CI Course - EX1.2

Theory Overview

The main objective of the class is to utilize various Python functionalities and libraries, particularly **Numpy** and **Matplotlib**, to demonstrate their usage briefly. Additionally, we will explore an implementation of linear regression using the **Scikit-learn** library.

Numpy

What is ***Numpy***?

NumPy is a Python library used for numerical computations. It provides an array object that is efficient for storing and manipulating large multidimensional arrays of homogeneous data, as well as a wide range of mathematical functions for performing operations on these arrays.

Some of the advantages of using NumPy include:

1. Efficient computation
2. Multidimensional array support
3. Integration with other libraries

Once you've installed NumPy you can import it as a library:

```
import numpy as np
```

Numpy arrays We can create an array by directly converting a **list**:

```
a = np.array([1,2,3,4,5])
b = np.array([1,2,3])
print(a, b)
print(a.shape, b.shape)
```

```
[1 2 3 4 5] [1 2 3]
(5,) (3,)
```

Shape is an attribute that arrays have and can be changed:

```
# Notice the two sets of brackets
```

```python
a = a.reshape(1,-1)
b = b.reshape(1,-1)
print(a, b)
print(a.shape, b.shape)
```

```
[[1 2 3 4 5]] [[1 2 3]]
(1, 5) (1, 3)
```

```python
a = a.reshape(-1,)
b = b.reshape(-1,)
print(a, b)
print(a.shape, b.shape)
```

```
[1 2 3 4 5] [1 2 3]
(5,) (3,)
```

We can create an array by directly converting a **list of lists** (matrix):

```python
c = np.array([[1,2,3],[4,5,6],[7,8,9],[-1,-8,9]])
print(c.shape)
print(c)
```

```
(4, 3)
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [-1 -8  9]]
```

Basic operations

```python
# Transpose
print(c.T)
print("---")

# dot product
print(c.dot(c.T)) # c * C.T
print("---")
print(c.T.dot(c)) # = np.matmul(c.T, c)
print("---")

# multiply
print(np.multiply(c, c)) # element wise multiplication
```

```
[[ 1  4  7 -1]
 [ 2  5  8 -8]
 [ 3  6  9  9]]
---
[[ 14  32  50  10]
 [ 32  77 122  10]
```

```
 [ 50 122 194  10]
 [ 10  10  10 146]]
---
[[ 67  86  81]
 [ 86 157  36]
 [ 81  36 207]]
---
[[ 1  4  9]
 [16 25 36]
 [49 64 81]
 [ 1 64 81]]
```

```
# concatenation
a = np.array([1,2,3,4,5])
b = np.array([6,7,8,9,10])

a = a.reshape(-1,1)
b = b.reshape(-1,1)
print(a.shape, b.shape)
c = np.concatenate((a,b), axis=0) # axis=0 - vertical | axis=1 - horizontal
print(c)
print(c.shape)
```

```
(5, 1) (5, 1)
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
(10, 1)
```

```
b = np.copy(a)
# b = a
print(a, b)
b[2] = -1000
print(a)
print(b)
```

```
[[1]
 [2]
 [3]
 [4]
 [5]] [[1]
```

```
 [2]
 [3]
 [4]
 [5]]
[[1]
 [2]
 [3]
 [4]
 [5]]
[[    1]
 [    2]
 [-1000]
 [    4]
 [    5]]
```

```
a1 = np.zeros((10,4))
a2 = np.ones((10,4))
print(a1)
print(a2)
```

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

Indexing and Slicing

```
#Get a value at an index
a[2]
```

```
array([3])
```

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

c = np.array([[1,2,3],[4,5,6],[7,8,9],[-1,-8,9]])
v = c[:2,:-1]
print(v)
```

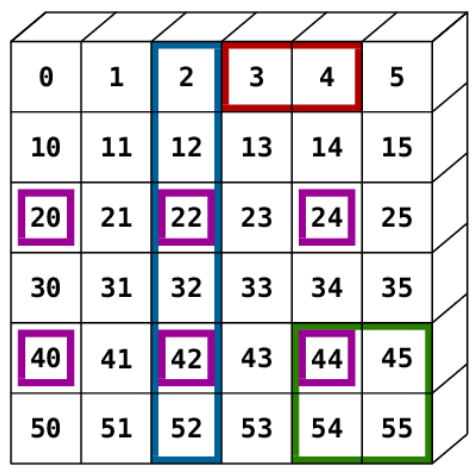
```
[[1 2]
 [4 5]]
```

