D**ATA SCIENCE & MACHINE LEARNING**

**LAB CYCLE 2.2**

**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

arr1 = np.arange(10, 16)

print("1D ARRAY ")

print("\nThe array is: ", arr1)

obj = 2

value = 40

arr = np.insert(arr1, obj, value, axis=None)

print("\nAfter inserting the new array is: ")

print(arr)

print("\nShape of the new array is : ", np.shape(arr))

print("\n2D ARRAY ")

arr1 = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9), (50, 51, 52)])

print("\nThe array is: ")

print(arr1)

print("\nThe shape of the array is: ", np.shape(arr1))

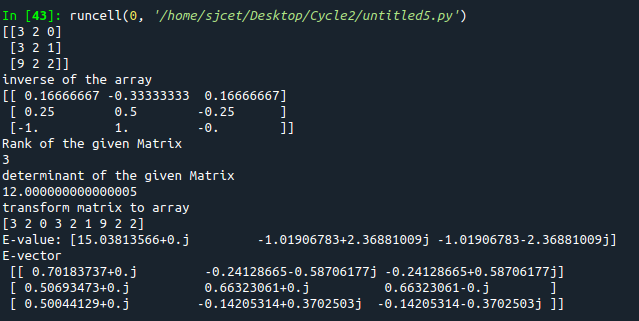
a = np.insert(arr1, 1, [[50], [100]], axis=0)

print("\nNew array is : ")

print(a)

print("\nShape of the array is: ", np.shape(a))

**Output**



**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**

**Program**

import numpy as np

arr1=np.arange(4,8).reshape(2,2)

print("\narray\n",arr1)

#1

print("\ncube using power\n")

b=np.power(arr1,3)

print(b)

f=arr1\*\*3

print("\n",f)

print("\ncube using multiply\n")

c=np.multiply(arr1,arr1)

d=np.multiply(c,arr1)

print("\n",d)

e=arr1\*arr1\*arr1

print("\n",e)

#2

print("\nidentity matrix\n")

h=np.identity(2)

print("\n",h)

#3

print("\nDisplay each element of the matrix to different powers.")

i=np.power(arr1[0][0],2)

j=np.power(arr1[0][1],3)

k=np.power(arr1[1][0],4)

m=np.power(arr1[1][1],5)

print("\n",i)

print("\n",j)

print("\n",k)

print("\n",m)

#4

print("\nCreate a matrix Y with same dimension as X and perform the operation X 2 +2Y")

arr2=np.arange(0,4).reshape((2,2))

print("\n",arr2)

n=np.power(arr1,2)

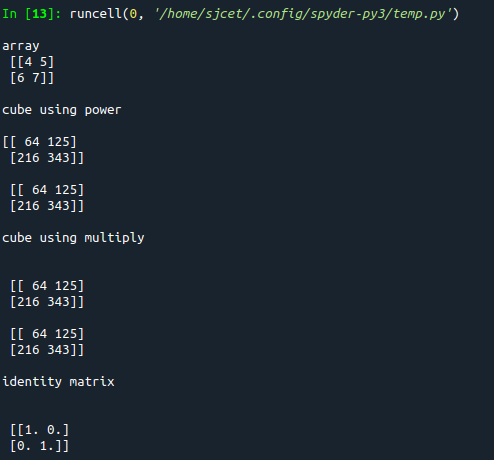
o=np.multiply(2,arr2)

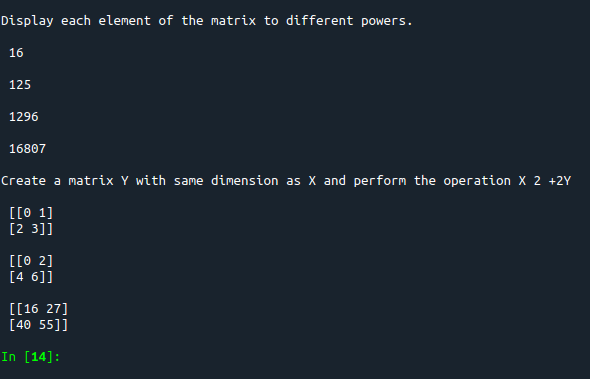
print("\n",o)

p=np.add(n,o)

print("\n",p)

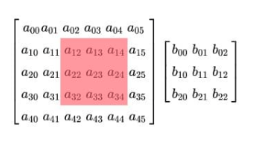
**Output**





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

**matrix.**

****

**Program**

import numpy as np

print("\nLarge matrix")

arr1=np.arange(1,37).reshape((6,6))

print(arr1)

print("\nsmall matrix")

arr2=np.arange(1,10).reshape((3,3))

print(arr2)

print("\ncutout portion")

a=arr1[0:3,1:4]

print(a)

print("\nmultiplying with smaller matrix")

c=np.multiply(a,arr2)

