_inverse = np.linalg.inv(mat)

print("inverse of the array")

print(M\_inverse)

rank = np.linalg.matrix\_rank(mat)

print("Rank of the given Matrix ")

print(rank)

det = np.linalg.det(mat)

print("determinant of the given Matrix ")

print(det)

arr = mat.flatten()

print("transform matrix to array ")

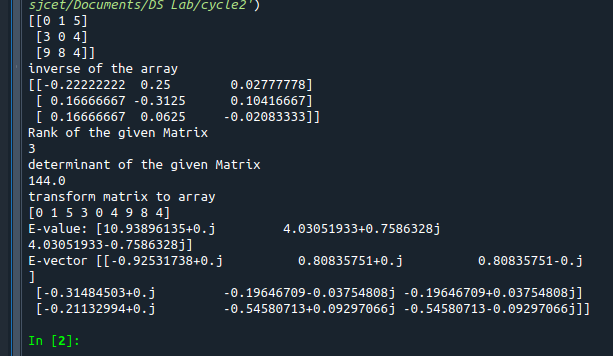
print(arr)

w, v = eig(array)

print('E-value:', w)

print('E-vector', v)

**output**



2.

2Create 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.

iv) Create a matrix Y with same dimension as X and perform the operation X2+2Y

**code**

import numpy as np

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

print(arr1)

print("using power()")

print(pow(arr1, 3))

print("using multiply()")

print(np.multiply(arr1,(arr1\*arr1)))

print("using \*")

print(arr1\*arr1\*arr1)

print("using \*\*")

print(arr1\*\*3)

b = np.identity(3, dtype = int)

print("Identity matrix:\n", b)

out = np.power(arr1, arr1)

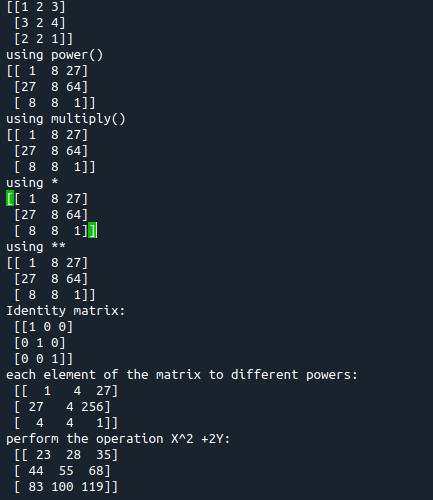
print("each element of the matrix to different powers:\n",out)

x = np.arange(1,10).reshape(3,3)

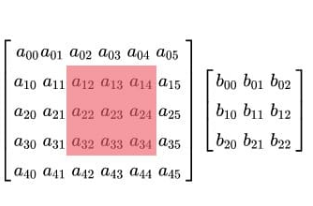
y = np.arange(11,20).reshape(3,3)

print("perform the operation X^2 +2Y: \n",np.add((np.power(x,2)),(np.multiply(y,2))))

**output**



3. Multiply a matrix with a submatrix of another matrix and replace the same in larger matrix.



**Code**

import numpy as np

A = np.array([[6, 1, 1,6,3],

[4, -2, 5,1,3],

[2, 8, 7,7,8],

[6, 1, 1,6,3],

[2, 8, 7,7,8]])

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

[3, 0, 1],

[1, 1, -1]])

print("Mat A=\n",A)

print("Mat B=\n",B)

C=A[:3, :3]

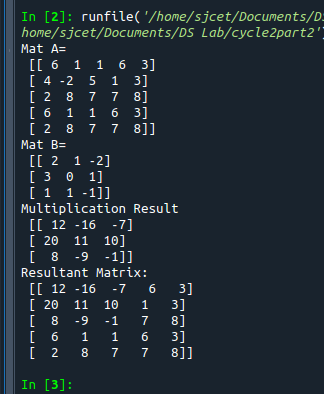
res = np.dot(B,C)

print("Multiplication Result\n",res)

