LAB CYCLE-2

PART-2

***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***

**PROGRAM**

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("Eigen values and vectors")

print('E-value:', w)

print('E-vector', v)

**OUTPUT**

[[4 8 2]

[9 5 5]

[5 5 2]]

**Inverse of the array**

[[-0.41666667 -0.16666667 0.83333333]

[ 0.19444444 -0.05555556 -0.05555556]

[ 0.55555556 0.55555556 -1.44444444]]

**Rank of the given Matrix**

3

**Determinant of the given Matrix**

36.000000000000014

**Transform matrix to array**

[4 8 2 9 5 5 5 5 2]

**Eigen values and vectors**

E-value: [11.9507873 +0.j 1.02460635+1.20850184j 1.02460635-1.20850184j]

E-vector [[-0.38278787+0.j -0.69581594+0.j -0.69581594-0.j ]

[-0.22028461+0.j 0.64089555+0.19482937j 0.64089555-0.19482937j]

[-0.89718902+0.j 0.00944972-0.25893105j 0.00944972+0.25893105j]]

***2. 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.***

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

***PROGRAM***

*import numpy as np*

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

*print(x)*

*print("Display the cube of each element of the matrix using different methods")*

*print(np.power(x,3))*

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

*print(x\*x\*x)*

*print(x\*\*3)*

*print("Display identity matrix of the given square matrix.")*

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

*print(b)*

*print(" Display each element of the matrix to different powers.")*

*out=np.power(x,x)*

*print(out)*

*print("Create a matrix Y with same dimension as X and perform the operation X^2 +2Y")*

*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***

*[[ 2 3 4]*

*[ 5 6 7]*

*[ 8 9 10]]*

***Display the cube of each element of the matrix using different methods***

*[[ 8 27 64]*

*[ 125 216 343]*

*[ 512 729 1000]]*

*[[ 8 27 64]*

*[ 125 216 343]*

*[ 512 729 1000]]*

*[[ 8 27 64]*

*[ 125 216 343]*

*[ 512 729 1000]]*

*[[ 8 27 64]*

*[ 125 216 343]*

*[ 512 729 1000]]*

***Display identity matrix of the given square matrix.***

*[[1 0 0]*

*[0 1 0]*

*[0 0 1]]*

*Display each element of the matrix to different powers.*

*[[ 4 27 256]*

*[ 3125 46656 823543]*

*[ 16777216 387420489 10000000000]]*

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

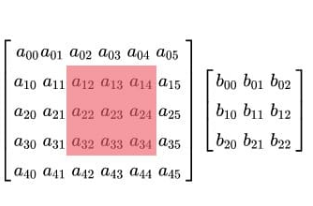
*perform the operation X^2 +2Y:*

*[[ 26 33 42]*

*[ 53 66 81]*

*[ 98 117 138]]*

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



**PROGRAM**

import numpy as np

np.random.seed(42)

A=np.random.randint(0,10,size=(5,6))

B=np.random.randint(0,10,size=(3,3))

print("Matris A:\n{},shape={}\n".format(A,A.shape))

print("Matrix B:\n{},shape={}\n".format(B,B.shape))

C=A[1:4,2:5]@B

A[1:4,2:5]

print("Matrix A after submatrix multiplication:\n{},shape={}\n".format(A, A.shape))

**OUTPUT**

Matris A:

[[6 3 7 4 6 9]

[2 6 7 4 3 7]

[7 2 5 4 1 7]

[5 1 4 0 9 5]

[8 0 9 2 6 3]],shape=(5, 6)

Matrix B:

[[8 2 4]

[2 6 4]

[8 6 1]],shape=(3, 3)

Matrix A after submatrix multiplication:

[[6 3 7 4 6 9]

[2 6 7 4 3 7]

[7 2 5 4 1 7]

[5 1 4 0 9 5]

[8 0 9 2 6 3]],shape=(5, 6)

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

**PROGRAM**

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**

*[[14 0]*

*[15 13]]*

*[[ 7 15]*

*[18 7]]*

*[[12 17]*

*[ 8 16]]*

*multiplication of the 3 matrices*

*[[ 98 210]*

*[339 316]]*

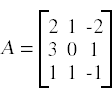
***5.Write a program to check whether given matrix is symmetric or Skew Symmetric.***

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.

**PROGRAM**

import numpy as np

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

[2, -6, 5],

[5, 5, 8]])

print("Original Matrix\n",A)

inv=np.transpose(A)

print ("Transpose matrix\n",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**

Original Matrix

[[ 6 1 1]

[ 2 -6 5]

[ 5 5 8]]

Transpose matrix

[[ 6 2 5]

[ 1 -6 5]

[ 1 5 8]]

Not Symmetric

Not Skew Symmetric

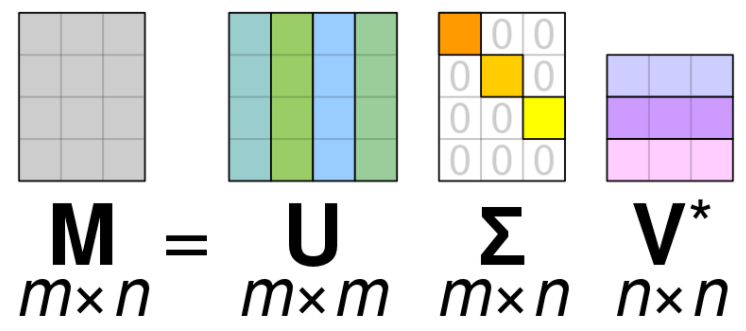
***6.Write a program to find out the value of X using solve(), given A and b as above***

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.

**PROGRAM**

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**

[[1 2]

[3 4]

[5 6]]

first [[-0.2298477 0.88346102 0.40824829]

[-0.52474482 0.24078249 -0.81649658]

[-0.81964194 -0.40189603 0.40824829]]

second [9.52551809 0.51430058]

3rd [[-0.61962948 -0.78489445]

[-0.78489445 0.61962948]]

[[1. 2.]

[3. 4.]

[5. 6.]]

***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.***

**PROGRAM**

from numpy import array

from scipy.linalg import svd

from numpy import dot

from numpy import diag

# define a matrix

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

print(A)

# SVD

U, s, VT = svd(A)

print(U)

print(s)

print(VT)

Sigma=diag(s)

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

print(B)

**OUTPUT**

[[1 2 1]

[3 4 2]

[5 6 4]]

[[-0.22783392 -0.67993976 -0.69697463]

[-0.50860448 -0.52728523 0.6806554 ]

[-0.83030909 0.50956081 -0.22568695]]

[10.56182278 0.58610151 0.3230861 ]

[[-0.55910736 -0.70744799 -0.43233818]

[ 0.48798477 -0.70236223 0.51822598]

[ 0.67027594 -0.07876951 -0.73791974]]

[[1. 2. 1.]

[3. 4. 2.]

[5. 6. 4.]]