Python Assignment - 2

Kunal Keshav Damame
121901050

Question 1:
AIM

1. Create a Python program that generates a random matrix based on the following user inputs obtained during run-time:

   - matrix dimensions
   - probability distribution
   - matrix structure - identity, diagonal, bi-diagonal, block diagonal, symmetric, skew-symmetric, Toeplitz, circulant, sparse, stochastic, and doubly stochastic
   - if block diagonal or sparse, input block-sizes or sparsity factor (number of non-zero elements), respectively

   Write the generated matrices to a file.

The python inbuilt libraries of scipy and numpy were used

CODE:

```
import numpy as np
from scipy import linalg , sparse , stats
import sys

from contextlib import redirect_stdout

with open('out.txt', 'w') as f:
    with redirect_stdout(f):

        #Take the rows and coloumns of the matrix
        rows = int(input("Enter the number of rows in the matrix"))
        cols = int(input("Enter the number of cols in the matrix"))
```

```
        #check if they require identity matrix
        ans = int(input("Do you want indentity matrix?"))

        #If the user requires identity , check for the symmetric matrix
        if ans and rows==cols:
            arr=np.identity(rows)
            print(arr)
            sys.exit()

        #Take the mean and standard deviaion of the data
        mean = int(input ("Enter the mean of the data"))
        std = int(input("Enter the standard deviation of the data"))

        #make an array using the above mea and standard deviation
        arr = np.random.normal(mean,std,(rows,cols))

        #for the symmetric matrix
        if rows == cols:
            matrix_structure = int(input("""Enter the number you want your matrix
to be
                                            1.diagonal
                                            2.bidiagonal
                                            3.Block diagonal
                                            4.Symmetric
                                            5.Skew-symmetric
                                            6.Toepliz
                                            7.Circulant
                                            8.Sparz
                                            9.stochastic
                                            10.Doubly stochastic"""))

        #print(arr)

            #if the user wants diagonal matrix
            if matrix_structure == 1:
                final = np.diag(np.diag(arr))
                print(final)

            #if the user wants bidioginal matrix
            elif matrix_structure ==2:

                #get the diagonal and put it in the function to get a diagonal
matrix
                diag = np.diag(np.diag(arr))

                #get the upper diagonal and put it in the diagonal matrix
                upper_diag = np.diag(np.diag(arr,1),1)
```

```
        final = diag+upper_diag

        print(final)

    #if the user wants block diagonal matrix
    elif matrix_structure ==3:
        size = int(input("Enter the size of the blocks"))

        #creating the random diagonal for size*size matrix
        diagonal = []
        for i in range(rows):
            diagonal.append(np.random.normal(mean,std , (size,size)))

        #now create the block diagonal matrix using the above diagonal
        final = linalg.block_diag(*diagonal)

        print(final)

    #if the user wants symmetric matrix
    elif matrix_structure ==4:
        final = (arr + arr.T)/2

        print(final)

    #if the user wants skew-symmetric matrix
    elif matrix_structure == 5:
        final = arr - arr.T

        print(final)

    #if the user wants Toepliz matrix
    elif matrix_structure == 6:

        #get all the first coloumn
        col = arr[:][0]

        #get the second coloumn
        row = arr[0]

        final = linalg.toeplitz(col,row)

        print(final)

    #if the user wants Circuilant matrix
    elif matrix_structure ==7:
```

```
                #get the first row in the matrix
                row = arr[0]

                final = linalg.circulant(row)
                print(final)

            #if the user wants stochastic matrix
            elif matrix_structure == 9:

                #divide the rows by their mean
                for i in range(rows):
                    arr[i] = arr[i]/sum(arr[i])

                print(arr)

            #if the user wants stochastic matrix
            elif matrix_structure ==8:

                #get the sparsity factor
                sparsity_factor = int(input("Enter the sparsity factor"))

                #generate the random variable function
                rvs = stats.norm(mean,std).rvs

                final = sparse.random(rows,cols , density =
((sparsity_factor)/(rows*cols)) , data_rvs=rvs)

                print(final)

            #if the user wants doubly stochastic matrix
            elif matrix_structure == 10:

                #keep dividing with avg of row and coloumn sum till both are 1 for
all rows and coloumns
                while True:
                    arr = arr/np.sum(arr,0)
                    arr = arr/np.sum(arr,1).reshape(rows,1)

                    if np.all(np.sum(arr,1)==1) and np.all(np.sum(arr,0)==1):
                        break

                print(arr)

        #For the non symmetric matrix
        else:
            structure = int(input("""Enter the structure of the required matrix
                                    1.stochastic
```

```
                                    2.Toepliz
                                    3.sparz"""))

            #if the user wants stochastic matrix
            if structure == 1:
                for i in range(rows):
                    arr[i] = arr[i]/sum(arr[i])
                print(arr)

            #If the user wants toepliz matrix
            if structure == 2:

                #get all the first coloumn
                col = arr[:][0]

                #get the first row in the matrix
                row = arr[0]

                final = linalg.toeplitz(col,row)

                print(final)

            #if the user wants sparz matrix
            elif structure == 3:
                sparsity_factor = int(input("Enter the sparsity factor"))
                rvs = stats.norm(mean,std).rvs
                final = sparse.random(rows,cols , density =
((sparsity_factor)/(rows*cols)) , data_rvs=rvs)
                print(final)
```

