

Introducere în *NumPy* și *Matplotlib*

1. Numpy

- cea mai utilizată bibliotecă Python pentru calculul matematic
- dispune de obiecte multidimensionale (vectori, matrici) și funcții optimizate să lucreze cu acestea

Importarea bibliotecii:

```
import numpy as np
```

Vectori multidimensionali:

- inițializați folosind o listă din *Python*

```
a = np.array([1, 2, 3])
print(a)           # => [1 2 3]
print(type(a))     # tipul obiectului a => <class 'numpy.ndarray'>
print(a.dtype)     # tipul elementelor din a => int32
print(a.shape)     # tuple continand lungimea lui a pe fiecare dimensiune => (3,)
print(a[0])        # acceseaza elementul avand indexul 0 => 1

b = np.array([[1, 2, 3], [4, 5, 6]])
print(b.shape)     # => (2, 3)
print(b[0][2])     # => 3
print(b[0, 2])     # => 3

c = np.asarray([[1, 2], [3, 4]])
print(type(c))     # => <class 'numpy.ndarray'>
print(c.shape)     # => (2, 2)
```

- creați folosind funcții din *NumPy*

```
zero_array = np.zeros((3, 2))           # creeaza un vector continand numai 0
print(zero_array)                       # => [[0. 0.]
                                         #      [0. 0.]
                                         #      [0. 0.]]

ones_array = np.ones((2, 2))            # creeaza un vector continand numai 1
print(ones_array)                       # => [[1. 1.]
                                         #      [1. 1.]]

constant_array = np.full((2, 2), 8)     # creeaza un vector constanta
print(constant_array)                   # => [[8 8]
                                         #      [8 8]]
```

```

identity_matrix = np.eye(3)      # creeaza matricea identitate de dimensiune 3x3
print(identity_matrix)           # => [[1. 0. 0.]
                                  #      [0. 1. 0.]
                                  #      [0. 0. 1.]]

random_array = np.random.random((1,2)) # creeaza un vector cu valori aleatoare
print(random_array)               # => ex: [[0.00672748 0.12277961]]

mu, sigma = 0, 0.1
gaussian_random = np.random.normal(mu, sigma, 10) # creeaza un vector cu valori
                                                    # random cu distributie
                                                    # Gaussiană de medie mu si
                                                    # deviatie standard sigma

first_5 = np.arange(5)           # creeaza un vector continand primele 5 numere naturale
print(first_5)                   # => [0 1 2 3 4]

```

Indexare:

Slicing: extragerea unei submultimi - trebuie specificati indicii doriti pe fiecare dimensiune

```

array_to_slice = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
slice = array_to_slice[:, 0:3]      # luam toate liniile si coloanele 0, 1, 2
print(slice)                        # => [[ 1  2  3]
                                      #      [ 5  6  7]
                                      #      [ 9 10 11]]

# !! modificarea slice duce automat la modificarea array_to_slice
print(array_to_slice[0][0])         # => 1
slice[0][0] = 100
print(array_to_slice[0][0])         # => 100

# pentru a nu se intampla acest lucru submultimea poate fi copiata
slice_copy = np.copy(array_to_slice[:, 0:3])
slice_copy[0][0] = 100
print(slice_copy[0][0])             # => 100
print(array_to_slice[0][0])         # => 1

```

În cazul în care unul din indicii este un întreg, dimensiunea submultimii returnate este mai mică decât dimensiunea inițială:

```

slice_1 = array_to_slice[2:3, :]
print(slice_1)                      # => [[ 9 10 11 12]]
slice_2 = array_to_slice[2, :]
print(slice_2)                      # => [ 9 10 11 12]

# returnarea tuturor elementelor într-un array 1D:
slice_1d = np.ravel(slice_1)
print(slice_1d)                     # => [ 9 10 11 12]

```



```

print(np.multiply(x, y))

# Impartire element cu element => [[ 0.2          0.33333333]
#                                [ 0.42857143  0.5]]
print(x / y)
print(np.divide(x, y))

# Radical element cu element => [[ 1.          1.41421356]
#                                [ 1.73205081  2.]]
print(np.sqrt(x))

# Ridicare la putere
my_array = np.arange(5)
powered = np.power(my_array, 3)
print(powered)           # => [ 0  1  8 27 64]