```
v = c[:,1:3]
print(v)
```

```
[[ 2  3]
 [ 5  6]
 [ 8  9]
 [-8  9]]
```

Indexing a 2D matrix:

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Numpy - linear algebra (np.linalg)

```
a = np.array([1,3,4])
b = np.array([9,3,4])
a = np.linalg.norm(a - b) # ||a-b||
print(np.linalg.det(c[1:,:]))
```

```
-72.0
```

Numpy - other usefull operations

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

# statistics
print(np.sum(a), a.sum())
print(np.mean(a), a.mean())
print(np.std(c, axis=1))
print("---")

# inserting and delete
a = np.insert(a, 3, -1)
print(a)
a = np.delete(a, 3)
print(a)
print("---")

c = np.array([[1,2,3],[4,5,6],[7,8,9],[-1,-8,9]])
b = np.array([9,3,4]).reshape(1,-1)
c = np.insert(c, 2, b, axis=0)
print(c)
```

```
15 15
3.0 3.0
[0.81649658 0.81649658 0.81649658 6.97614985]
---
[ 1  2  3 -1  4  5]
[1 2 3 4 5]
---
[[ 1  2  3]
 [ 4  5  6]
 [ 9  3  4]
 [ 7  8  9]
 [-1 -8  9]]
```

Plotting with Matplotlib

What is **Matplotlib**?

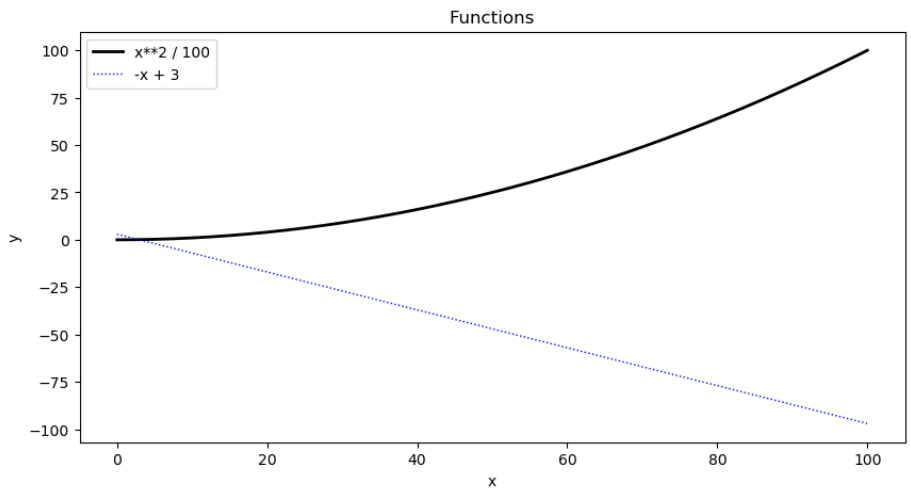
Matplotlib is a Python library used for creating static, animated, and interactive visualizations in Python.It provides a way to create a wide variety of charts, graphs, and other types of visualizations using a simple and intuitive syntax.

Once you've installed Matplotlib you can import its sublibraries such as pyplot:

```python
import matplotlib.pyplot as plt
```

Plotting 2d

```
x = np.linspace(0, 100, 1000)
y1 = x**2 / 100
y2 = -x + 3

f = plt.figure(figsize=(10,5))
plt.plot(x, y1, '-k', label = 'x**2 / 100', linewidth = 2)
plt.plot(x, y2, ':b', label = '-x + 3', linewidth = 1)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Functions')
plt.legend()

plt.show()
```