INPUT:
Rows = columns = 3 (for all except sparz , stchochastic and Toeplitz)

Mean = 4 , standard deviation = 1

For others , rows = 3 , columns=4

OUTPUT:

1.Identity matrix

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

2.Diagonal

```
[[4.30887146 0.         0.        ]
 [0.         2.41253471 0.        ]
 [0.         0.         5.95074462]]
```

3.Bi - Diagonal

```
[[4.88231606 5.55899761 0.        ]
 [0.         2.37399688 3.78469605]
 [0.         0.         3.2404591 ]]
```

4.Block Diagonal

```
[[3.83745791 5.939736   3.58760158 0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]
 [4.42073018 3.54859413 4.95878076 0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]
 [4.50450496 4.4502976  3.81056662 0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         2.59639667 5.21978026 3.22021687
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         3.81682561 5.22058727 5.04379909
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         1.92727055 1.95941577 2.5574509
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  4.04147962 3.58355645 3.31726718 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  2.9092359  2.79768938 3.68084242 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  4.34984263 4.45572822 4.89239082 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         4.41746291 4.98990228 4.57285723]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         4.51966601 2.69220563 6.37161048]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         2.25484883 2.0376375  2.47022506]]
```

5.Symmetric

```
[[5.13933692 2.93697586 4.11471607]
 [2.93697586 4.39507903 4.16580497]
 [4.11471607 4.16580497 4.72978651]]
```

6.Skew-symmetric

```
[[ 0.          1.94003739 -2.31783802]
 [-1.94003739  0.         -0.63335045]
 [ 2.31783802  0.63335045  0.        ]]
```

7.Toeplitz

```
[[3.68041721 4.54870092 4.59542515 4.5130746 ]
 [4.54870092 3.68041721 4.54870092 4.59542515]
 [4.59542515 4.54870092 3.68041721 4.54870092]
 [4.5130746  4.59542515 4.54870092 3.68041721]]
```

8.Circulant

```
[[4.90136107 4.89346836 4.97882117]
 [4.97882117 4.90136107 4.89346836]
 [4.89346836 4.97882117 4.90136107]]
```

9.Sparz (sparisity factor was 3)

```
(1, 1)    5.154589620793383
(0, 1)    3.581207750459803
(0, 0)    3.6457775898975995
```

10.Stochastic

```
[[0.23908257 0.28892555 0.27201397 0.19997791]
 [0.21826932 0.21624729 0.29028508 0.2751983 ]
 [0.23950299 0.19551739 0.31584582 0.2491338 ]]
```

11.Doubly Stochastic

```
[[0.16576513 0.46558306 0.3686518 ]
 [0.50451429 0.21102142 0.28446429]
 [0.32972058 0.32339551 0.34688391]]
```

Question 2:
AIM:

2. Create a Python program that reads matrices from a file and does the following:

   (a) Matrix – addition, multiplication, Kronecker product, Hadamard product, pseudo-inverse

   (b) Computation of – determinant, rank, Eigen values, Eigen vectors

   (c) Computation of p-norm, where the value of $p > 1$ is given by the user

   Append the results to the file containing the matrices.