```

Produsul scalar:

```

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# vector x vector => 219
print(v.dot(w))
print(np.dot(v, w))

# matrice x vector => [29 67]
print(np.matmul(x, v))

# matrice x matrice => [[19 22]
#                       [43 50]]
print(np.matmul(x, y))

```

Operatii pe matrici:

```

# transpusa unei matrici
my_array = np.array([[1, 2, 3], [4, 5, 6]]) # [[1, 2, 3],
#                                           # [4, 5, 6]]

print(my_array.T)           # => [[1, 4],
#                               # [2, 5],
#                               # [3, 6]]

# inversa unei matrici
my_array = np.array([[1., 2.], [3., 4.]])
print(np.linalg.inv(my_array)) # => [[-2. ,  1. ],
#                                     [ 1.5, -0.5]]

```

NumPy dispune de funcții care fac operații pe o anumită dimensiune.

```
x = np.array([[1, 2],[3, 4]])

# suma pe o anumita dimensiune
print(np.sum(x))          # Suma tuturor elementelor => 10
print(np.sum(x, axis=0))  # Suma pe coloane => [4 6]
print(np.sum(x, axis=1))  # Suma pe linii => [3 7]
# putem specifica si mai multe axe pe care sa se faca operatia:
print(np.sum(x, axis=(0,1))) # Suma tuturor elementelor => 10

# media pe o anumita dimensiune
y = np.array([[1, 2, 3, 4], [5, 6, 7, 8]], [[1, 2, 3, 4], [5, 6, 7, 8]], [[1, 2, 3, 4], [5, 6, 7, 8]])
print(y.shape)            # => (3, 2, 4)
print(y)                  # => [[[1 2 3 4]
#                          [5 6 7 8]]
#                          [[1 2 3 4]
#                          [5 6 7 8]]
#                          [[1 2 3 4]
#                          [5 6 7 8]]]

print(np.mean(y, axis=0))  # => [[1. 2. 3. 4.]
#                          [5. 6. 7. 8.]]

print(np.mean(y, axis=1))  # => [[3. 4. 5. 6.]
#                          [3. 4. 5. 6.]
#                          [3. 4. 5. 6.]]

# indexul elementului maxim pe fiecare linie
z = np.array([[10, 12, 5],[17, 11,19]])
print(np.argmax(z, axis=1)) # => [1 2]
```

Broadcasting:

- mecanism care ofera posibilitatea de a face operatii aritmetice intre vectori de dimensiuni diferite
- vectorul mai mic este multiplicat astfel incat sa se potriveasca cu cel mai mare, operatia fiind apoi realizata pe cel din urma

```
# Vrem sa adaugam un vector (v) la fiecare linie a unei matrici (x)
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v
print(y) # => [[ 2  2  4]
#           [ 5  5  7]
#           [ 8  8 10]
#           [11 11 13]]
```

Reguli de broadcasting:

1. Dacă vectorii nu au același număr de dimensiuni, vectorul mai mic este extins cu câte o dimensiune, până când acest lucru este realizat.

ex: Dacă avem 2 vectori **a** și **b** cu
a.shape = (3, 4)
b.shape = (6,)
b este extins la dimensiunea (6, 1)

2. Cei 2 vectori se numesc compatibili pe o dimensiune dacă au aceeași lungime pe acea dimensiune sau dacă unul dintre ei are lungimea 1.

ex: Considerăm vectorii:
a astfel încât: a.shape = (3, 4)
b astfel încât: b.shape = (6, 1)
c astfel încât: c.shape = (3, 5)

a și **c** sunt compatibili pe prima dimensiune
a și **b** sunt compatibili pe cea de-a doua dimensiune

