04/09/2023

Python pour l'ingénieur

Calculs scientifiques

Module de formation

TABLE DES MATIERES

Les packages nécessaires

http://numpy.org

NumPy is an extension to the Python programming language, adding support for large, **multi-dimensional arrays and matrices**, along with a large library of high-level mathematical functions to operate on these arrays.

http://scipy.org

SciPy contains modules for **optimization**, **linear algebra**, **integration**, **interpolation**, **special functions**, **FFT**, **signal and image processing**, **ODE solvers** and other tasks common in science and engineering.

http://matplotlib.org

Matplotlib is a python **2D plotting** library
(alternatives : seaborn, pyqtgraph)

Installation

- Windows
  - Les distributions **WinPython** et **Anaconda** contiennent tous les packages nécessaires au calcul scientifique.

- OSX
  - La distribution **Anaconda** contient tous les packages nécessaires au calcul scientifique.

- Linux (Ubuntu)
  - Les packages nécessaires peuvent êtres installées avec apt : **python3-numpy, python3-scipy, python3-matplotlib**

Classe ndarray

- Les vecteurs (1D), matrices (2D) et tableaux à N dimensions sont représentés par un objet de la classe **ndarray** définie par numpy.

- La fonction **array()** de numpy permet de créer un objet ndarray à partir d'une liste python ou d'un autre objet ndarray fourni en paramètre.

- L'attribut **shape** d'un objet ndarray renvoie le nombre d'éléments par dimension.

```
In [1]: import numpy as np

In [2]: v = np.array([1, 2, 3])
        print(v)
        [1 2 3]

In [3]: m = np.array([[1, 2, 3], [4, 5, 6]])
        print(m)
        [[1 2 3]
         [4 5 6]]

In [4]: t = np.array([m, m])
        print(t)
        [[[1 2 3]
          [4 5 6]]

         [[1 2 3]
          [4 5 6]]]

In [5]: v.shape
Out[5]: (3,)

In [6]: m.shape
Out[6]: (2, 3)

In [7]: t.shape
Out[7]: (2, 2, 3)

In [8]: type(v), type(m), type(t)
Out[8]: (numpy.ndarray, numpy.ndarray, numpy.ndarray)
```

Créer un ndarray

```
In [2]: np.array([2, 3, 1, 0])
Out[2]: array([2, 3, 1, 0])

In [3]: np.array([[1, 2], [3, 4]])
Out[3]: array([[1, 2],
               [3, 4]])

In [4]: np.zeros((2, 3))
Out[4]: array([[ 0.,  0.,  0.],
               [ 0.,  0.,  0.]])

In [5]: np.zeros((2, 3, 5))
Out[5]: 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.],
                [ 0.,  0.,  0.,  0.,  0.]]])

In [6]: np.arange(10)
Out[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [7]: np.arange(2, 3, 0.1)
Out[7]: array([ 2. ,  2.1,  2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9])

In [8]: np.arange(9).reshape(3, 3)
Out[8]: array([[0, 1, 2],
               [3, 4, 5],
               [6, 7, 8]])

In [9]: np.linspace(1, 4, 6)
Out[9]: array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])

In [10]: np.random.randint(1, 10, 9).reshape(3, 3)
Out[10]: array([[1, 5, 5],
                [4, 9, 7],
                [7, 3, 6]])

In [11]: np.random.randn(3, 3)
Out[11]: array([[ 1.3386108 ,  0.06383958, -0.93302699],
                [-1.61788917, -0.12871647,  0.343064  ],
                [-0.81545832,  0.5010163 , -0.1750691 ]])
```

Et aussi :

```
ones()
empty()
full()
eye()
zeros_like()
ones_like()
empty_like()
full_like()
```

Rem : Un array numpy peut être converti en liste Python avec tolist()

```
In [10]:
v = np.arange(9)

In [11]:
m = np.arange(9).reshape(3,3)

In [12]:
v.tolist()

Out[12]:
[0, 1, 2, 3, 4, 5, 6, 7, 8]

In [13]:
m.tolist()

Out[13]:
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Pour en savoir plus :

https://numpy.org/doc/stable/user/basics.creation.html

Exercices

1) Créer un tableau 8x8 de type « checkboard » (comme ci-contre)
   - Indice : utiliser np.tile()

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

2) Créer une matrice à 3 dimensions (4x4x3) d'entiers aléatoires compris entre 1 et 48, sans qu'aucun nombre ne soit présent plus d'une fois.

Types de données

Un array numpy utilise un type de données précis pour tous ses éléments (entier, flottant, signé, non signé, nombre de bits). Exemples de types : int8, uint8, int16, uint16, int32, uint32, int64, float16, float32, float64, complex....

U comme Unsigned

[1]:
```
import numpy as np

arr = np.ndarray([1,2,3])
print(arr.dtype)
```

```
float64
```

Par défaut le type float64 a été choisi

Le choix du type est important : il détermine les valeurs min et max des valeurs, la quantité de mémoire nécessaire, la rapidité des calculs. Il ne faut pas prendre float64 systématiquement si ce n'est pas nécessaire, car cela a un coût. float64 a de toute façon aussi une valeur maximum.

[2]:

dtype permet de choisir le type de données

```
arr = np.ndarray([1,2,3], dtype="int32")    # ou dtype=np.int32
print(arr.dtype)
arr = np.zeros((4,4), dtype="float32")      # ou dtype=np.float32
print(arr.dtype)
```