A[:3,:3]=res[:3,:3]

print("Resultant Matrix:\n",A)

**output**



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

**Code**

import numpy as np

m1 = np.random.randint(20, size=(2, 2))

print(m1)

m2 = np.random.randint(20, size=(2, 2))

print(m2)

m3 = np.random.randint(20, size=(2, 2))

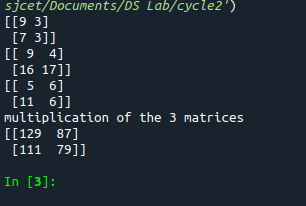
print(m3)

print("multiplication of the 3 matrices")

m4 = np.dot(m1,m2,m3)

print(m4)

**output**



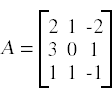
5.Write a program to check whether given matrix is symmetric or Skew Symmetric.

Solving systems of equations with numpy

One of the more common problems in linear algebra is solving a matrix-vector equation.

Here is an example. We seek the vector x that solves the equation

A X = b

Where  

And X=A-1 b.

Numpy provides a function called solve for solving such eauations.

**Code**

import numpy as np

A = np.array([[6, 1, 1],

[4, -2, 5],

[2, 8, 7]])

inv=np.transpose(A)

print (inv)

neg=np.negative(A)

comparison = A == inv

comparison1 = inv== neg

equal\_arrays = comparison.all()

skew=comparison1.all()

if equal\_arrays :

print("Symmetric")

else:

print("not Symmetric")

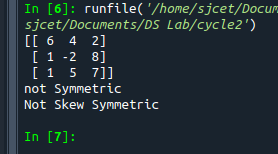
if skew:

print("Skew Symmetric")

else:

print("Not Skew Symmetric")

**output**



6.Write a program to find out the value of X using **solve(),** given **A** and **b** as above

**code**

import numpy as np

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

[3, 0, 1],

[1, 1, -1]])

b=np.array([[3],

[5],

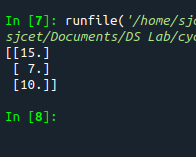
[-2]])

inv=np.linalg.inv(A)

x=np.linalg.solve(inv,b)

print(x)

**output**

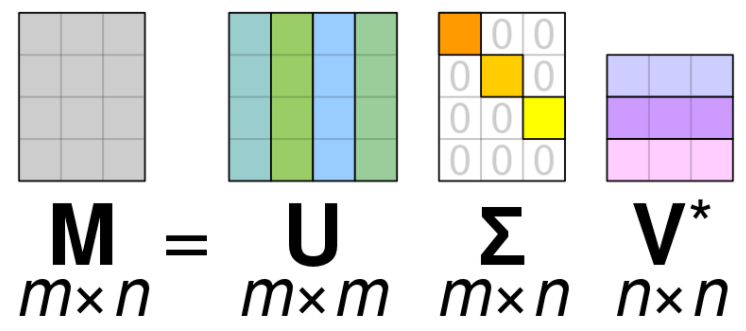


Singular value Decomposition

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.

**M= U ∑V^T**



* **M**-is original matrix we want to decompose
* **U**-is left singular matrix (columns are left singular vectors). **U** columns contain eigenvectors of matrix **MM**ᵗ
* **Σ**-is a diagonal matrix containing singular (eigen) values.
* **V**-is right singular matrix (columns are right singular vectors). **V** columns contain eigenvectors of matrix **M**ᵗ**M**

**Numpy** provides a function for performing svd, which decomposes the given matrix into 3 matrices.

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

**Code**

from numpy import array

from scipy.linalg import svd

from numpy import diag

from numpy import dot

from numpy import zeros

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print(A)

# SVD

U, s, VT = svd(A)

print("first" ,U)

print("second",s)

print("3rd" ,VT)

Sigma = zeros((A.shape[0], A.shape[1]))

# populate Sigma with n x n diagonal matrix

Sigma[:A.shape[1], :A.shape[1]] = diag(s)

# reconstruct matrix

B = U.dot(Sigma.dot(VT))

print(B)

**output**