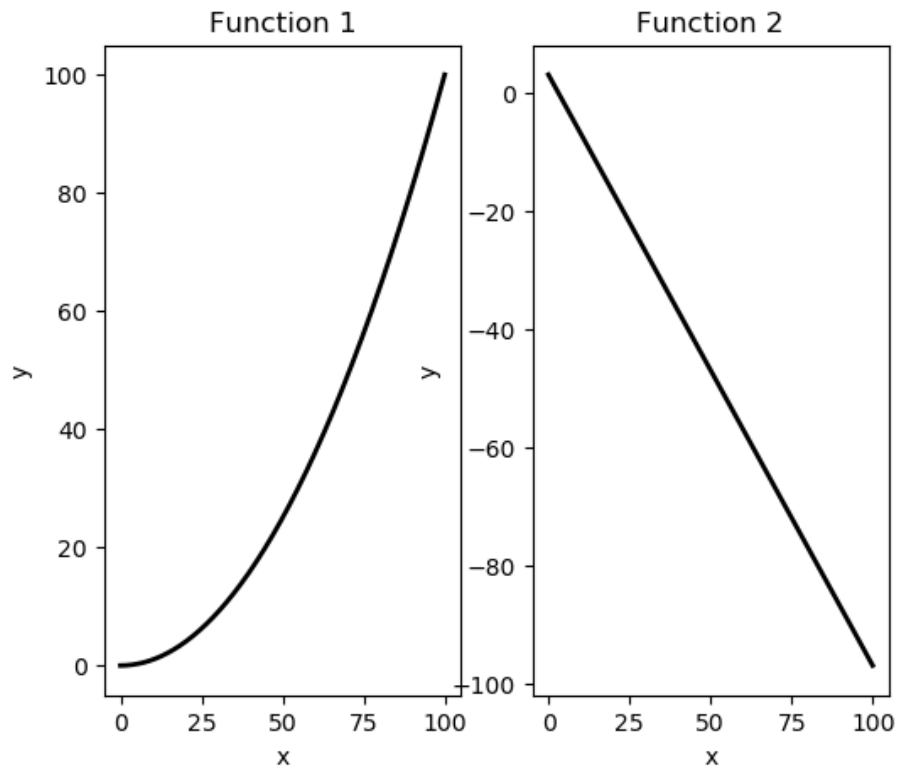


Another method of 2d plotting is using the **plt.subplots()** function, two variables are typically returned: figure and ax.

figure refers to the overall figure object that contains one or more subplots.**ax** is an array of axes objects that represent individual subplots within the figure.

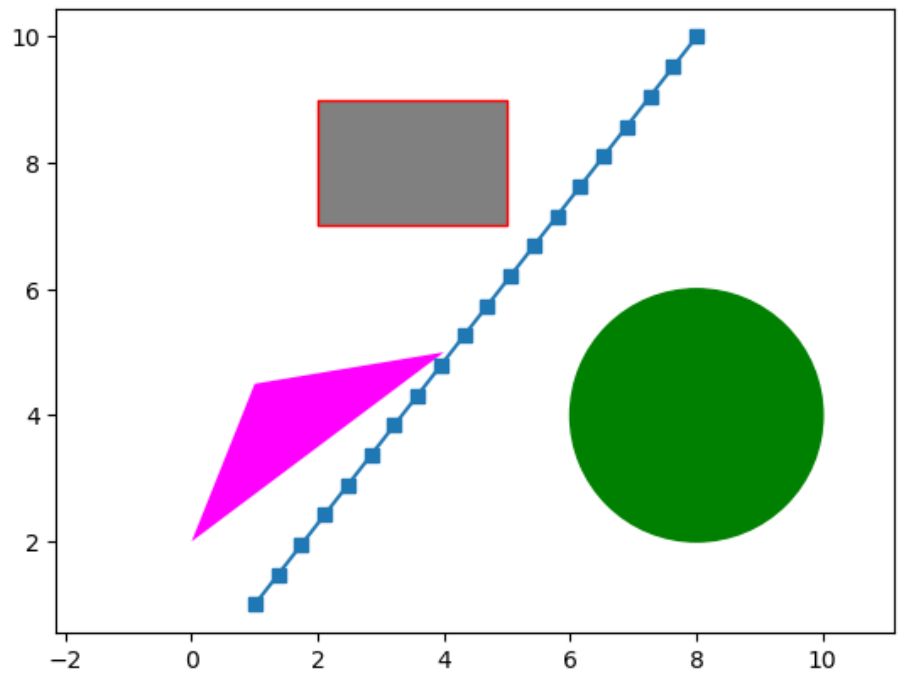
```
fig, ax = plt.subplots(1, 2, figsize=(6,5))#, sharey=True)
ax[0].plot(x, y1, '-k', linewidth = 2)
ax[0].set_title('Function 1')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y')

ax[1].plot(x, y2, '-k', linewidth = 2)
ax[1].set_title('Function 2')
ax[1].set_xlabel('x')
ax[1].set_ylabel('y')
```

```
plt.show()
```




```
xs = np.array([1,1])
xg = np.array([8,10])
S = np.linspace(0, 1, 20)   # line = xs + s * (xg-xs)
X = np.array([xs + s * (xg-xs) for s in S])

circ = [8,4]
ro = 2
pts = np.array([[0,2], [4,5], [1,4.5]])

fig, ax = plt.subplots()
ax.plot(X[:,0], X[:,1], '-s')
circle = plt.Circle((circ[0], circ[1]), ro, color='g')
rec = plt.Rectangle((2,7), 3, 2, fc='gray',ec="red")
p = plt.Polygon(pts, fc = 'magenta')

ax.add_artist(circle)
ax.add_artist(rec)
ax.add_artist(p)
```

```
plt.axis('equal')

plt.show()
```

