Linear Algebra

Youtube Channel ---> Ahmad Bazzi (NumPy Linear Algebra | Python # 3)

In [182]:
```
import numpy as np
import statistics as st
import seaborn as sns
import matplotlib.pyplot as plt
```

In [14]:
```
a = np.array([1,2,3,4,5])
a
```

Out[14]:
```
array([1, 2, 3, 4, 5])
```

In [16]:
```
type(a)
```

Out[16]:
```
numpy.ndarray
```

Indexing Elements in my numpy array

In [18]:
```
a[4]
```

Out[18]:
```
5
```

In [22]:
```
a[:5]
```

Out[22]:
```
array([1, 2, 3, 4, 5])
```

Finding the dimensions of an array

In [28]:
```
#It is a one dimensional array
a.shape
```

Out[28]:
```
(5,)
```

Creating a matrix

In [86]:
```
b = np.array([[1,2,3,4],[10,20,30,40],[100,200,300,400],[1000,2000,3000,4000]])
b
```

Out[86]:
```
array([[   1,    2,    3,    4],
       [  10,   20,   30,   40],
       [ 100,  200,  300,  400],
       [1000, 2000, 3000, 4000]])
```

In [87]:
```
#type(b)
b.dtype
```

Out[87]:
```
dtype('int32')
```

In [88]:
```
b.shape
```

Out[88]:
```
(4, 4)
```

In [89]:
```
# Number of Rows
b.shape[0]
```

Out[89]:
```
4
```

In [90]:
```
# Number of Column
b.shape[1]
```

Out[90]:
```
4
```

In [91]:
```
#First argument is Row , second argument is column
b[2,3]
```

Out[91]:
```
400
```

Extracting a submatrix

In [92]:
```
B = b[0:4,1:3]
B
```

Out[92]:
```
array([[   2,    3],
       [  20,   30],
       [ 200,  300],
       [2000, 3000]])
```

In [93]:
```
c = b[0:4,1:4]
c
```

Out[93]:
```
array([[   2,    3,    4],
       [  20,   30,   40],
       [ 200,  300,  400],
       [2000, 3000, 4000]])
```

Modify elements in an array or Matrix

In [94]:
```
b[2,1] = 1/3
b
```

Out[94]:
```
array([[   1,    2,    3,    4],
       [  10,   20,   30,   40],
       [ 100,    0,  300,  400],
       [1000, 2000, 3000, 4000]])
```

Creating Special Matrices

Identity Matrix

In [95]:
```
I = np.eye(5)
I
```

Out[95]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Zero Matrix

In [96]:
```
np.zeros(5)
```

Out[96]:
```
array([0., 0., 0., 0., 0.])
```

In [97]:
```
np.zeros((5,5))
```

Out[97]:
```
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., 0.]])
```

Ones Matrix

In [98]:
```
np.ones((6,4))
```

Out[98]:
```
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.]])
```

Constant Matrix

In [99]:
```
#np.full((shape) , value)
C = np.full((4,4),5)
C
```

Out[99]:
```
array([[5, 5, 5, 5],
       [5, 5, 5, 5],
       [5, 5, 5, 5],
       [5, 5, 5, 5]])
```

Random Matrix

In [100]:
```
X = np.random.random((4,4))
X
```

Out[100]:
```
array([[0.80206702, 0.42979754, 0.8296099 , 0.22468783],
       [0.39333832, 0.89256044, 0.45368663, 0.34153495],
       [0.66274338, 0.24003215, 0.54964449, 0.13668616],
       [0.46310365, 0.40912784, 0.07158216, 0.05287107]])
```

In [101]:
```
Y = np.random.random((1000,4))
np.mean(Y)
```

Out[101]:
```
0.4953514129779248
```

Standard Devition

In [102]:
```
np.std(Y)
```

Out[102]:
```
0.28818301244417505
```

In [105]:
```
b
```

Out[105]:
```
array([[   1,    2,    3,    4],
       [  10,   20,   30,   40],
       [ 100,    0,  300,  400],
       [1000, 2000, 3000, 4000]])
```