```
int32
float32
```

Attention aux overflows qui peuvent se produire. Une fois la valeur maximum dépassée, la valeur numérique "reboucle", en général sans aucun message d'erreur.

[3]:
```
arr = np.zeros(3, dtype="uint8")   # valeur max d'un uint8 = 255
arr += 255
print(arr)
arr += 1
print(arr)
arr += 1
print(arr)
```

```
[255 255 255]
[0 0 0]
[1 1 1]
```

Numpy permet de convertir un tableau dans un autre type (en créant un nouveau tableau)

[4]:

astype() permet de convertir vers un autre type

```
arr_int = np.zeros(3, dtype="uint8")
print(arr_int.dtype)
arr_float = arr_int.astype("float32")
print(arr_float.dtype)
```

```
uint8
float32
```

Opérations basiques

```
In [1]: import numpy as np

In [2]: v = np.arange(9)
        print(v)
        [0 1 2 3 4 5 6 7 8]

In [3]: v+v
Out[3]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16])

In [4]: v+2
Out[4]: array([ 2,  3,  4,  5,  6,  7,  8,  9, 10])

In [5]: v*v
Out[5]: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64])

In [6]: np.dot(v,v)
Out[6]: 204
```

```
In [7]: m = v.reshape(3, 3)
        print(m)
        [[0 1 2]
         [3 4 5]
         [6 7 8]]

In [8]: m*m
Out[8]: array([[ 0,  1,  4],
               [ 9, 16, 25],
               [36, 49, 64]])

In [9]: np.dot(m, m)      # ou m @ m à partir de Python 3.5
Out[9]: array([[ 15,  18,  21],
               [ 42,  54,  66],
               [ 69,  90, 111]])

In [10]: m.transpose()  # ou m.T
Out[10]: array([[0, 3, 6],
                [1, 4, 7],
                [2, 5, 8]])

In [11]: m.min(), m.max()
Out[11]: (0, 8)

In [12]: m.mean(), m.std()
Out[12]: (4.0, 2.5819888974716112)

In [13]: m.sum(), m.prod()
Out[13]: (36, 0)

In [14]: np.cos(m), np.sin(m)
Out[14]: (array([[ 1.        ,  0.54030231, -0.41614684],
                 [-0.9899925 , -0.65364362,  0.28366219],
                 [ 0.96017029,  0.75390225, -0.14550003]]),
          array([[ 0.        ,  0.84147098,  0.90929743],
                 [ 0.14112001, -0.7568025 , -0.95892427],
                 [-0.2794155 ,  0.6569866 ,  0.98935825]]))
```

Exercices

3) Créer un tableau de nombres entiers aléatoires de taille 4x3 (4 colonnes, 3 lignes), puis créer un tableau 4x1 dans lequel chaque élément est la somme des éléments d'une colonne du premier tableau.

4) Soit A et B deux vecteurs de même taille. Calculer (A+B)*(-A/2) sans création d'autres matrices (c'est à dire sans copies, on dit « in-place »)
   - Indice : np.add(out=), np.negative(out=), np.multiply(out=), np.divide(out=)

Indexing

Les ndarray sont indexables :

In [9]:
```
v[0] = -1
print(v)
print(v[0])
```

```
[-1  2  3]
-1
```

In [10]:
```
m[0, 0] = -1
print(m)
print(m[0, 0])
```

```
[[-1  2  3]
 [ 4  5  6]]
-1
```

[0,0]	[0,1]
[1,0]	[1,1]

m[ligne, colonne]

In [11]:
```
t[0, 0, 0] = -1
print(t)
print(t[0, 0, 0])
```

```
[[[-1  2  3]
  [ 4  5  6]]

 [[ 1  2  3]
  [ 4  5  6]]]
-1
```

Les ndarray sont itérables :

In [4]:
```
for i in v:
    print(i)
```

```
1
2
3
```

Les ndarray sont sliceable :

In [12]:
```
v = np.array([1, 2, 3, 4, 5, 6])
print(v[2:5])
v[2:5] = -1
print(v)
```

```
[3 4 5]
[ 1  2 -1 -1 -1  6]
```

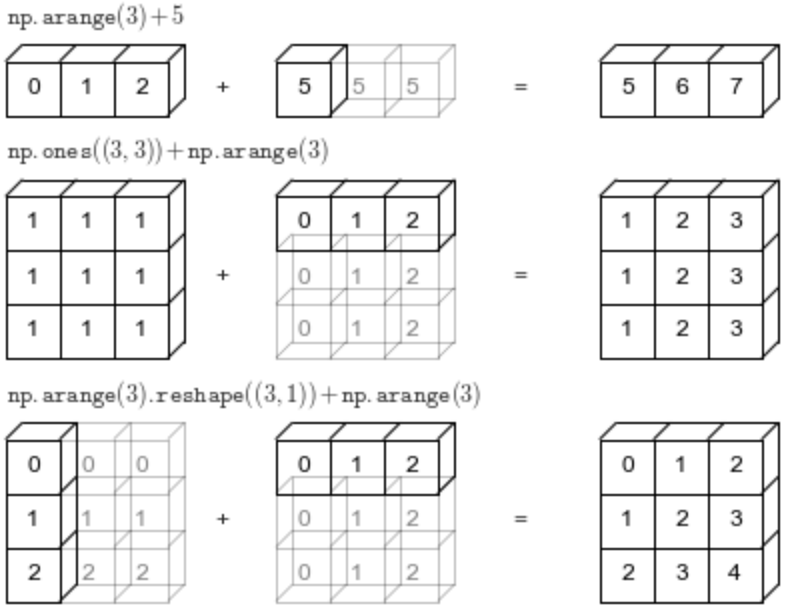
Broadcasting

```
In [1]: import numpy as np
```

```
In [13]: v = np.arange(3)
         v, v + 5
Out[13]: (array([0, 1, 2]), array([5, 6, 7]))
```

```
In [14]: m = np.ones((3, 3))
         v = np.arange(3)
         m, v, m + v
Out[14]: (array([[1., 1., 1.],
                 [1., 1., 1.],
                 [1., 1., 1.]]),
          array([0, 1, 2]),
          array([[1., 2., 3.],
                 [1., 2., 3.],
                 [1., 2., 3.]]))
```

```
In [19]: l = np.arange(3).reshape(3, 1)
         c = np.arange(3)
         l, c, l + c
Out[19]: (array([[0],
                 [1],
                 [2]]),
          array([0, 1, 2]),
          array([[0, 1, 2],
                 [1, 2, 3],
                 [2, 3, 4]]))
```



Source : https://jakevdp.github.io/PythonDataScienceHandbook/

Fancy indexing

```
In [1]: import numpy as np
```

```
In [2]: v = np.random.randint(100, size=10)
        v
```

```
Out[2]: array([51, 59, 52, 86, 26, 19, 29, 60, 89, 11])
```

```
In [3]: ind = [3, 4, 7]
        v[ind]
```

```
Out[3]: array([86, 26, 60])
```

```
In [4]: m = np.random.randint(100, size=16).reshape(4, 4)
        m
```

```
Out[4]: array([[30, 42, 13, 37],
               [66, 30,  1, 67],
               [33, 39,  4, 24],
               [39, 77, 28, 13]])
```

```
In [5]: icols = [0, 3]
        m[:, icols]
```

```
Out[5]: array([[30, 37],
               [66, 67],
               [33, 24],
               [39, 13]])
```

```
In [6]: irows = [0, 1]
        m[irows, icols]
```

```
Out[6]: array([30, 67])
```

```
In [7]: irows = np.array([0, 1]).reshape(2, 1)
        irows
```

```
Out[7]: array([[0],
               [1]])
```

```
In [8]: m[irows, icols]
```

```
Out[8]: array([[30, 37],
               [66, 67]])
```

Les règles du broadcasting s'appliquent.
Le format des indices définit le format du tableau produit.

Exercice

5) Créer la matrice ci-contre en multipliant les vecteurs (1,2) et (3,4) entre eux.

3	4
6	8

6) Créer une matrice 3x3 avec des entiers aléatoires puis soustrayez de chaque élément la moyenne de sa ligne.

7) Créer une matrice 5x5 de 1 avec des 0 sur le pourtour.

- méthode 1 : avec fancy indexing

- méthode 2 : avec np.pad

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Masking

```
In [1]: import numpy as np

In [2]: m = np.arange(9).reshape(3, 3)
        print(m)

[[0 1 2]
 [3 4 5]
 [6 7 8]]

In [3]: m > 3

Out[3]: array([[False, False, False],
               [False,  True,  True],
               [ True,  True,  True]], dtype=bool)

In [4]: (m > 5) | (m < 2)

Out[4]: array([[ True,  True, False],
               [False, False, False],
               [ True,  True,  True]], dtype=bool)

In [5]: selection = (m > 5) | (m < 2)
        m[selection] = -1       # ou directement: m[(m > 5)|(m < 2)] = -1
        print(m)

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

Grilles

[2]:

```
# meshgrid pour créer des grilles 2D
x = np.arange(0, 5)
y = np.arange(0, 3)
X, Y = np.meshgrid(x, y)
print(X, X.shape)
print(Y, Y.shape)
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]] (3, 5)
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]] (3, 5)
```

[3]:

```
# mgrid pour crééer des grilles à N dimensions
Y, X = np.mgrid[0:3, 0:5]
print(X, X.shape)
print(Y, Y.shape)
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]] (3, 5)
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]] (3, 5)
```

A partir des matrices de coordonnées X et Y on peut par exemple calculer les valeurs d'une fonction sur la grille de points 2D.

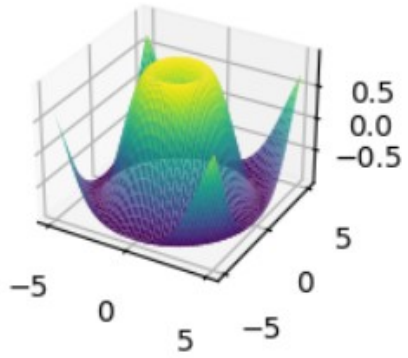
[4]:

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

Qu'on peut ensuite éventuellement utiliser pour tracer un graph 2D ou 3D

[5]:

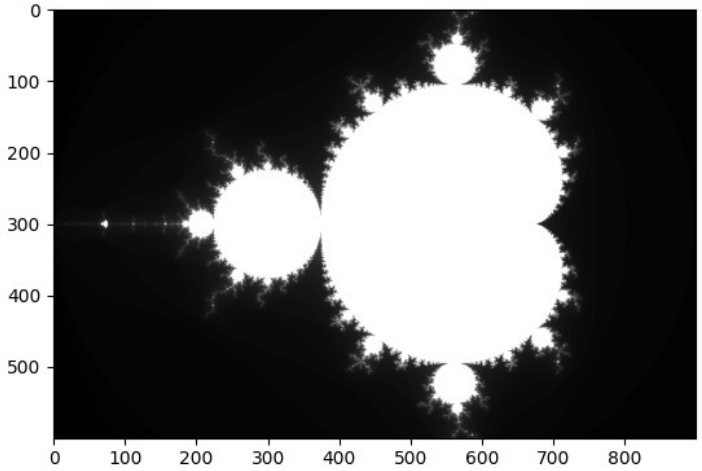
```
fig = plt.figure(figsize=(2,2))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
plt.show()
```



Exercice

Fractale de Mandelbrot

- Convergence de la suite de nombres complexes $z_{n+1} = z_n^2 + c$ $(z_0=0)$ selon la valeur du complexe c.
- Représentation graphique binaire (ci-contre) :
  - En x : partie réelle de c (de -2.0 à +1.0), N valeurs
  - En y : partie imaginaire de c (de -1.0 à +1.0), M valeurs
  - Blanc : la suite est bornée
  - Noir : la suite diverge
- Pour tester la divergence pour une valeur c donnée :
  - On itère pour calculer z_n
  - S'il existe n < 100 tel que $|z_n| > 2$, on déclare la suite divergente, sinon bornée



➔ Implémenter le calcul de cette fractale avec numpy

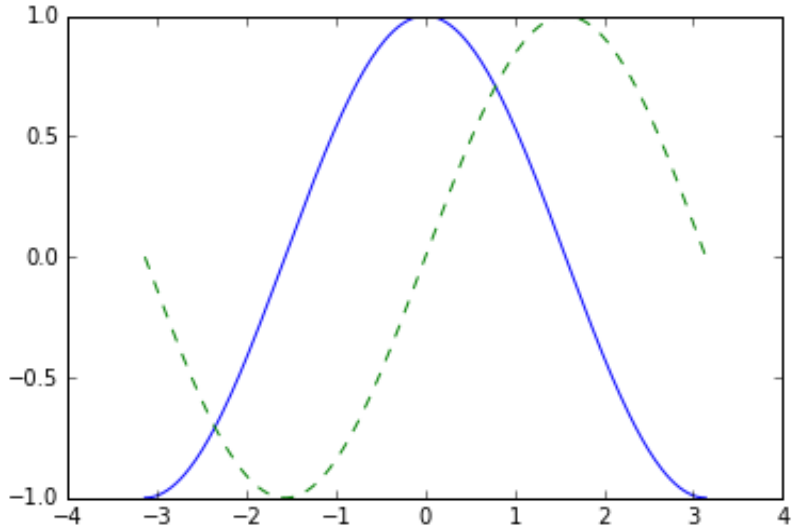
- Représenter les valeurs possibles de c sous forme d'une matrice de nombres complexes (utiliser une grille) de taille NxM (prendre par exemple N=900 et M=600) => matrice C
- Définir une matrice de Zn initialement nuls de taille NxM => matrice Z
- Utiliser le masking pour définir les pixels qui divergent (false) ou pas (true) => matrice D
- Vous verrez que lors du calcul de Z certaines valeurs sont en overflow (cas des points en divergence rapide). Trouvez une solution (utiliser le masking).
- Une seule boucle for (nombre d'itérations n)
- Représenter graphiquement la matrice D avec matplolib (plt.imshow)

➔ Pour les plus rapides, calculer une représentation « en niveaux de gris » (ou de couleurs) le niveau de gris / couleur étant associé au nombre d'itérations n qu'il a fallu pour que $|z_n| > 2$ (représente la rapidité de la divergence)

- Indication : introduire une autre matrice numpy contenant les nombres d'itérations

Tracer un graph

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C = np.cos(X)
S = np.sin(X)

plt.plot(X, C)
plt.plot(X, S, color="green", linewidth=1.0, linestyle="--")

plt.show()
```

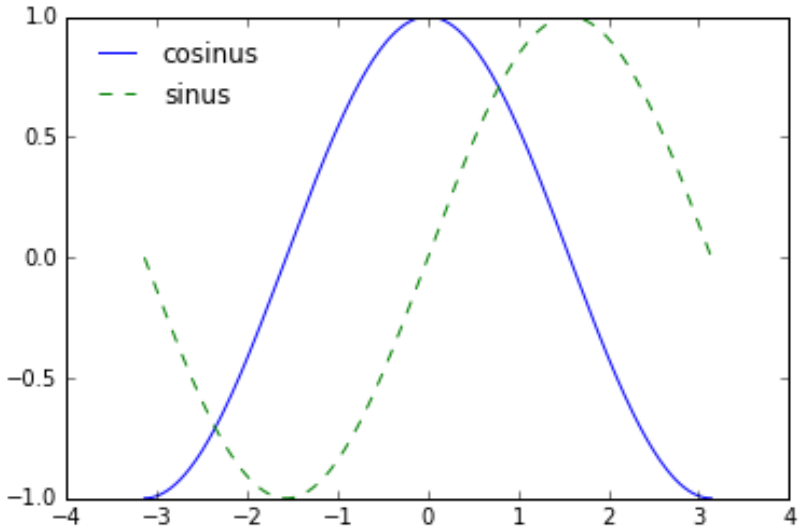


Tutoriel de Nicolas P. Rougier

https://github.com/rougier/matplotlib-tutorial

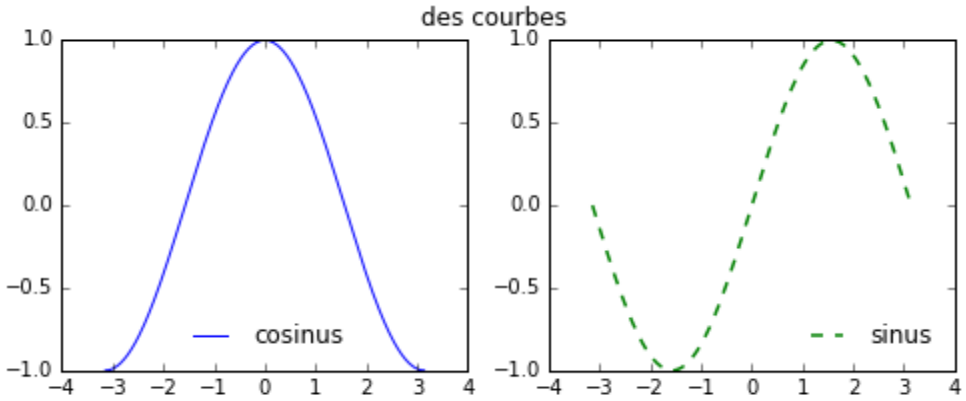
Ajouter une légende

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C = np.cos(X)
S = np.sin(X)

plt.plot(X, C, label="cosinus")
plt.plot(X, S, color="green", linewidth=1.0, linestyle="--", label="sinus")

plt.legend(loc='upper left', frameon=False)

plt.show()
```



Subplots

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C = np.cos(X)
S = np.sin(X)

figure = plt.figure(figsize=(8,3)) # l'argument figsize est optionnel
figure.suptitle('des courbes', size=12)

plot1 = figure.add_subplot(1, 2, 1) # 1 ligne, 2 colonnes, 1er élément
plot1.plot(X, C, label="cosinus")
plot1.legend(loc="lower center", frameon=False)

plot2 = figure.add_subplot(1, 2, 2) # 1 ligne, 2 colonnes, 2e élément
plot2.plot(X, S, color="green", linewidth=1.5, linestyle="--", label="sinus")
plot2.legend(loc="lower right", frameon=False)

plt.show()
```



Tip : Sauvegarder la figure dans un fichier image

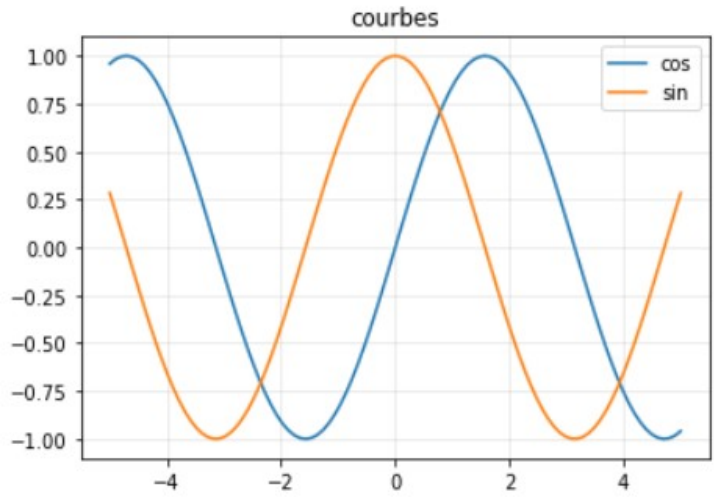
```
figure.savefig("figure.png")
```

Deux API

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

x = np.linspace(-5, 5, 100)
y1 = np.sin(x)
y2 = np.cos(x)
```

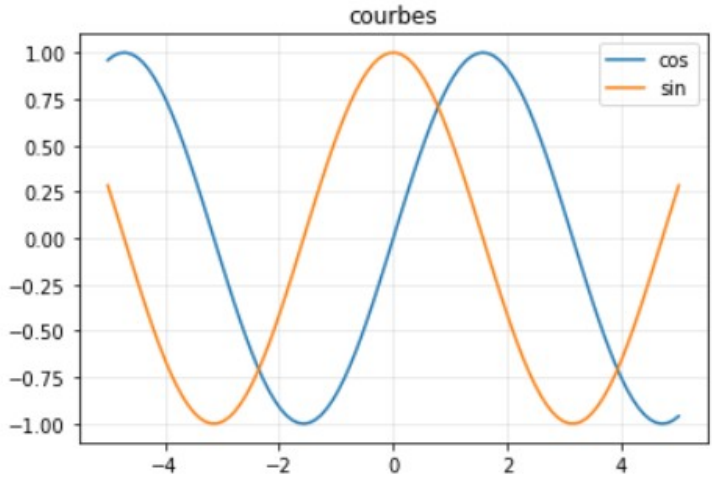
Matlab like

```python
plt.plot(x, y1, label="cos")
plt.plot(x, y2, label="sin")
plt.grid(True, alpha=0.3)
plt.legend()
plt.title("courbes")
plt.show()
```



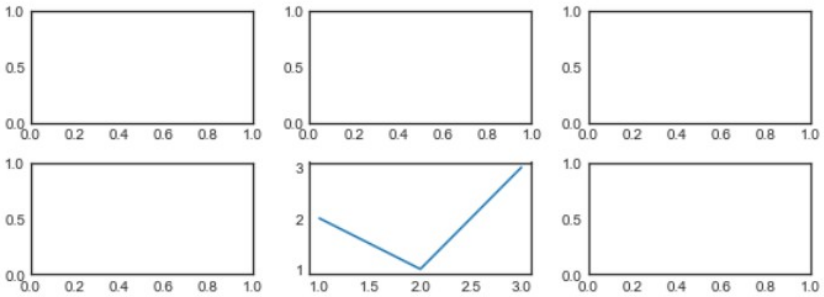
Object oriented

```python
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y1, label="cos")
ax.plot(x, y2,label="sin")
ax.grid(True, alpha=0.3)
ax.legend()
ax.set_title("courbes")
plt.show()
```

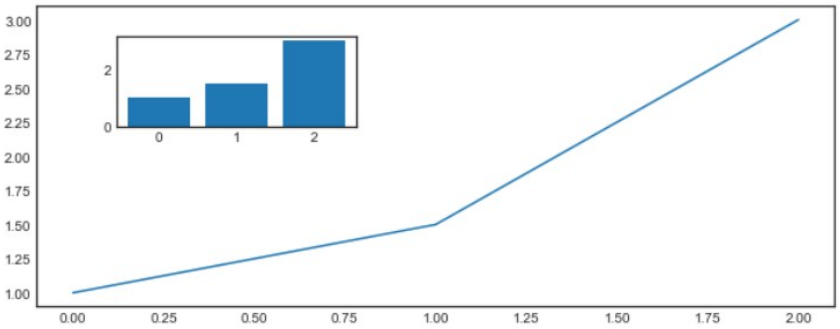


Subplots avancé

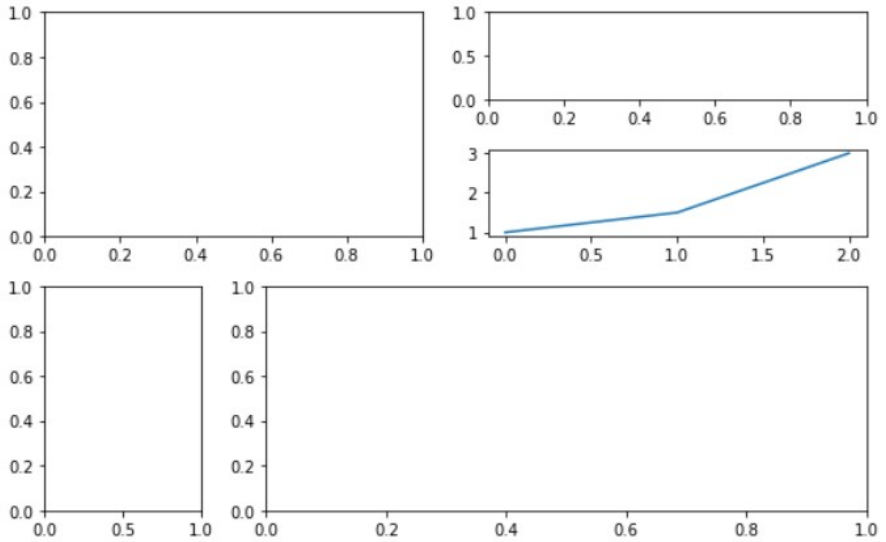
[4]:
```
fig, axes = plt.subplots(2, 3, figsize=(8, 3))
axes[1, 1].plot([1, 2, 3], [2, 1, 3])
plt.tight_layout()
plt.show()
```



[2]:
```
figure = plt.figure(figsize=(8,3))

# Arguments : gauche, bas, largeur, hauteur (0 à 1)
ax1 = figure.add_axes([0, 0, 1, 1])
ax2 = figure.add_axes([0.1, 0.6, 0.3, 0.3])

ax1.plot([0, 1, 2], [1, 1.5, 3])
ax2.bar([0, 1, 2], [1, 1.5, 3])

plt.show()
```



[3]:
```
import matplotlib.gridspec as gridspec

figure = plt.figure(figsize=(8, 5))
spec = gridspec.GridSpec(4, 4)

ax1 = figure.add_subplot(spec[:2, :2])
ax2 = figure.add_subplot(spec[0, 2:])
ax3 = figure.add_subplot(spec[1, 2:])
ax4 = figure.add_subplot(spec[2:, 0])
ax5 = figure.add_subplot(spec[2:, 1:])

ax3.plot([0, 1, 2], [1, 1.5, 3])

spec.tight_layout(figure)
```

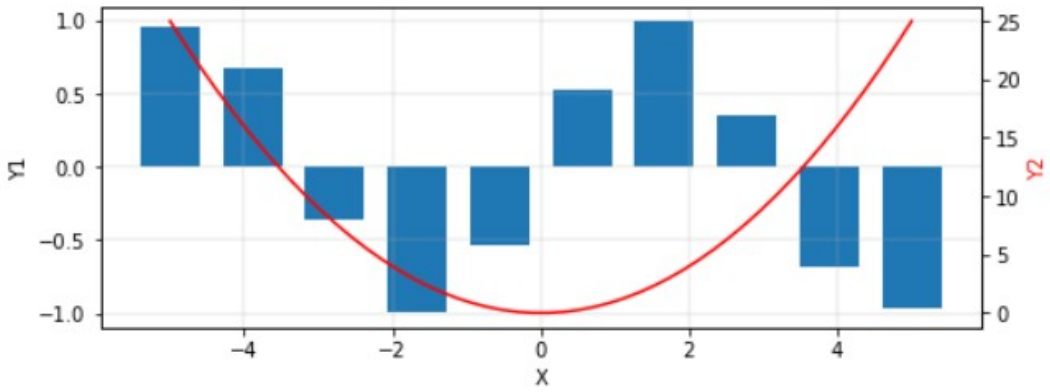


Axe secondaire

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

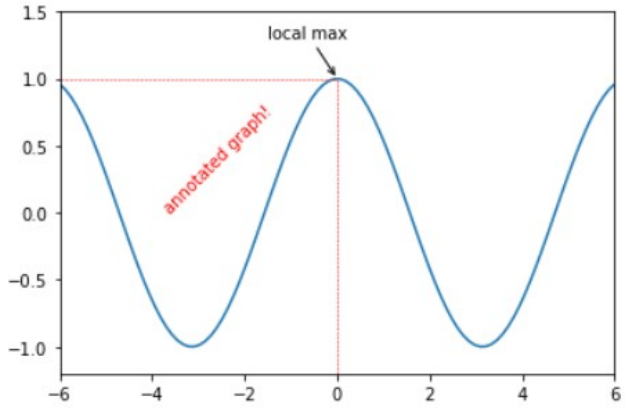
```
In [2]: x1 = np.linspace(-5, 5, 10)
        y1 = np.sin(x1)
        x2 = np.linspace(-5, 5, 50)
        y2 = x2**2
```

```
In [3]: figure = plt.figure(figsize=(8,3))
        ax1 = figure.add_subplot(1, 1, 1)
        ax2 = ax1.twinx()
        ax1.bar(x1, y1)
        ax2.plot(x2, y2, "-r")
        ax1.set_xlabel('X')
        ax1.set_ylabel('Y1')
        ax2.set_ylabel('Y2', color='r')
        ax1.grid(visible=True, axis="both", linewidth=0.3)
```



Annotations

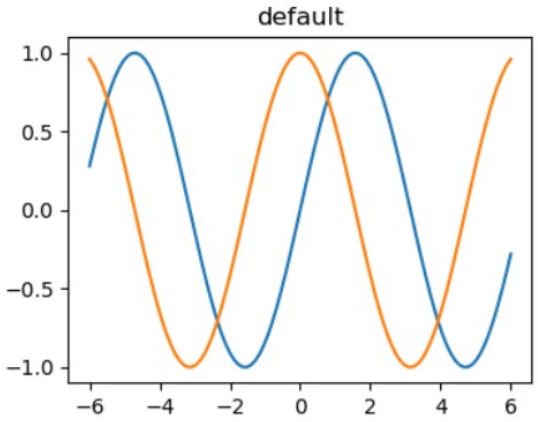
```python
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = np.linspace(-6, 6, 100)
y = np.cos(x)
ax.plot(x, y)

ax.set_xlim([-6, 6])
ax.set_ylim([-1.2, 1.5])
ax.vlines(x=0, ymin=-1.2, ymax=1, color="r", lw=0.5, ls='--')
ax.hlines(y=1, xmin=-6, xmax=0, color="r", lw=0.5, ls='--')

ax.text(-3.8, 0, 'annotated graph!', color="r", rotation=45)
ax.annotate('local max', xy=(0, 1), xytext=(-1.5, 1.3), arrowprops=dict(arrowstyle="->", connectionstyle="arc3"))

plt.show()
```

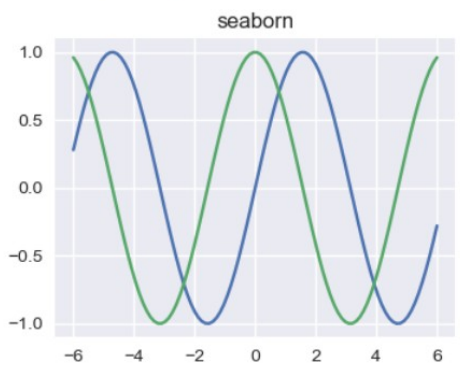


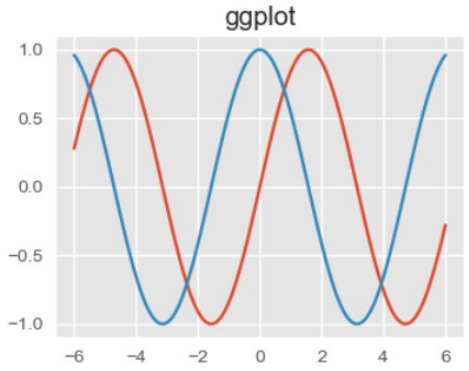
https://matplotlib.org/stable/tutorials/text/annotations.html
https://matplotlib.org/stable/gallery/text_labels_and_annotations/annotation_demo.html

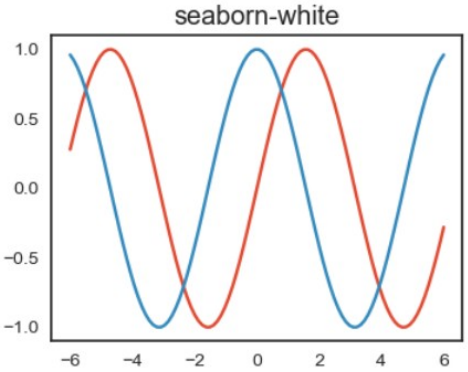
Styles

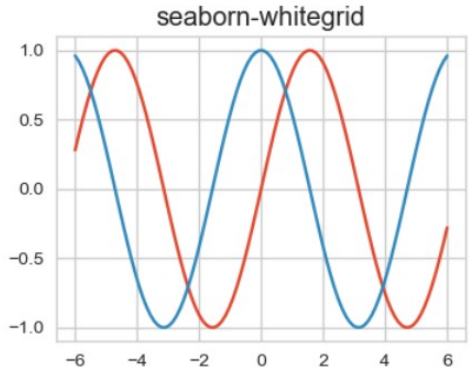
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        def draw_plot_with_style(style):
            plt.style.use(style)
            x = np.linspace(-6, 6, 100)
            y1 = np.sin(x)
            y2 = np.cos(x)
            fig, ax = plt.subplots(1, 1, figsize=(4, 3))
            ax.plot(x, y1)
            ax.plot(x, y2)
            ax.set_title(style)
            plt.show()

