2. Introduction to PYTHON

Simple Initialization

```
A = 5
B = 2
C = A + B
C
Out [1] : 7
```

Numpy library

numpy is a PYTHON library for working with arrays and matrices.

```
# initialize array of ones
import numpy as np
A = np.ones([1,10])
```

1×10 array of 1's

```
A
Out [2] : array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

Numpy ones

```
# initialize matrix of ones
A = np.ones([5,5])
```

5×5 array of 1's

```
A
Out [3] : 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., 1.]])
```

Data type: Array of float 64 1.

Numpy Zeros

```
# initialize array of zeros
np.zeros([2,2])
```

2×2 array of 0's

```
out[4]: array([[0., 0.],
               [0., 0.]])
```

Array Creation

```
# initialize general array (1D array)
np.array([1, 2, 3, 4])
out[5]: array([1, 2, 3, 4])

# initialize 2D array
np.array([[1,2], [3, 4]])
out[6]: array([[1, 2],
               [3, 4]])
```

INT Datatype

Numpy arange

⊙ Generate uniformly spaced array.

```
A = np.arange(1, 11)
A
out[7]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

1×10 array
Default step size = 1

⊙ arange can also be used to specify step-size.

```
A = np.arange(1, 10, 2)
A
Out[31]: array([1, 3, 5, 7, 9])
```

step-size

Array addition

Addition of arrays

```
A = np.arange(1, 11)   <-- 1x10 array
B = np.ones([1,10])    <-- 1x10 array
A + 2 * B
Out[33]: array([[ 3., 4., 5., 6., 7., 8., 9., 10., 11., 12.]])
```

Array multiplication

```
A = np.arange(1, 11)   <-- 1x10 array
B = 2 * np.ones([1,10])  <-- 1x10 array
A * B
Out[35]: array([[ 2., 4., 6., 8., 10., 12., 14., 16., 18., 20.]])
```

* Directly performs element-wise multiplication!
  No need to use .*

Array division

```
A = np.arange(1, 11)
B = 2 * np.ones([1,10])
A / B
Out[36]: array([[0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5.]])
```

Exponentiation

```
A = np.arange(1, 11)
A ** 2
```

A^2

```
Out[37]: array([  1,  4,  9, 16, 25, 36, 49, 64, 81, 100], dtype=int32)
```

Logical operators

```
A = np.arange(1, 11)
A > 5
array([False, False, False, False, False, True, True, True, True, True])
A != 5
array([True, True, True, True, False, True, True, True, True, True])
```

General commands (Self explanatory)

np.abs()	np.log10()	np.real()
np.sqrt()	np.max	np.imag()
np.log2()		np.sum()

Numpy Random Library.

'Random' is a module in numpy for random number generation.

```
import numpy.random as nr
# Generate samples of Gaussian RV
# Mean = 0, variance = 1
nr.normal()
Out[54]: -0.2407689372...
```

Gaussian RV

⊙ Generate block of samples of Gaussian RV, with arbitrary mean and variance.

```
# nr.normal(mean, std_dev = sigma, blocklength)
nr.normal(2, np.sqrt(5), 10)
array([2.87..., -2.06..., 1.28..., 0.39..., -1.166...,
       5.93..., 4.16..., 1.42..., 1.16..., -0.58...])
```

⊙ Generate matrix of samples of Gaussian RV

```
# nr.normal (mean, sigma, [m,n])
nr.normal (2, np.sqrt (5), [2,2])   <---- 2x2 Matrix
Out [83] :
  array ([[ -0.41... , -1.19.... ],
          [4.69.... , 1.99 ]])
```

Generate 10 iid samples white Gaussian noise of power 3 dB.

```
                  Mean = 0            Variance = 2
nr.normal (0, np.sqrt (2), [1, 10])
```

Random integers

⊙ To generate random integers, use randint in Numpy Random.

```
# Generate random integers in [0, a)    <- 0 included, a not-included
# nr.randint (a).
nr.randint (2)
Out [111] : 1
```

nr.randint (5) — Gives random number b/w 0 and 4.

⊙ To generate array, and matrix of random integers, use syntaxes below.

```
nr.randint (2, size = 2)    1D array of 2 elements
Out [117] :
      array ([1, 0])

nr.randint (2, size = [2,2])    2D array of random integers
Out [118] :
      array ([[0, 1],
              [0, 1]])
```

BPSK symbols

To generate array of BPSK ±1 symbols, use syntax below.

```
2 * nr.randint(2, size=[2,2]) - 1
Out[41]:
    array([[-1, -1],
           [ 1, -1]])
```

0 → 1

1 → -1

Generate 2×2 array of QPSK symbols of power = 3dB

$$\left(2 * nr.randint(2, size=[2,2]) - 1\right) + 1j\left(2 * nr.randint(2, size=[2,2]) - 1\right)$$

$10 \log_{10} SNR = SNR\_dB$

$10 \log_{10} SNR = 3$

$\log_{10} SNR = 0.3$

$SNR = 10^{0.3}$

$\boxed{SNR \approx 2}$

Linear Algebra

numpy.linalg module has the functions required for operations in linear algebra.

```
import numpy.linalg as nl
A = np.array([[1,2], [3,4]])
nl.inv(A)
Out[208]:
    array([[-2. ,  1. ],
           [ 1.5, -0.5]])
```

Matrix Multiplication

For matrix multiplication, there are 2 options:

(i) For two matrices, use matmul in Numpy

(ii) For more than two matrices, use multi_dot in Numpy linalg.

```
A = np.array([[1,2],[3,4]])
B = np.array([[1,2],[3,4]])
np.matmul(A,B)
Out[97]:
        array([[7, 10],
               [15, 22]])
```

Option 1

```
nl.multi_dot([A,B,A])
Out[98]:
        array([[37, 54],
               [81, 118]])
```

Option 2

Note: A * B → Element wise multiplication of matrices.