Checking DataType

In [106]:
```
b.dtype
```

Out[106]:
```
dtype('int32')
```

Matrix Operations

Matrix Addition

In [108]:
```
A = np.array([[1,3,5],[2,6,8],[7,-5,9]])
B = np.array([[5,7,9],[3,7,9],[2,-8,5]])
```

In [109]:
```
A
```

Out[109]:
```
array([[ 1,  3,  5],
       [ 2,  6,  8],
       [ 7, -5,  9]])
```

In [110]:
```
B
```

Out[110]:
```
array([[ 5,  7,  9],
       [ 3,  7,  9],
       [ 2, -8,  5]])
```

In [111]:
```
A+B
```

Out[111]:
```
array([[  6,  10,  14],
       [  5,  13,  17],
       [  9, -13,  14]])
```

In [112]:
```
C = A+B
C
```

Out[112]:
```
array([[  6,  10,  14],
       [  5,  13,  17],
       [  9, -13,  14]])
```

In [119]:
```
np.add(A,B , dtype = np.float32)
```

Out[119]:
```
array([[  6.,  10.,  14.],
       [  5.,  13.,  17.],
       [  9., -13.,  14.]], dtype=float32)
```

Matrix Subtraction

In [120]:
```
A-B
```

Out[120]:
```
array([[-4, -4, -4],
       [-1, -1, -1],
       [ 5,  3,  4]])
```

In [122]:
```
np.subtract(A,B , dtype = np.float64)
```

Out[122]:
```
array([[-4., -4., -4.],
       [-1., -1., -1.],
       [ 5.,  3.,  4.]])
```

MATRIX MULTIPLICATION (POINTWISE MULTIPLICATION)

In [123]:
```
A
```

Out[123]:
```
array([[ 1,  3,  5],
       [ 2,  6,  8],
       [ 7, -5,  9]])
```

In [124]:
```
B
```

Out[124]:
```
array([[ 5,  7,  9],
       [ 3,  7,  9],
       [ 2, -8,  5]])
```

In [125]:
```
A*B
```

Out[125]:
```
array([[ 5, 21, 45],
       [ 6, 42, 72],
       [14, 40, 45]])
```

In [127]:
```
np.multiply(A,B,dtype = np.float64)
```

Out[127]:
```
array([[ 5., 21., 45.],
       [ 6., 42., 72.],
       [14., 40., 45.]])
```

Matrix Devision (Pointwise)

In [128]:
```
A
```

Out[128]:
```
array([[ 1,  3,  5],
       [ 2,  6,  8],
       [ 7, -5,  9]])
```

In [129]:
```
B
```

Out[129]:
```
array([[ 5,  7,  9],
       [ 3,  7,  9],
       [ 2, -8,  5]])
```

In [130]:
```
C = A/B
```

In [131]:
```
C
```

Out[131]:
```
array([[0.2       , 0.42857143, 0.55555556],
       [0.66666667, 0.85714286, 0.88888889],
       [3.5       , 0.625     , 1.8       ]])
```

In [135]:
```
## We cannot convert dtype from float into int through divide method for that we use another metho
C = np.divide(A,B , dtype = np.float32)
```

