CYCLE-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

CODE

```
import numpy as np
from numpy import random
x=random.randint(100, size=(3,3))
inv = np.linalg.inv(x)
print("Matrix:\n",x)
print("Inverse:\n",inv)
rank=np.linalg.matrix_rank(x)
print("Rank:",rank)
print ("Determinant:",np.linalg.det(x))
arr = x.flatten()
print("Matrix to array:\n",arr)
w, v = np.linalg.eig(x)
print("Eigen value:\n",w)
print("Eigen Vector:\n",v)
```

OUTPUT

```
Matrix:
 [[78 66  1]
 [65 21 10]
 [15 63 63]]
Inverse:
 [[-0.00342161  0.02021863 -0.00315499]
 [ 0.01947802 -0.02418829  0.00353024]
 [-0.01866335  0.01937433  0.01309397]]
Rank: 3
Determinant: -202536.0000000003
Matrix to array:
 [78 66  1 65 21 10 15 63 63]
Eigen value:
 [-25.92582736 125.85172975  62.07409761]
Eigen Vector:
 [[-0.47449916  0.64425209 -0.14036951]
 [ 0.75404373  0.45781704  0.0188726 ]
 [-0.45416803  0.61265227  0.9899193 ]]
```

OR

```
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

```
[[3 1 9]
 [4 2 8]
 [2 2 8]]
inverse of the array
[[ 0.   0.5 -0.5]
 [-0.8  0.3  0.6]
 [ 0.2 -0.2  0.1]]
Rank of the given Matrix
3
determinant of the given Matrix
19.999999999999996
transform matrix to array
[3 1 9 4 2 8 2 2 8]
E-value: [ 6.       -2.244998 10.244998]
E-vector [[ 1.          0.02284217 -0.88453379]
 [ 0.         -0.87936938 -0.25394881]
 [ 0.          0.47559198 -0.3912927 ]]
```

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 X^2+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))))
```

<u>OUTPUT</u>

```
[[1 2 3]
 [3 2 4]
 [2 2 1]]
using power()
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
using multiply()
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
using *
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
using **
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
Identity matrix:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
each element of the matrix to different powers:
 [[  1   4  27]
perform the operation X^2 +2Y:
 [[ 23  28  35]
 [ 44  55  68]
 [ 83 100 119]]
```

OR

```
import numpy as np

matrix=np.random.randint(0,10,4).reshape(2,2)
print("Display the cube of each element of the matrix using different methods (use multiply(), *, power(),**)")
x=np.power(matrix,3)
print("power()",x)

y=np.multiply(matrix,(matrix*matrix))
print("multiply()")
print(y)

z=matrix*matrix*matrix
print("**")
print(z)

cube=matrix*3
```

```
print("*")
print(cube)

print("Display identity matrix of the given square matrix.")
identity=np.identity(2,dtype=int)
print(identity)

print("Display each element of the matrix to different powers.")
dpow=np.power(matrix,matrix)
print(dpow)

print("Create a matrix Y with same dimension as X and perform the operation X^2 +2Y")
a=np.add((np.power(x,2)),(np.multiply(y,2)))
print(a)
```

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

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

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

```
Mat A=
 [[ 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]]
Mat B=
 [[ 2  1 -2]
 [ 3  0  1]
 [ 1  1 -1]]
Multiplication Result
 [[ 12 -16  -7]
 [ 20  11  10]
 [  8  -9  -1]]
Resultant Matrix:
 [[ 12 -16  -7   6   3]
 [ 20  11  10   1   3]
 [  8  -9  -1   7   8]
 [  6   1   1   6   3]
 [  2   8   7   7   8]]
```

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("1 st matrix \n",m1)
m2 = np.random.randint(20, size=(2, 2))
print("2nd matrix \n",m2)
```

```
m3 = np.random.randint(20, size=(2, 2))
print("3rd matrix \n",m3)
print("multiplication of the 3 matrices")
m4 = np.dot(m1,m2,m3)
print(m4)
```

<u>OUTPUT</u>

```
1 st matrix
 [[12  0]
 [17 12]]
2nd matrix
 [[ 7  5]
 [13 11]]
3rd matrix
 [[ 1  5]
 [ 7 12]]
multiplication of the 3 matrices
[[ 84  60]
 [275 217]]
```

OR

<u>CODE</u>

```
import numpy as np
M1 = np.array([[3, 6], [4, 2]])
M2 = np.array([[9, 2], [1, 2]])
M3=np.array([[2,4],[3,1]])
Mul = M1.dot(M2)
mul1=M3.dot(Mul)
print("Matrix1:\n",M1)
print("Matrix2:\n",M2)
print("Matrix3:\n",M3)
print("multiplication of 3 matrices")
print(mul1)
```

<u>OUTPUT</u>

```
Matrix1:
 [[3 6]
 [4 2]]
Matrix2:
 [[9 2]
 [1 2]]
Matrix3:
 [[2 4]
 [3 1]]
multiplication of 3 matrices
[[218  84]
 [137  66]]
```

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

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  4  2]
 [ 1 -2  8]
 [ 1  5  7]]
not Symmetric
Not 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 $A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix}$ $\mathbf{b} = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$

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

Numpy provides a function called solve for solving such eauations.

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

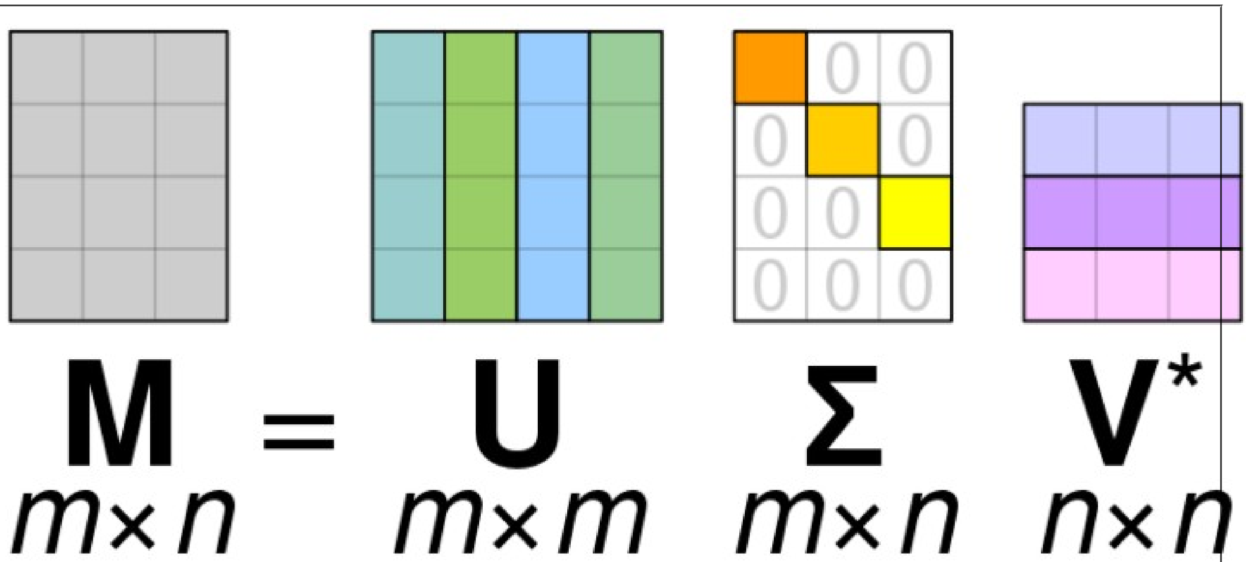
```
[[15.]
 [ 7.]
 [10.]]
```

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.

<u>CODE</u>

```
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.]]
```