Name: Somit Jain

Reg. No. 20BDS0181

Course: Mathematical Modelling for Data Science(CSE3045 ELA)

Faculty: Dr. Ilanthenral K P S K

Slot: L49+L50

Lab-Assignment 1

1. Using Python NumPy package linalg, explore all the possible routines along with different possible scenarios with specific reference to

import numpy as np

A = np.array([[1,2],

[4,5]])

B = np.array([[6,2],

[7,0]])

C = np.array([[1,2,5],

[3,4,6]])

```
import numpy as np
A = np.array([[1,2],
              [4,5]])
B = np.array([[6,2],
            [7,0]])
C = np.array([[1,2,5],
              [3,4,6]])
```

a. **Matrix addition**

Code: #Addition of A and B

print(A+B)

print(A+C)#dimension should be same

```
#Addition of A and B
print(A+B)
print(A+C)#dimension should be same
```

```
[[ 7  4]
 [11  5]]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 #Addition of A and B
      2 print(A+B)
----> 3 print(A+C)

ValueError: operands could not be broadcast together with shapes (2,2) (2,3)
```

b. Matrix multiplication

Code: #Multiplication of A and B

print(np.matmul(A,B))

#When CxA can't be done because of dimension 2x3 and 2x2

print(np.matmul(C,A))

```
#Multiplication of A and B
print(np.matmul(A,B))
#When CxA can't be done because of dimension 2x3 and 2x2
print(np.matmul(C,A))
```

```
[[20  2]
 [59  8]]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [4], in <cell line: 4>()
      2 print(np.matmul(A,B))
      3 #When CxA can't be done because of dimension 2x3 and 2x2
----> 4 print(np.matmul(C,A))

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 3)
```

c. Determinant of a matrix

Code: #Determinant of matrix A

np.linalg.det(A)

```
#Determinant of matrix A
np.linalg.det(A)
```

```
-2.9999999999999996
```

d. Rank of a matrix

Code: #Rank of matrix A

np.linalg.matrix_rank(A)

```
#Rank of matrix A
np.linalg.matrix_rank(A)
```

```
2
```

e. Multiplicative inverse of a matrix

#Multiplicative inverse of A

np.linalg.inv(A)

```
#Multiplicative inverse of A
np.linalg.inv(A)
```

```
array([[-1.66666667,  0.66666667],
       [ 1.33333333, -0.33333333]])
```

Code: #Let D be a matrix of det = 0

D = np.matrix([[1,2],

[2,4]])

print(np.linalg.det(D))

print(np.linalg.inv(D))#Inverse can't be found for matrix with det 0

```
#Let D be a matrix of det = 0
D = np.matrix([[1,2],
               [2,4]])
print(np.linalg.det(D))
print(np.linalg.inv(D))#Inverse can't be found for matrix with det 0
```

```
0.0

---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Input In [5], in <cell line: 5>()
      2 D = np.matrix([[1,2],
      3               [2,4]])
      4 print(np.linalg.det(D))
----> 5 print(np.linalg.inv(D))

File <__array_function__ internals>:5, in inv(*args, **kwargs)

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:545, in inv(a)
    543 signature = 'D->D' if isComplexType(t) else 'd->d'
    544 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 545 ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    546 return wrap(ainv.astype(result_t, copy=False))

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:88, in _raise_linalgerror_singular(err, flag)
     87 def _raise_linalgerror_singular(err, flag):
---> 88     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

2. Without using inbuilt routines, solve a system of linear equations with two variables.

a. By substitution method

Code: """Let two equations be a1x+b1y=c1 and a2x+b2y=c2

We know that in substitution we substitute a variable in 1 equation using other equation. By doing this we get the following

solution x = (c1b2-b2c1)/(a1b2-a2b1) and y = x = (a1c2-a2c1)/(a1b2-a2b1) and if denominator is infinity the lines are

parallel"""

a1,b1,c1,a2,b2,c2 = map(int,input().split())

det = a1*b2 - b1*a2

if det==0:

print("Lines are parallel")

else:

x = (c1*b2 - b1*c2)/det

y = (a1*c2 - c1*a2)/det

print("Solution is x={} and y={}".format(x,y))

```
"""Let two equations be a1x+b1y=c1 and a2x+b2y=c2
We know that in substitution we substitute a variable in 1 equation using other equation. By doing this we get the following
solution x = (c1b2-b2c1)/(a1b2-a2b1) and y = x = (a1c2-a2c1)/(a1b2-a2b1) and if denominator is infinity the lines are
parallel"""
a1,b1,c1,a2,b2,c2 = map(int,input().split())
det = a1*b2 - b1*a2
if det==0:
    print("Lines are parallel")
else:
    x = (c1*b2 - b1*c2)/det
    y = (a1*c2 - c1*a2)/det
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 18 5 1 19
Solution is x=3.0 and y=4.0
```

No Solution:

```
"""Let two equations be a1x+b1y=c1 and a2x+b2y=c2
We know that in substitution we substitute a variable in 1 equation using other equation. By doing this we get the following
solution x = (c1b2-b2c1)/(a1b2-a2b1) and y = x = (a1c2-a2c1)/(a1b2-a2b1) and if denominator is infinity the lines are
parallel"""
a1,b1,c1,a2,b2,c2 = map(int,input().split())
det = a1*b2 - b1*a2
if det==0:
    print("Lines are parallel")
else:
    x = (c1*b2 - b1*c2)/det
    y = (a1*c2 - c1*a2)/det
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 14 4 6 10
Lines are parallel
```

Infinite Solution:

```
"""Let two equations be a1x+b1y=c1 and a2x+b2y=c2
We know that in substitution we substitute a variable in 1 equation using other equation. By doing this we get the following
solution x = (c1b2-b2c1)/(a1b2-a2b1) and y = x = (a1c2-a2c1)/(a1b2-a2b1) and if denominator is infinity the lines are
parallel"""
a1,b1,c1,a2,b2,c2 = map(int,input().split())
det = a1*b2 - b1*a2
if det==0:
    print("Lines are parallel")
else:
    x = (c1*b2 - b1*c2)/det
    y = (a1*c2 - c1*a2)/det
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 4 4 6 8
Lines are parallel
```

b. By Elimination method

Code: #Let two equations be a1x+b1y=c1 and a2x+b2y=c2

```
a1,b1,c1,a2,b2,c2 = map(int,input().split())

b1 = b1*a2

c1 = c1*a2

b2 = b2*a1

c2 = c2*a1

a1 = a1*a2

a2 = a1

if b1-b2==0:

   print("The lines are parallel")

else:

   y = (c1-c2)/(b1-b2)

   x = (c1 - b1*y)/(a1)

   print("Solution is x={} and y={}".format(x,y))
```

```
#Let two equations be a1x+b1y=c1 and a2x+b2y=c2
a1,b1,c1,a2,b2,c2 = map(int,input().split())
b1 = b1*a2
c1 = c1*a2
b2 = b2*a1
c2 = c2*a1
a1 = a1*a2
a2 = a1
if b1-b2==0:
    print("The lines are parallel")
else:
    y = (c1-c2)/(b1-b2)
    x = (c1 - b1*y)/(a1)
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 18 5 1 19
Solution is x=3.0 and y=4.0
```

No Solution:

```
#Let two equations be a1x+b1y=c1 and a2x+b2y=c2
a1,b1,c1,a2,b2,c2 = map(int,input().split())
b1 = b1*a2
c1 = c1*a2
b2 = b2*a1
c2 = c2*a1
a1 = a1*a2
a2 = a1
if b1-b2==0:
    print("The lines are parallel")
else:
    y = (c1-c2)/(b1-b2)
    x = (c1 - b1*y)/(a1)
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 14 4 6 10
The lines are parallel
```

Infinite Solution:

```
#Let two equations be a1x+b1y=c1 and a2x+b2y=c2
a1,b1,c1,a2,b2,c2 = map(int,input().split())
b1 = b1*a2
c1 = c1*a2
b2 = b2*a1
c2 = c2*a1
a1 = a1*a2
a2 = a1
if b1-b2==0:
    print("The lines are parallel")
else:
    y = (c1-c2)/(b1-b2)
    x = (c1 - b1*y)/(a1)
    print("Solution is x={} and y={}".format(x,y))
```

```
2 3 4 4 6 8
The lines are parallel
```

c. Gaussian Elimination and Gauss- Jordan Method

Code: #Gauss Elimination Method: Every 1st element of the row should be the only non zero element in its column

import numpy as np

a = np.zeros((2,3))

x = np.zeros(2)

for i in range(2):#taking input of augmented matrix

for j in range(3):

a[i][j] = float(input())

print(a)

rat = a[1][0]/a[0][0]

for k in range(3):

a[1][k]-=(rat*a[0][k])

print(a)

x[1] = a[1][2]/a[1][1]#finding the last variable because in matrix that can be caluclated only

x[0] = (a[0][2]-x[1]*a[0][1])/a[0][0]

print('\nSolution : %f and %f' %(x[0],x[1]))

```
#Gauss Elimination Method: Every 1st element of the row should be the only non zero element in its column
import numpy as np
a = np.zeros((2,3))
x = np.zeros(2)
for i in range(2):#taking input of augmented matrix
    for j in range(3):
        a[i][j] = float(input())
print(a)
rat = a[1][0]/a[0][0]
for k in range(3):
    a[1][k]-=(rat*a[0][k])
print(a)
x[1] = a[1][2]/a[1][1]#finding the last variable because in matrix that can be caluclated only
x[0] = (a[0][2]-x[1]*a[0][1])/a[0][0]
print('\nSolution : %f and %f' %(x[0],x[1]))
```

```
2
3
18
5
1
19
[[ 2.  3. 18.]
 [ 5.  1. 19.]]
[[  2.    3.   18. ]
 [  0.   -6.5 -26. ]]

Solution : 3.000000 and 4.000000
```

Code: #Gauss Jordan Elimination: There can be only one non zero element in each row and column

import numpy as np

a = np.zeros((2,3))

x = np.zeros(2)

for i in range(2):#taking input of augmented matrix

for j in range(3):

a[i][j] = float(input())

print(a)

rat1 = a[1][0]/a[0][0]

for k in range(3):

a[1][k]-=(rat1*a[0][k])

rat2 = a[0][1]/a[1][1]

for k in range(3):

a[0][k]-=(rat2*a[1][k])

print(a)

x[1] = a[1][2]/a[1][1]

x[0] = a[0][2]/a[0][0]

print('\nSolution : %f and %f' %(x[0],x[1]))

```
#Gauss Jordan Elimination: There can be only one non zero element in each row and column
import numpy as np
a = np.zeros((2,3))
x = np.zeros(2)
for i in range(2):#taking input of augmented matrix
    for j in range(3):
        a[i][j] = float(input())
print(a)
rat1 = a[1][0]/a[0][0]
for k in range(3):
    a[1][k]-=(rat1*a[0][k])
rat2 = a[0][1]/a[1][1]
for k in range(3):
    a[0][k]-=(rat2*a[1][k])
print(a)
x[1] = a[1][2]/a[1][1]
x[0] = a[0][2]/a[0][0]
print('\nSolution : %f and %f' %(x[0],x[1]))
```

```
2
3
18
5
1
19
[[ 2.  3. 18.]
 [ 5.  1. 19.]]
[[  2.    0.    6. ]
 [  0.   -6.5 -26. ]]

Solution : 3.000000 and 4.000000
```

Code: import numpy as np

a = np.ones((2,2))

b = np.zeros((2,1))

x = np.zeros(2)

for i in range(2):#taking input of augmented matrix

  a[i][0]= float(input())

  a[i][1]= float(input())

  b[i][0]= float(input())

print(a)

print(b)

np.linalg.solve(a,b)

```
import numpy as np
a = np.ones((2,2))
b = np.zeros((2,1))
x = np.zeros(2)
for i in range(2):#taking input of augmented matrix
    a[i][0]= float(input())
    a[i][1]= float(input())
    b[i][0]= float(input())
print(a)
print(b)
np.linalg.solve(a,b)
```

```
2
3
18
5
1
19
[[2. 3.]
 [5. 1.]]
[[18.]
 [19.]]

array([[3.],
       [4.]])
```

No Solution:

```
import numpy as np
a = np.ones((2,2))
b = np.zeros((2,1))
x = np.zeros(2)
for i in range(2):#taking input of augmented matrix
    a[i][0]= float(input())
    a[i][1]= float(input())
    b[i][0]= float(input())
print(a)
print(b)
np.linalg.solve(a,b)
```

```
2
3
14
4
6
10
[[2. 3.]
 [4. 6.]]
[[14.]
 [10.]]