3. Pe cei 2 vectori se poate aplica broadcasting dacă ei sunt compatibili pe toate dimensiunile.
4. La broadcasting fiecare vector se comportă ca și cum ar avea, pe fiecare dimensiune, lungimea maximă dintre cele două dimensiuni inițiale (maximul dimensiunilor elementwise)

ex: La broadcasting vectorii **a** și **b** cu
a.shape = (3, 4)
b.shape = (3, 1)
se comportă ca și cum ar avea dimensiunea (3, 4)

5. În fiecare dimensiune pe care unul din vectori avea dimensiunea 1, iar celălalt mai mare, primul vector se comportă ca și cum ar fi copiat de-a lungul acelei dimensiuni.

ex: Considerăm vectorii:
a = [[1, 2, 3],
[4, 5, 6]] , a.shape = (2, 3)

b = [[1.],
[1.]] , b.shape = (2, 1)

Când vrem să facem o operație de broadcasting, vectorul b va fi copiat de-a lungul celei de-a doua dimensiuni, astfel încât el devine:

```
b = [[1., 1., 1.],  
      [1., 1., 1.]] , b.shape = (2, 3)
```

operația fiind acum realizată pe vectori de aceeași dimensiune.

2. Matplotlib

- bibliotecă utilizată pentru plotarea datelor

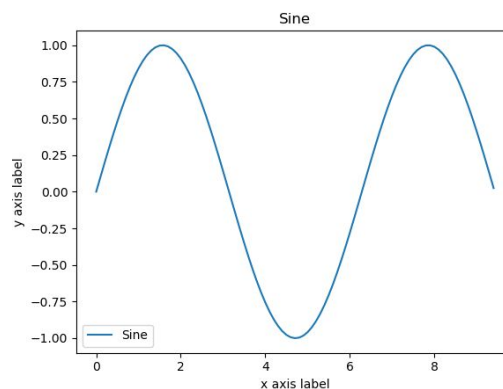
Importarea bibliotecii:

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

Plotare:

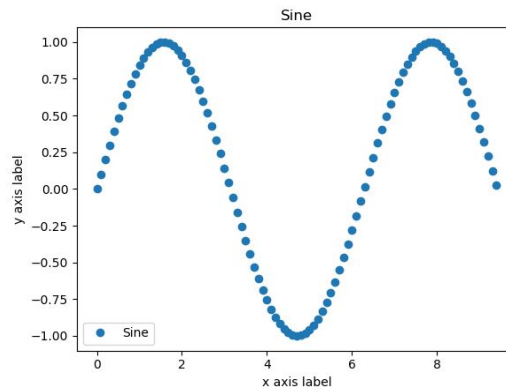
- cea mai importantă funcție este *plot*, care permite afișarea datelor 2D

```
# Calculează coordonatele (x, y) ale punctelor de pe o curbă sin  
# x - valori de la 0 la 3 * np.pi, luate din 0.1 în 0.1  
x = np.arange(0, 3 * np.pi, 0.1)  
y = np.sin(x)  
  
# Plotează punctele  
plt.plot(x, y)  
  
# Adaugă etichete pentru fiecare axă  
plt.xlabel('x axis label')  
plt.ylabel('y axis label')  
  
# Adaugă titlu  
plt.title('Sine')  
  
# Adaugă legendă  
plt.legend(['Sine'])  
  
# Afișează figura  
plt.show()
```



OBS. Pentru a plota punctele independente, fără a face interpolare ca în exemplul anterior, se poate specifica un al treilea parametru în funcția plot, astfel:

```
plt.plot(x, y, 'o')
```



Plotarea mai multor grafice in cadrul aceleiasi figuri:

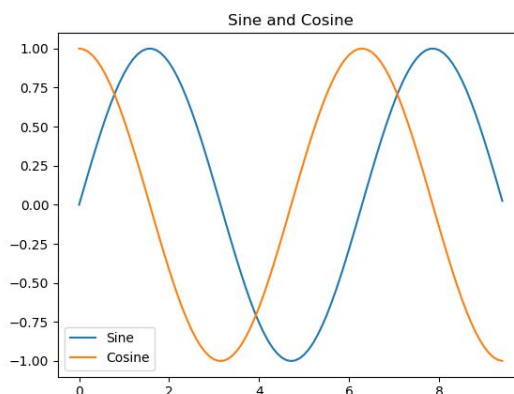
```
# Calculeaza coordonatele (x, y) ale punctelor de pe o curba sin, respectiv cos
# x - valori de la 0 la 3 * np.pi, Luata din 0.1 in 0.1
x = np.arange(0, 3 * np.pi, 0.1)
y_1 = np.sin(x)
y_2 = np.cos(x)

# Ploteaza punctele in aceeasi figura
plt.plot(x, y_1)
plt.plot(x, y_2)

# Adauga titlu
plt.title('Sine and Cosine')

# Adauga Legenda
plt.legend(['Sine', 'Cosine'])

# Afiseaza figura
plt.show()
```