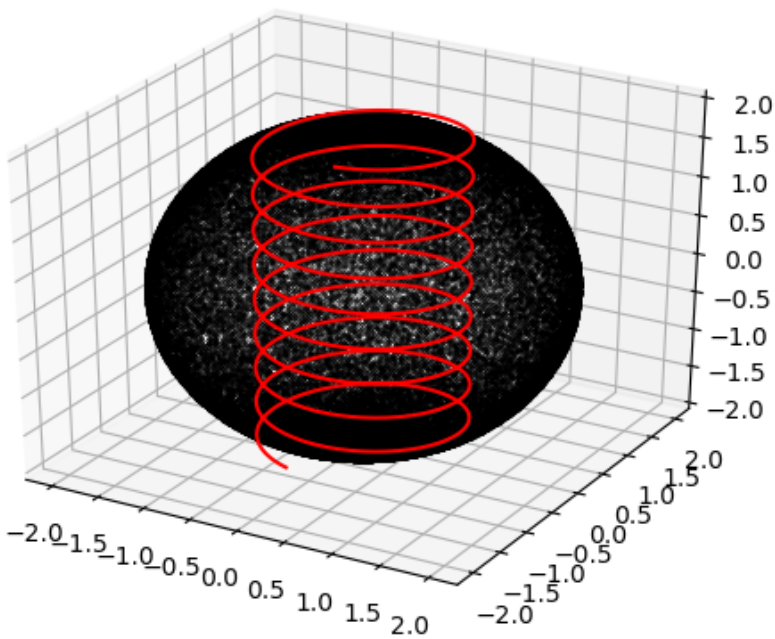


Plotting 3d A easy top use library for 3d plotting is **mplot3d**. Once installed you can import it by:

```
from mpl_toolkits import mplot3d ## check with avishai

r = 2.0
P = []
for _ in range(100000):

    # a -> b
    # np.random.random() * (a-b) + b
    theta = np.random.random() * (2 * np.pi) - np.pi
    phi = np.random.random() * 2 * np.pi + 0

    p = np.array([r * np.sin(theta) * np.cos(phi),
                  r * np.sin(theta) * np.sin(phi),
                  r * np.cos(theta)])
    P.append(p)
P = np.array(P)
```

```
print(P.shape)

Z = np.linspace(-2, 2, 1000)
X = np.sin(14*Z)
Y = np.cos(14*Z)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot3D(P[:,0], P[:,1], P[:,2], '.k', markersize = 0.5)
ax.plot3D(X, Y, Z, 'red')

plt.show()
```

```
(100000, 3)
```



Sklearn - Linear Regression

What is **scikit-learn** (sklearn) ?

Scikit-learn (also known as sklearn) is a popular open-source library for machine learning in Python. It provides a range of supervised and unsupervised learning algorithms for classification, regression, clustering, dimensionality reduction, model selection, and data preprocessing.The library is considered as easy to use,