---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Input In [12], in <cell line: 11>()
      9 print(a)
     10 print(b)
---> 11 np.linalg.solve(a,b)

File <__array_function__ internals>:5, in solve(*args, **kwargs)

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:393, in solve(a, b)
    391 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    392 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 393 r = gufunc(a, b, signature=signature, extobj=extobj)
    395 return wrap(r.astype(result_t, copy=False))

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:88, in _raise_linalgerror_singular(err, flag)
     87 def _raise_linalgerror_singular(err, flag):
---> 88     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

Infinite Solution:

```
import numpy as np
a = np.ones((2,2))
b = np.zeros((2,1))
x = np.zeros(2)
for i in range(2):#taking input of augmented matrix
    a[i][0]= float(input())
    a[i][1]= float(input())
    b[i][0]= float(input())
print(a)
print(b)
np.linalg.solve(a,b)
```

```
2
3
4
4
6
8
[[2. 3.]
 [4. 6.]]
[[4.]
 [8.]]

---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Input In [13], in <cell line: 11>()
      9 print(a)
     10 print(b)
---> 11 np.linalg.solve(a,b)

File <__array_function__ internals>:5, in solve(*args, **kwargs)

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:393, in solve(a, b)
    391 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    392 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 393 r = gufunc(a, b, signature=signature, extobj=extobj)
    395 return wrap(r.astype(result_t, copy=False))

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:88, in _raise_linalgerror_singular(err, flag)
     87 def _raise_linalgerror_singular(err, flag):
---> 88     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

Lab 2: Eigen Values and Eigen Vectors

2. Find Eigen values and eigen vectors using Python NumPy package linalg, explore all the possible routines along with different possible scenarios.

Code: import numpy as np

import numpy.linalg as alg

A = np.matrix([[2,-3,0],

[2,-5,0],

[0,0,3]])

evalu,evect = alg.eig(A)

evalu = np.round_(evalu)

evect = np.round_(evect)

print("Eigen Values:")

print(evalu)

print("Eigen Vectors:")

print(evect)

```
import numpy as np
import numpy.linalg as alg
A = np.matrix([[2,-3,0],
               [2,-5,0],
               [0,0,3]])
evalu,evect = alg.eig(A)
evalu = np.round_(evalu)
evect = np.round_(evect)
print("Eigen Values:")
print(evalu)
print("Eigen Vectors:")
print(evect)
```

```
Eigen Values:
[ 1. -4.  3.]
Eigen Vectors:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Not a Square Matrix:

```
import numpy as np
import numpy.linalg as alg
A = np.matrix([[1,2,3,4],
               [4,5,6,5],
               [7,8,9,6]])
evalu,evect = alg.eig(A)
evalu = np.round_(evalu)
evect = np.round_(evect)
print("Eigen Values:")
print(evalu)
print("Eigen Vectors:")
print(evect)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Input In [15], in <cell line: 6>()
      2 import numpy.linalg as alg
      3 A = np.matrix([[1,2,3,4],
      4                [4,5,6,5],
      5                [7,8,9,6]])
----> 6 evalu,evect = alg.eig(A)
      7 evalu = np.round_(evalu)
      8 evect = np.round_(evect)

File <__array_function__ internals>:5, in eig(*args, **kwargs)

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:1316, in eig(a)
   1314 a, wrap = _makearray(a)
   1315 _assert_stacked_2d(a)
-> 1316 _assert_stacked_square(a)
   1317 _assert_finite(a)
   1318 t, result_t = _commonType(a)

File D:\Anaconda\lib\site-packages\numpy\linalg\linalg.py:203, in _assert_stacked_square(*arrays)
    201 m, n = a.shape[-2:]
    202 if m != n:
--> 203     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

4. Using R, find the eigen values and vectors.

Code: A = matrix(c(2,-3,0,2,-5,0,0,0,3),3,3,byrow = TRUE)

print(A)

print(eigen(A))