Plotarea simultana a mai multor grafice in figuri diferite:

```
# Calculeaza coordonatele (x, y) ale punctelor de pe o curba sin, respectiv cos
# x - valori de la 0 la 3 * np.pi, Luata din 0.1 in 0.1
```

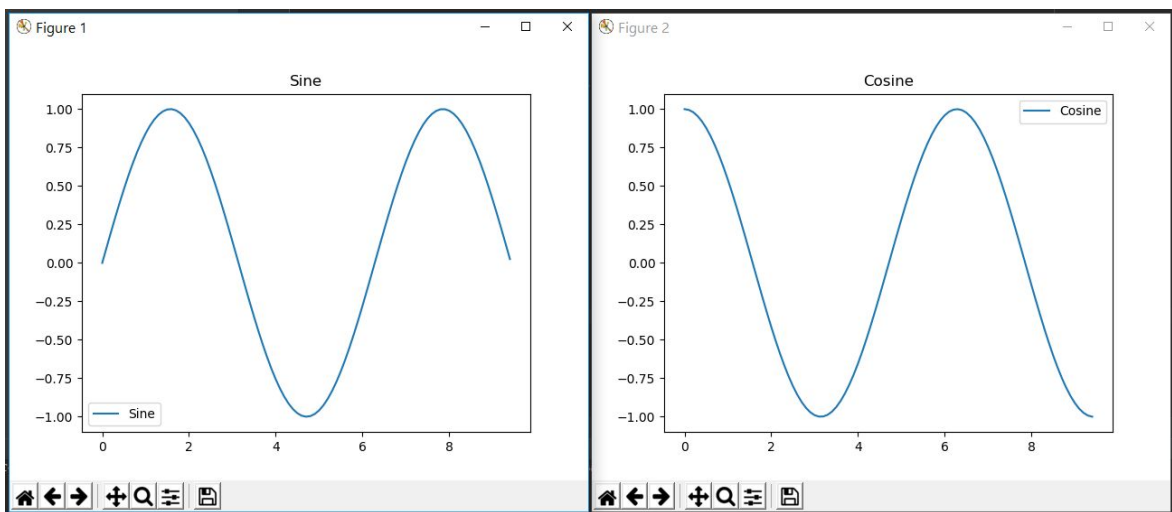


```
x = np.arange(0, 3 * np.pi, 0.1)
y_1 = np.sin(x)
y_2 = np.cos(x)

# definim primul plot in figura 1
first_plot = plt.figure(1)
plt.plot(x, y_1)
plt.title('Sine')
plt.legend(['Sine'])

# definim cel de-al doilea plot in figura 2
second_plot = plt.figure(2)
plt.plot(x, y_2)
plt.title('Cosine')
plt.legend(['Cosine'])

# afisam figurile
plt.show()
```



Sublotare:

- putem plota mai multe lucruri in cadrul aceleiasi figuri

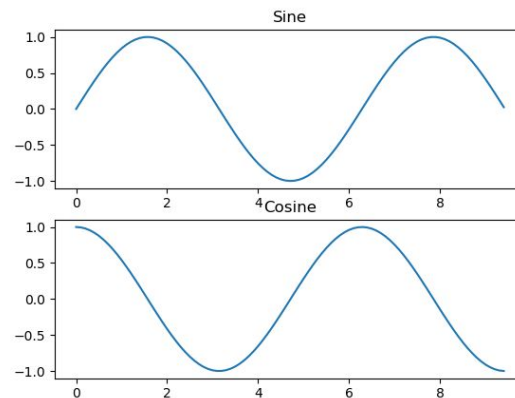
```
# Calculeaza coordonatele (x, y) ale punctelor de pe o curba sin, respectiv cos
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Creeaza un grid avand inaltimea 2 si latimea 1
# si seteaza primul subplot ca activ
plt.subplot(2, 1, 1)

# Ploteaza primele valori
plt.plot(x, y_sin)
plt.title('Sine')
```