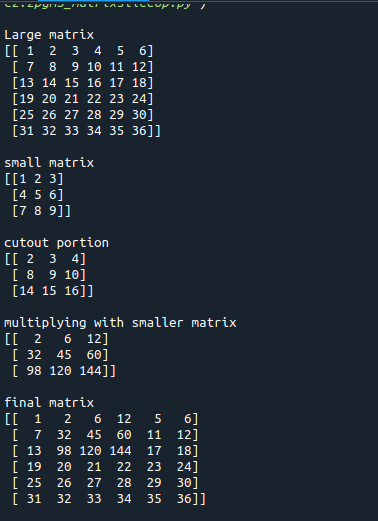
print(c)

print("\nfinal matrix")

arr1[0:3,1:4]=c

print(arr1)

**Output**



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

**the 3 matrices.**

**Program**

import numpy as np

#matrix multiplication of 3 matrices.

arr1=np.arange(1,10).reshape((3,3))

arr2=np.arange(11,20).reshape((3,3))

arr3=np.arange(21,30).reshape((3,3))

print("\n1stmatrix")

print(arr1)

print("\n2ndmatrix")

print(arr2)

print("\n3rdmatrix")

print(arr3)

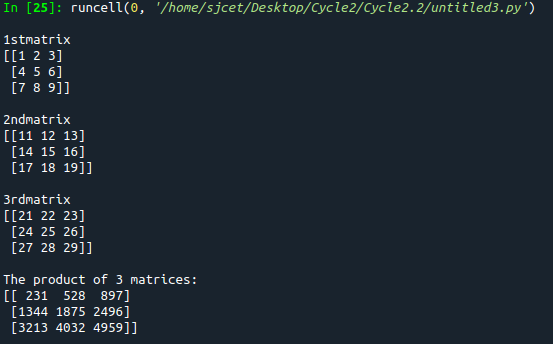
a=np.multiply(arr1,arr2)

b=np.multiply(a,arr3)

print("\nThe product of 3 matrices:")

print(b)

**Output**



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

**Program**

import numpy as np

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

[2, 1, 2],

[3, 2, 1]])

x=np.transpose(a)

print(a)

c=np.array\_equal(a, x)

print("symmetric or not:" ,c)

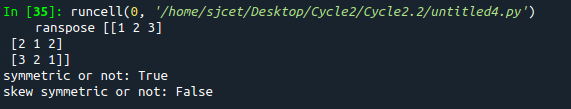
b=np.negative(a)

print(b)

d=np.array\_equal(x,b)

print("skew symmetric or not:",d)

**Output**



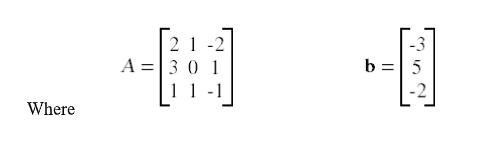
**6. Write a program to find out the value of X using solve(), given A and b as above**

**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**

****

**And X=A -1 b.**

**Numpy provides a function called solve for solving such eauations.**

**Program**

import numpy as np

a=np.array([[2,1,-2],[3,0,1],[1,1,-1]])

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

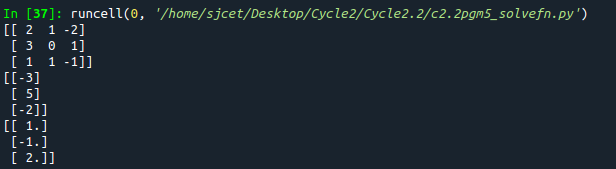
print(a)

print(b)

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

print(x)

**Output**



**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.**

**Singular value Decomposition**

**Matrix decomposition, also known as matrix factorization, involves describing a given**

**matrix using its constituent elements.**

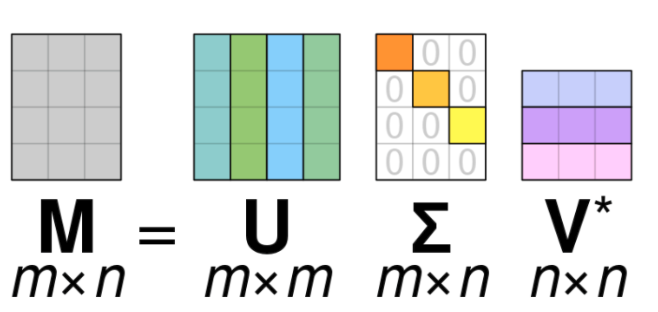
**The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for**

**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 MMᵗ**

** 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 MᵗM**

**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 dot

from numpy import diag

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

print(A)

U, s, VT = svd(A)

print(U)

print(s)

print(VT)

Sigma=diag(s)

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

print(B)

**Output**