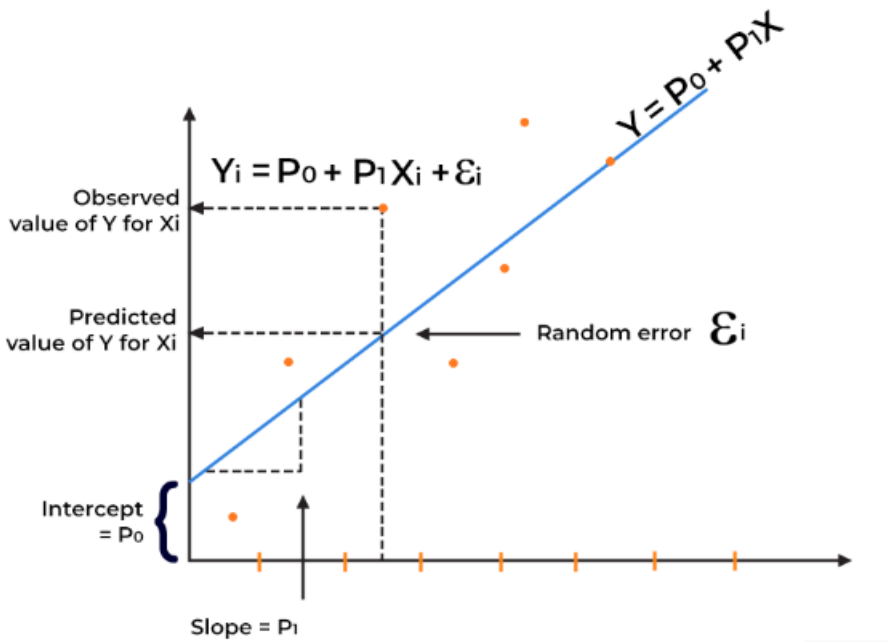
efficient and scalable.

What is **linear regression**?

Linear regression is a type of regression analysis that models the relationship between a dependent variable and one or more independent variables as a linear function. The goal of linear regression is to find the best-fit line that describes the relationship between the dependent variable and the independent variable(s).

Linear regression is often taught as a fundamental machine learning technique for predictive modeling. **The basic idea** is to use a set of training data to learn the relationship between the input variables (independent variables) and the output variable (dependent variable).

The relationship can be presented as a line that defined by a slope and an intercept, which are calculated using the training data **to minimizes the sum of the squared differences between the predicted values and the actual values**.This method is called the least-squares method. Once the line is calculated, we can use it to make predictions for new data.



Once you've installed Scikit-learn (=sklearn) you can import its sublibraries and objects such as LinearRegression Class:

```
from sklearn.linear_model import LinearRegression
```

The general flow of fitting and using a model is:

1. Generating / Load training data
2. Fit model

3. Predict / Test new data samples
4. Evaluate and visualize if possible

First lets present what is **np.random.RandomState()**:

np.random.RandomState() is a function in the NumPy library that allows you to create a random number generator with a specific seed value. The seed value determines the sequence of random numbers that will be generated.

for example:

```
# create a random number generator with seed 42
rng = np.random.RandomState(42)

# generate 5 random integers between 0 and 10
rand_ints = rng.randint(low=0, high=10, size=5)

print(rand_ints)
```

```
[6 3 7 4 6]
```

In the example above, we created a random number generator rng with seed value 42.

Then we used this generator to generate an array of 5 random integers between 0 and 10. The sequence of random numbers generated by rng will always be the same, as long as we use the same seed value.

Now lets move on to np.random.RandomState().normal(). This function generates random numbers from a normal (Gaussian) distribution.

Heres an example:

```
# create a random number generator with seed 42
rng = np.random.RandomState(42)

# generate 5 random numbers from a normal distribution with mean 0 and standard deviation 1
rand_nums = rng.normal(loc=0, scale=1, size=5)

print(rand_nums)
```

```
[ 0.49671415 -0.1382643   0.64768854  1.52302986 -0.23415337]
```

In the example above, we used the same random number generator rng with seed value 42 to generate an array of 5 random numbers from a normal distribution with mean 0 and standard deviation 1.

Generate training data

```
# Generate training data
rng = np.random.RandomState(42)
```

```
x = np.linspace(0, 10, num=2000).reshape(-1,1)
y = 3.2 * x + rng.normal(scale=x / 2)
```

In the code above - rng.normal(scale=x / 2) generates random values from a normal distribution with mean 0 and a standard deviation of x / 2. The scale parameter determines the standard deviation of the distribution, which is proportional to x. So as x increases, the range of possible values for y generated by the normal distribution also increases.

Fit linear model

```
model = LinearRegression()
model.fit(x, y)
```

Predict values for the x range

```
y_predicted = model.predict(x)
y_predicted
```

Plot

```
plt.figure()
plt.plot(x, y, 'o', alpha=0.5, markersize=1, label = 'Data')
plt.plot(x, y_predicted, '-k', label = 'Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Summary

In this class we covered:

- Numpy and matplotlib functionalities
- Linear regression example using sklearn

Helpful and extra links

- Numpy tutorials - practice online
- matplotlib tutorials - practice online
- sklearn - LinearRegression class
- Linear regression - in 2 minutes