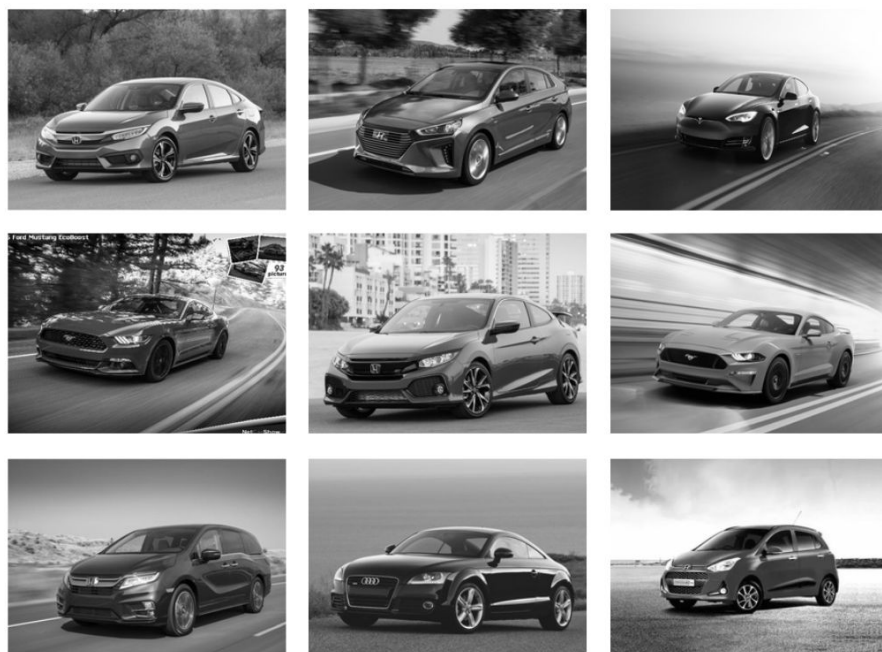
```
# Seteaza cel de-al doilea subplot ca activ
# si ploteaza al doilea set de date
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Afiseaza figura
plt.show()
```



Exercitii

1. Se dau urmatoarele 9 imagini de dimensiuni 400x600. Valorile acestora au fost salvate in fisierele "images/car_{idx}.npy".



- a. Cititi imaginile din aceste fisiere si salvati-le intr-un np.array (va avea dimensiunea 9x400x600).

Obs: Citirea din fisier se face cu ajutorul funcției:

```
image = np.load(file_path)
```

Aceasta întoarce un `np.array` de dimensiune 400x600.

- b. Calculați suma valorilor pixelilor tuturor imaginilor.
- c. Calculați suma valorilor pixelilor pentru fiecare imagine în parte.
- d. Afișați indexul imaginii cu suma maximă.
- e. Calculați imaginea medie și afișați-o.

Obs: Afișarea imaginii medii se poate face folosind biblioteca *scikit-image* în următorul mod:

```
from skimage import io
io.imshow(mean_image.astype(np.uint8)) # petru a putea fi afisata
                                         # imaginea trebuie sa aiba
                                         # tipul unsigned int
io.show()
```

Dacă biblioteca nu este instalată, acest lucru se poate face prin rularea comenzii sistem `pip install scikit-image`.

- f. Cu ajutorul funcției `np.std(images_array)`, calculați deviația standard a imaginilor.
- g. Normalizați imaginile. (se scade imaginea medie și se împarte rezultatul la deviația standard)
- h. Decupați fiecare imagine, afișând numai liniile cuprinse între 200 și 300 și coloanele cuprinse între 280 și 400.