Matrices_EigenValuesVectorsUsingNumpy

December 25, 2021

[1]:
```
import numpy as np
```

[2]:
```
np.matrix([5,6,8])
```

[2]:
```
matrix([[5, 6, 8]])
```

[3]:
```
a=np.matrix([5,6,7])
print(a)
```

```
[[5 6 7]]
```

[4]:
```
np.matrix([[1],[2],[3],[4]])
```

[4]:
```
matrix([[1],
        [2],
        [3],
        [4]])
```

[5]:
```
#To find dimension of a matrix
b=np.matrix([[1],[2],[3],[4]])
b.shape
```

[5]:
```
(4, 1)
```

[6]:
```
#generate a identity matrix of order 7
np.eye(7)
```

[6]:
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.]])
```

[7]:
```
np.zeros(24).reshape(4,6) #zero matrix of size 4x6
```

[7]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
```

```
        [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.]])
```

[8]:
```
#Write a matrix of size 4x6 with all entries equal to 1
np.ones(24).reshape(4,6)
```

[8]:
```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

[9]:
```
#Write a matrix of size 4x6 with all entries equal to 2.5 i.e. constant matrix␣
 ↪of size 4x6 with entries 2.5
c=np.ones(24).reshape(4,6)
c=2.5*c
print(c)
```

```
[[2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]]
```

[10]:
```
#Generate a diagonal matrix with diagonal entries 1,2,6
np.diag([1,2,6])
```

[10]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 6]])
```

Operations on Matrices

[11]:
```
A=np.matrix([[1,2],[3,4]])
B=np.matrix([[5,6],[7,8]])
print(A)
print(B)
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
```

[12]:
```
A+B
```

[12]:
```
matrix([[ 6,  8],
        [10, 12]])
```

[13]:
```
A-B
```

[13]: matrix([[-4, -4],
        [-4, -4]])

[14]:
```
A*B
```

[14]: matrix([[19, 22],
        [43, 50]])

[15]:
```
B**2+A**3  #B^2+A^3
```

[15]: matrix([[104, 132],
        [172, 224]])

[16]:
```
#Observe that A*B = A@B
A@B
```

[16]: matrix([[19, 22],
        [43, 50]])

[17]:
```
#Find A^2 + 6B - 7I
A*A +6*B-7*np.eye(2)
```

[17]: matrix([[30., 46.],
        [57., 63.]])

Linear Algebra using Python

[18]:
```
import numpy.linalg as la
```

[19]:
```
#Find inverse of B
la.inv(B)
```

[19]: matrix([[-4. , 3.],
        [3.5, -2.5]])

[20]:
```
la.det(A)
```

[20]: -2.0000000000000004

[21]:
```
#Find transpose of A
np.transpose(A)
```

[21]: matrix([[1, 3],
        [2, 4]])

[22]:
```
#Find transpose of A
A.T
```

[22]:
```
matrix([[1, 3],
        [2, 4]])
```

Eigenvalues and Eigenvectors

[23]:
```
A=np.matrix([[1,2],[5,6]])
```

[24]:
```
#Find eigenvalues of A
la.eig(A)
```

[24]:
```
(array([-0.53112887,  7.53112887]),
 matrix([[-0.79402877, -0.2928046 ],
         [ 0.60788018, -0.9561723 ]]))
```

[25]:
```
u,v=la.eig(A)
print(u)
print(v)
#u stores eigenvalues and v stores eigenvecors
```

```
[-0.53112887  7.53112887]
[[-0.79402877 -0.2928046 ]
 [ 0.60788018 -0.9561723 ]]
```

[26]:
```
B=np.matrix([[2,27,0],[0,4,40],[0,3,30]])
B
```

[26]:
```
matrix([[ 2, 27,  0],
        [ 0,  4, 40],
        [ 0,  3, 30]])
```

[27]:
```
#Find eigenvalues of B
u,v=la.eig(B)
print(u)
```

```
[ 2.  0. 34.]
```

[28]:
```
#Find rank of B
la.matrix_rank(B)
```

[28]:
```
2
```

[29]:
```
#Find trace of B
np.trace(B)
```

[29]:
```
36
```

Solving system of linear equations

[30]:
```
C=np.matrix([[1,1,1],[0,1,1],[0,0,1]])
print(C)
```

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

[31]:
```
D=np.matrix([[3],[-1],[2]])
D
```

[31]:
```
matrix([[ 3],
        [-1],
        [ 2]])
```

[32]:
```
la.solve(C,D)
```

[32]:
```
matrix([[ 4.],
        [-3.],
        [ 2.]])
```

[33]:
```
la.matrix_rank(C)
```

[33]:
```
3
```

[34]:
```
#Add D as a column to C (Augmented matrix [C,D])
#Last argument 1 adds a column and last argument 0 adds a row
E=np.insert(C,1,np.matrix((3,-1,2)),1)
E
```

[34]:
```
matrix([[ 1,  3,  1,  1],
        [ 0, -1,  1,  1],
        [ 0,  2,  0,  1]])
```

[35]:
```
la.matrix_rank(E)
```

[35]:
```
3
```

[36]:
```
if la.matrix_rank(C)==la.matrix_rank(E):
    print("CX=D is diagonalizable")
else:
    print("CX=D is not diagonalizable")
```

```
CX=D is diagonalizable
```

[35]:
```
# Matrix and System of linear equation #
```

[37]:
```
#Enter matrix A
A=np.matrix([[1,2,4],[1,5,2],[1,1,0]])
A
```

[37]:
```
matrix([[1, 2, 4],
        [1, 5, 2],
```

```
        [1, 1, 0]])
```

[38]:
```
#Find determinant of A
la.det(A)
```

[38]:
```
-14.000000000000004
```

[39]:
```
B=np.matrix([[1],[0],[0]])
B
```

[39]:
```
matrix([[1],
        [0],
        [0]])
```

[40]:
```
#Solve AX=B using inverse of A
soln=la.inv(A)@B
soln
```

[40]:
```
matrix([[ 0.14285714],
        [-0.14285714],
        [ 0.28571429]])
```

[41]:
```
#Solve AX+B using solve function
la.solve(A,B)
```

[41]:
```
matrix([[ 0.14285714],
        [-0.14285714],
        [ 0.28571429]])
```

[ ]:
```

```

[42]:
```
# x+y+z=3, x-y+z=1, x-y-z=-1#
A=np.matrix([[1,1,1],[1,-1,1],[1,-1,-1]])
B=np.matrix([[3],[1],[-1]])
la.solve(A,B)
```

[42]:
```
matrix([[1.],
        [1.],
        [1.]])
```

[43]:
```
# x+y=3, x-y=1 #
A=np.matrix([[1,1],[1,-1]])
B=np.matrix([[3],[1]])
la.solve(A,B)
```

[43]:
```
matrix([[2.],
        [1.]])
```

```
[ ]:
```

```
[ ]:
```