Pseudo-inverse

For pseudo-inverse, use pinv in Numpy linalg.

Recall $pinv(A) = (A^H A)^{-1} A^H$

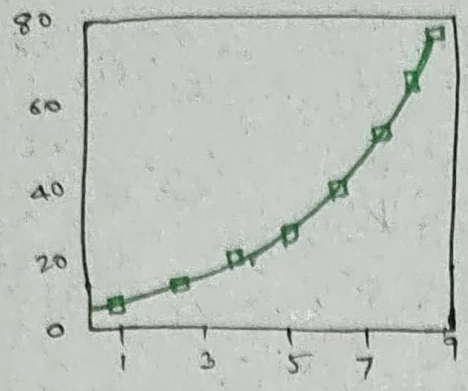
```
A = np.array([[1,2],[3,4],[5,6]])
nl.pinv(A)
Out[101]:
        array([[-1.33..., -0.33..., 0.66...],
               [1.08...,  0.33...,  -0.41...]])
```

Matplotlib

pyplot submodule in Matplotlib can be used for plotting.

```
import matplotlib.pyplot as plt
A = np.arange(1,10)        <- uniformly spaced array of 1 to 9
plt.plot(A, A**2, 'g-s')
```

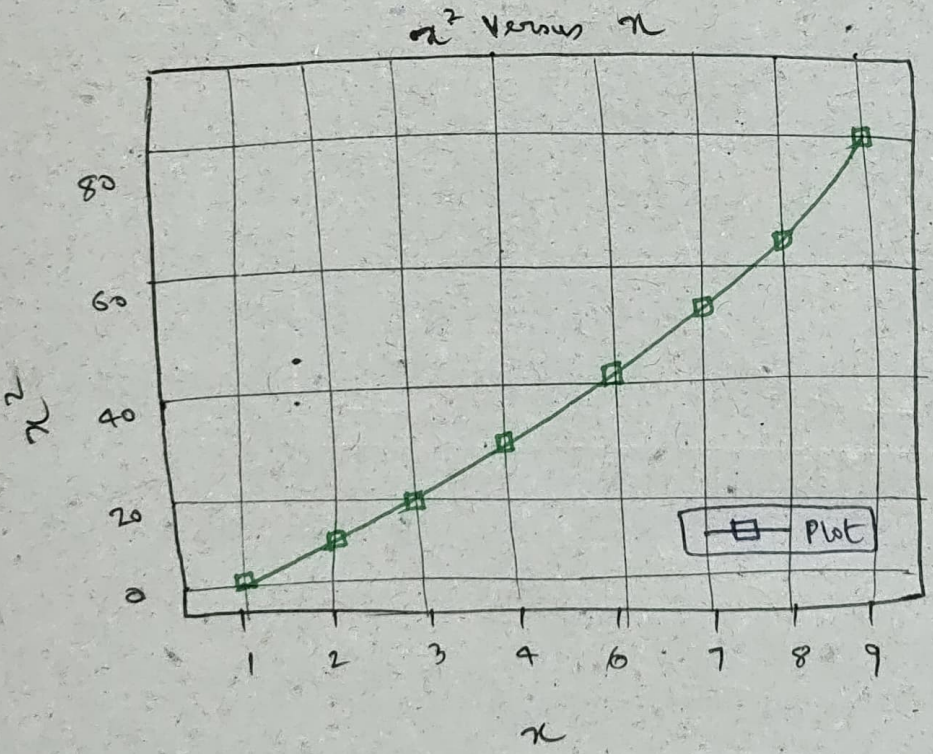
- A → x-axis
- A**2 → y-axis
- 's' → squares □
- '-' → solid line
- 'g' → green



Python plot

Plot with grid, title, legend, x-axis label, y-axis label

```
A = np.arange (1, 10)
plt.plot (A, A**2, 'g-s');
plt.grid (1, which = 'both');
plt.suptitle ('$ x^2 $ versus x');
plt.legend (["Plot"], loc = "lower right");
plt.xlabel ('x')
plt.ylabel ('$ x^2 $')
```



Plot the PDF of a Standard Gaussian RV in the range $[-5, 5]$

```
x = np.linspace(-5, 5, 1000)
f_x = (1 / np.sqrt(2 * np.pi)) * np.exp(-x**2 / 2)
plt.plot(x, f_x, 'b-', label="Standard Gaussian PDF")
plt.grid(1, which='both')
plt.suptitle("Standard Gaussian PDF")
plt.legend(loc='upper right')
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.show()
```