In [137]:
```
np.int32(C)
```

Out[137]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [3, 0, 1]])
```

Matrix Product

It is (first row * first column) type multiplication

In [139]:
```
A
```

Out[139]:
```
array([[ 1,  3,  5],
       [ 2,  6,  8],
       [ 7, -5,  9]])
```

In [140]:
```
B
```

Out[140]:
```
array([[ 5,  7,  9],
       [ 3,  7,  9],
       [ 2, -8,  5]])
```

In [138]:
```
np.matmul(A,B)
```

Out[138]:
```
array([[ 24, -12,  61],
       [ 44,  -8, 112],
       [ 38, -58,  63]])
```

In [147]:
```
np.transpose(A)
```

Out[147]:
```
array(<built-in method transpose of numpy.ndarray object at 0x00000257906EFC90>,
      dtype=object)
```

In [148]:
```
np.transpose(B)
```

Out[148]:
```
array([[ 5,  3,  2],
       [ 7,  7, -8],
       [ 9,  9,  5]])
```

Statistics

In [149]:
```
x = np.array([1,2,3,4,5])
x
```

Out[149]:
```
array([1, 2, 3, 4, 5])
```

In [150]:
```
np.mean(x)
```

Out[150]:
```
3.0
```

In [151]:
```
np.median(x)
```

Out[151]:
```
3.0
```

In [156]:
```
y = np.array([1,2,3,3,3,3,4,4,4,55,5])
y
```

Out[156]:
```
array([ 1,  2,  3,  3,  3,  3,  4,  4,  4, 55,  5])
```

In [157]:
```
st.mode(y)
```

Out[157]:
```
3
```

Quantiles

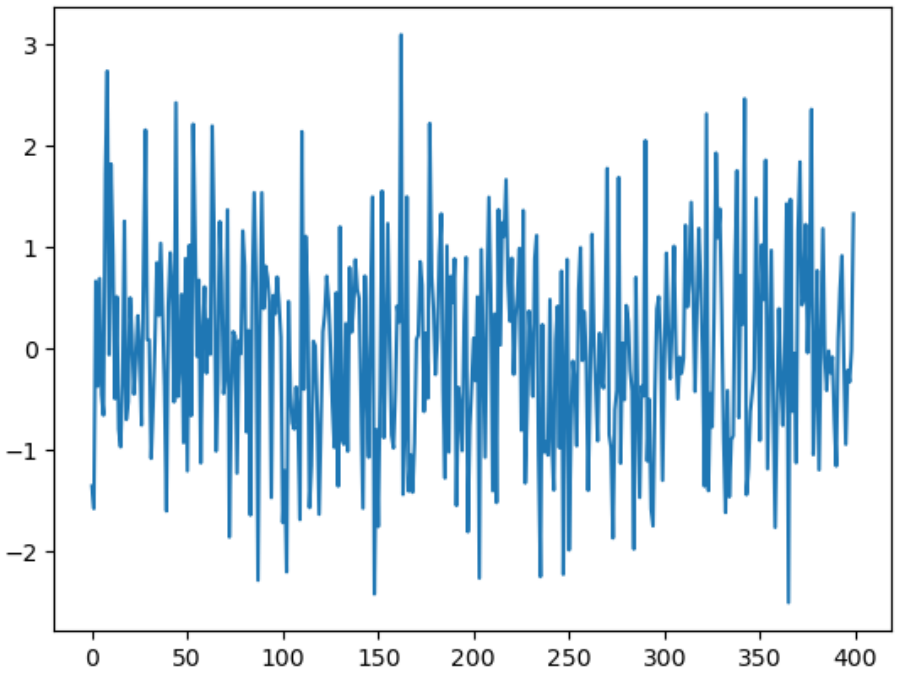
In [158]:
```
arr = [20,36,10,6,9]
```

In [172]:
```
print("arr :" , arr)
print("Q1 quantile of array : " ,  np.quantile(arr , .50))
print("Q2 quantile of array : " ,  np.quantile(arr , .25))
print("Q3 quantile of array : " ,  np.quantile(arr , .75))
print("100 quantile of array : " ,  np.quantile(arr , .1))
```

```
arr : [20, 36, 10, 6, 9]
Q1 quantile of array :  10.0
Q2 quantile of array :  9.0
Q3 quantile of array :  20.0
100 quantile of array :  7.2
```

In [181]:
```
import matplotlib.pyplot as plt
arr = np.random.normal(loc=0,scale=1,size=(400,))
plt.plot(arr)
```

Out[181]:
```
[<matplotlib.lines.Line2D at 0x2579591f3d0>]
```



In [ ]: