SCRIPTING AND PROGRAMMING LABORATORY FOR DATA ANALYSIS

Lecture 4 - numpy random numbers, plotting tools

Random numbers generator in numpy

NumPy- Random

Numpy's random number routines (**np.random**) produce pseudo random numbers using combinations of a BitGenerator to create sequences of random bits and a Generator to use those sequences to sample from different statistical distributions.

```
import numpy as np

rnd1 = np.random.random_sample() # 1 single random number
rnd2 = np.random.random_sample(7) # 7 random numbers
print(rnd1)
print(rnd2)
```

```
0.5820270803531266
[0.01075197 0.63074389 0.71969266 0.27217267 0.08726966 0.04203846
 0.46602294]
```

The function **random_sample** *called with no argument just generates a single random number drawn from the Uniform distribution in [0,1).*

If you call the function with a single integer number N, than N random numbers are generated in the form of a 1D numpy array.

NumPy- Random

Numpy's random number routines (**np.random**) produce pseudo random numbers using combinations of a BitGenerator to create sequences of random bits and a Generator to use those sequences to sample from different statistical

```
np.random.random_sample((3,3,4))
```

```
array([[[0.0841283 , 0.17685018, 0.18800577, 0.03807204],
        [0.30717857, 0.16209332, 0.09955438, 0.26659028],
        [0.93859288, 0.10107969, 0.15150067, 0.97975616]],

       [[0.30054528, 0.8486428 , 0.68002234, 0.69296248],
        [0.71142304, 0.97722757, 0.55454659, 0.0042325 ],
        [0.55115759, 0.46762784, 0.69950924, 0.52557438]],

       [[0.92477739, 0.13113899, 0.15631568, 0.4233557 ],
        [0.07579841, 0.68205432, 0.39262586, 0.29760239],
        [0.38471883, 0.2796238 , 0.81385195, 0.37302345]]])
```

Passing a tuple as shape, we get an array filled with random numbers with the desired shape

NumPy- Random

For each call of the random generator, you get different random numbers! If you need to generate always the same sequence you have to specify the **seed** of the random generator, i.e.:

```
np.random.seed(55) # explicit specification of the seed
print(np.random.random_sample(3))
```

```
[0.09310829 0.97165592 0.48385998]
```

***seed** takes in input a positive integer. The sequence of random numbers is now fixed. By re-executing the cell, you will get the same numbers!*

NumPy- Random

np.random has built-in functions to sample from several statistical distributions, e.g.:

- **random.uniform(low=0.0, high=1.0, size=None) ->** generates random number in an interval from low to high with equal probability.
- **random.normal(loc=0.0, scale=1.0, size=None)** -> generates samples from a Gaussian distribution with mean loc and standard deviation scale.
- **random.poisson(lam=1.0, size=None) ->** draw samples from a Poisson distribution with lam expected value. The Poisson distribution is the limit of the binomial distribution for large N.
- Many others, check here

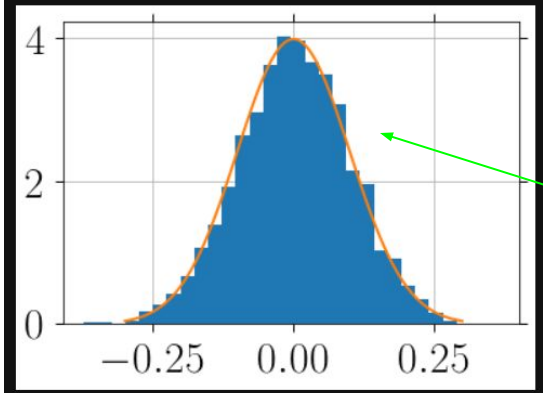
NumPy- Random

```
mu, sigma = 0, 0.1 # mean and standard deviation
s = np.random.normal(mu, sigma, 5000)

# gaussian expression
x_val = np.linspace(-0.3,0.3,100)
f_val = 1/(np.sqrt(2*np.pi*sigma**2)) * np.exp( -(x_val-mu)**2/(2*sigma**2) )
```

```
import matplotlib.pyplot as plt
plt.hist(s,bins=30,density=True) # note that density=True means that the histrogram is normalised such that the area is 1
plt.plot(x_val,f_val)
```

```
[<matplotlib.lines.Line2D at 0x7f440d1b3278>]
```



Generate 5000 random sample from a Gaussian with a given mean and std dev

We can check that the samples actually follow what we expect

NumPy- Random

Other useful function within **np.random** are:

- **random.choice(x, size=None, replace=True, p=None) ->** generates a random sample from a given 1-D array **x**. If **replace** is set to false, then no repetitions are allowed. With **p** different weights to the array elements are given.
- **random.randint(low, high=None, size=None, dtype=int) ->** returns random integers from low (inclusive) to high (exclusive).