Note:
The arrays were stored in two different files and result was appended to the first file

CODE:

```
import numpy as np
import sys

#load the matrix from the file
a1 = np.loadtxt("C:/Study/SEM 4/Cad/assignment 4/array1.txt")
a2 = np.loadtxt("C:/Study/SEM 4/Cad/assignment 4/array2.txt")

sys.stdout = open("array1.txt" , "a")

#addition
print("Addition of a1 and a2\n",a1+a2)
print()

#Multiplication
print("Multiplication of the matrices\n",np.dot(a1,a2))
print()

#Kronecker
print("Kronecker product of the Matrices\n",np.kron(a1,a2))
print()

#Hadmard
print("Hadamard of the Matrices\n",a1*a2)
print()

#Pseudo inverse of a2 and a1
```

```
print("Pseudo inverse of the array 1\n",np.linalg.pinv(a1))
print()

print("Pseudo inverse of the array 1\n",np.linalg.pinv(a2))
print()

#determinant
print("Determinant of the
matrices\n","a1=",np.linalg.det(a1),"a2=",np.linalg.det(a2))
print()

#rank
print("Rank of the
matrices\n","a1=",np.linalg.matrix_rank(a1),"a2=",np.linalg.matrix_rank(a2)
)
print()

#eigen value of the vectors , the function returns eigen value and eigen
vector
e1_a,e_vec_a = np.linalg.eig(a1)
e1_b,e_vec_b = np.linalg.eig(a2)
print("Array of eigen values and eigen vectors of array 1\n" , e1_a ,"\n",
e_vec_a)
print("Array of eigen values and eigen vectors of array 2\n" , e1_b ,"\n",
e_vec_b)

#Take the pnorm of the matrices by taking the input of the p , use while
loop to get the correct input
while True:
    p1 = int(input("Enter the value of p"))
    if(p1>1):
        break
    else:
        print("Wrong input try again")

pnorm1 = np.sum(np.abs(a1**p1))**1/p1
print(pnorm1)

while True:
    p2 = int(input("Enter the value of p"))
    if(p2>1):
        break
    else:
```

```
        print("Wrong input try again")

pnorm2 = np.sum(np.abs(a1**p2))**1/p2
print(pnorm2)

sys.stdout.close()
```

INPUT:

File 1

```
1 2 3
3 4 5
5 7 9
```

File 2

```
4.0 5.0 6.0
1.0 1.0 1.0
2.0 3.0 4.0
```

Output:

```
1 2 3
3 4 5
5 7 9
Addition of a1 and a2
 [[ 5.  7.  9.]
 [ 4.  5.  6.]
 [ 7. 10. 13.]]

Multiplication of the matrices
 [[12. 16. 20.]
 [26. 34. 42.]
 [45. 59. 73.]]

Kronecker product of the Matrices
 [[ 4.  5.  6.  8. 10. 12. 12. 15. 18.]
 [ 1.  1.  1.  2.  2.  2.  3.  3.  3.]
 [ 2.  3.  4.  4.  6.  8.  6.  9. 12.]
 [12. 15. 18. 16. 20. 24. 20. 25. 30.]
 [ 3.  3.  3.  4.  4.  4.  5.  5.  5.]
 [ 6.  9. 12.  8. 12. 16. 10. 15. 20.]
 [20. 25. 30. 28. 35. 42. 36. 45. 54.]
 [ 5.  5.  5.  7.  7.  7.  9.  9.  9.]
 [10. 15. 20. 14. 21. 28. 18. 27. 36.]]

Hadamard of the Matrices
 [[ 4. 10. 18.]
 [ 3.  4.  5.]
 [10. 21. 36.]]

Pseudo inverse of the array 1
 [[-1.22619048  0.48809524  0.11904762]
 [-0.19047619  0.0952381   0.04761905]
 [ 0.8452381  -0.29761905 -0.02380952]]

Pseudo inverse of the array 1
 [[ 0.52777778  0.77777778 -1.02777778]
 [ 0.11111111  0.11111111 -0.11111111]
 [-0.30555556 -0.55555556  0.80555556]]

Determinant of the matrices
 a1= 1.48769885299771e-15 a2= 0.0

Rank of the matrices
 a1= 2 a2= 2

Array of eigen values and eigen vectors of array 1
 [ 1.44833148e+01 -4.83314774e-01  2.55012131e-16]
 [[ 0.25767102  0.89580026  0.40824829]
 [ 0.47503892 -0.41114679 -0.81649658]
 [ 0.84139389 -0.16882006  0.40824829]]
Array of eigen values and eigen vectors of array 2
 [8.53112887e+00 4.68871126e-01 9.76971494e-17]
 [[ 0.85100788  0.41439676  0.40824829]
 [ 0.17857442  0.56423841 -0.81649658]
 [ 0.49385905 -0.71408007  0.40824829]]
483.0
```

Question 3:
AIM:

3. Create a Python program that reads a matrix $\mathbf{A}$ and vector $\mathbf{b}$ from a file and computes:

   (a) $\mathbf{x}$ that solves $\mathbf{Ax} = \mathbf{b}$

   (b) if not solvable, then gives an error message

   (c) if there are many solutions, then gives the least squares solution

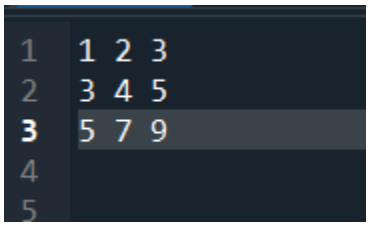
Code:

```
import numpy as np

#load the matrix from the file

arrays = []

with open("C:/Study/SEM 4/Cad/assignment 4/array2.txt") as file:
    array = []

    #iterate over each line
    for line in file:

        #if the line has no data or empty
        if line.strip() == "":

            #append the array to the list of arrays
            arrays.append(array)

            #empty array for the next array
            array = []

        else:

            #append the list to the array
            array.append(list(map(float , line.rstrip().split())))

    #append the final list to the list of arrays
    arrays.append(array)

file.close()
```

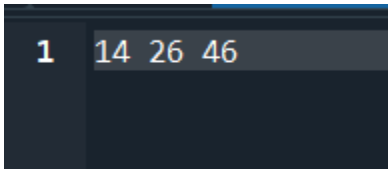
```
a1 = np.array(arrays[0])
v1 = np.array(arrays[1])[0].T

print(a1,v1)

#Find the solution for the given matrix equation
sol = np.linalg.lstsq(a1,v1,rcond=None)

#check if it has no solution
#The linalg.lstsq was not giving error for no solution for some reason so i
did like this
if(np.dot(a1,sol[0].T).round(decimals=5) == v1).all():
    print("The solution is : ", sol[0])
else:
    print("No solution")
```

Input:

A



B:

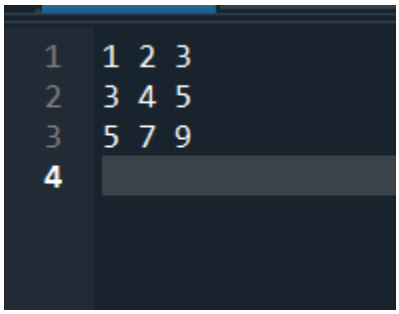


Final Ans:

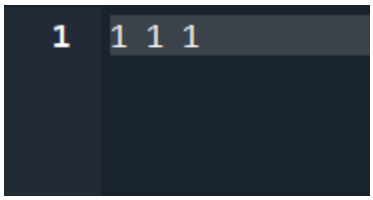
```
Console 1/A  Console 2/A

In [26]: runfile('C:/Study/SEM 4/Cad/assignment 4/q3_.py', wdir='C:/Study/SEM 4/Cad/assignment 4')
The solution is :  [1. 2. 3.]

In [27]:
```

NO solution variant

A:



B:



Output:

```
In [27]: runfile('C:/Study/SEM 4/Cad/assignment 4/q3_.py', wdir='C:/Study/SEM 4/Cad/assignment 4')
No solution

In [28]:
```