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

```
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("Square matrix \n",mat)

M_inverse = np.linalg.inv(mat)
print("Inverse of the matrix\n",M_inverse)

rank = np.linalg.matrix_rank(mat)
print("Rank of the given Matrix \n",rank)

det= np.linalg.det(mat)
print("Determinant of the given Matrix \n",det)

arr=mat.flatten()
print("Transform matrix to 1D array \n",arr)

w,v=eig(array)
print('Eigen value \n', w)
print('Eigen vector \n', v)
```

```
Square matrix
 [[3 4 2]
 [6 5 0]
 [7 8 9]]
Inverse of the matrix
 [[-0.81818182  0.36363636  0.18181818]
 [ 0.98181818 -0.23636364 -0.21818182]
 [-0.23636364 -0.07272727  0.16363636]]
Rank of the given Matrix
 3
Determinant of the given Matrix
 -54.999999999999964
Transform matrix to 1D array
 [3 4 2 6 5 0 7 8 9]
Eigen value
 [ 8.80165105  0.11679014 -2.91844118]
Eigen vector
 [[-0.50675971 -0.60678131 -0.07810574]
 [-0.68511425  0.79299868 -0.75150289]
 [-0.52327149  0.0544935   0.65508999]]
```

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

```
import numpy as np
mat =np.array([[1, 2, 3],[3,2,4],[2,2,1]])
print("Matrix is....\n",mat)

print("Cubes using *")
print(mat*mat*mat)

print("Cubes using **")
print(mat**3)

print("Cubes using multiply()")
print(np.multiply(mat,(mat*mat)))

print("Cubes using power()")
print(np.power(mat,3))
print(pow(mat, 3))


b = np.identity(3, dtype = int)
print("Identity matrix:\n", b)

out = np.power(mat, mat)
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("matrix x \n", x ,"\n Matrix y\n",y)
print("perform the operation X∧2 +2Y: \n",np.add((np.power(x,2)),
(np.multiply(y,2))))
```

```
Matrix is....
 [[1 2 3]
 [3 2 4]
 [2 2 1]]
Cubes using *
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
Cubes using **
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
Cubes using multiply()
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
Cubes using power()
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
[[ 1  8 27]
 [27  8 64]
 [ 8  8  1]]
```

```
[ 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]
 [ 27   4 256]
 [  4   4   1]]
matrix x
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
 Matrix y
 [[11 12 13]
 [14 15 16]
 [17 18 19]]
perform the operation X^2 +2Y:
 [[ 23  28  35]
 [ 44  55  68]
 [ 83 100 119]]
```

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

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

```
import numpy as np
mat = np.array([[6, 1, 1, 4],
                [1, 2, 5, 2],
                [1, 5, 7, 3],
                [3, 2, 4, 1]])
print("Original Matrix....\n",mat)
sub = mat[1:3, 1:3]
print("Sub matrix....\n",sub)
mat2 = np.array([[1, 4],
                [3, 2]])
print("Second matrix is....\n",mat2)
mul=np.dot(sub,mat2)
print("Multiplication...\n",mul)
mat[1:3, 1:3] = mul
print("Initial matrix after replacement..\n",mat)
```

```
Original Matrix....
 [[6 1 1 4]
 [1 2 5 2]
 [1 5 7 3]
 [3 2 4 1]]
Sub matrix....
 [[2 5]
 [5 7]]
Second matrix is....
 [[1 4]
 [3 2]]
Multiplication...
 [[17 18]
 [26 34]]
Initial matrix after replacement..
 [[ 6  1  1  4]
 [ 1 17 18  2]
 [ 1 26 34  3]
 [ 3  2  4  1]]
```

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

```
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\n",mul1)
```

```
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.**

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

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

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

Numpy provides a function called solve for solving such eauations.

```
import numpy as np
A = np.array([[6, 1, 1],
              [1, -2, 5],
              [1, 5, 7]])
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")
```

```
Original Matrix
 [[ 6  1  1]
 [ 1 -2  5]
 [ 1  5  7]]
Transpose matrix
 [[ 6  1  1]
 [ 1 -2  5]
 [ 1  5  7]]
Symmetric
Not Skew Symmetric
```

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

```
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)
```

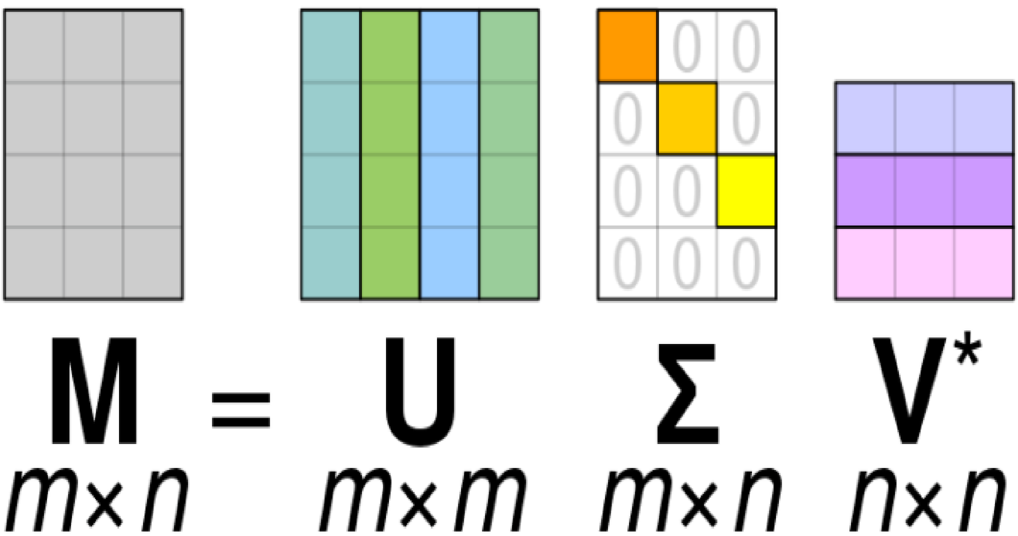
```
[[15.]
 [ 7.]
 [10.]]

In [41]:
```

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 ΣV∧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.**

```
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)
```

```
[[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.]]
```