In [2]: draw_plot_with_style("default")
```





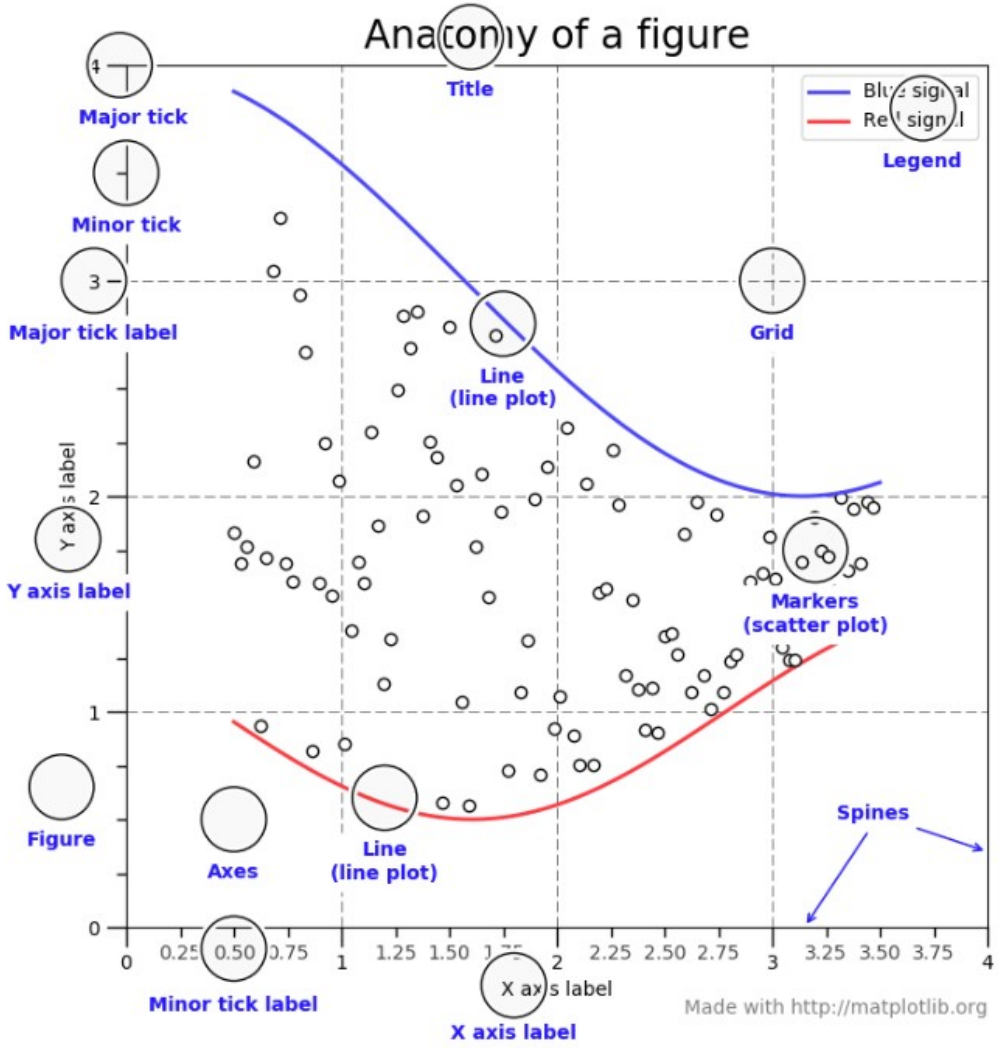






https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

Lexique



Traitement d'image

- http://www.scipy-lectures.org/advanced/image_processing/
- Une image est représentée par un tableau 2D (image monochrome) ou 3D (image multispectrale) de numpy (classe ndarray)
- Scipy comprend des fonctions de traitement d'image
  - Input/Output, displaying images
  - Basic manipulations: cropping, flipping, rotating, ...
  - Image filtering: denoising, sharpening
  - Image segmentation: labeling pixels corresponding to different objects
  - Classification
  - Feature extraction
  - Registration
- Module de base : ndimage
- Module avancé : scikit-image
  - http://scikit-image.org/
  - http://scikit-image.org/docs/stable/auto_examples/index.html
- Matplotlib permet d'afficher une image avec imshow()

Traitement d'image

```
In [2]: import numpy as np
        import scipy.misc
        import matplotlib.pyplot as plt
```

```
In [3]: image = scipy.misc.face(gray=True)  # image prédéfinie dans scipy
        plt.imshow(image, cmap=plt.cm.gray) # affichage en niveaux de gris
```

```
Out[3]: <matplotlib.image.AxesImage at 0x1b1597fed68>
```


```
In [4]: type(image), image.dtype
```

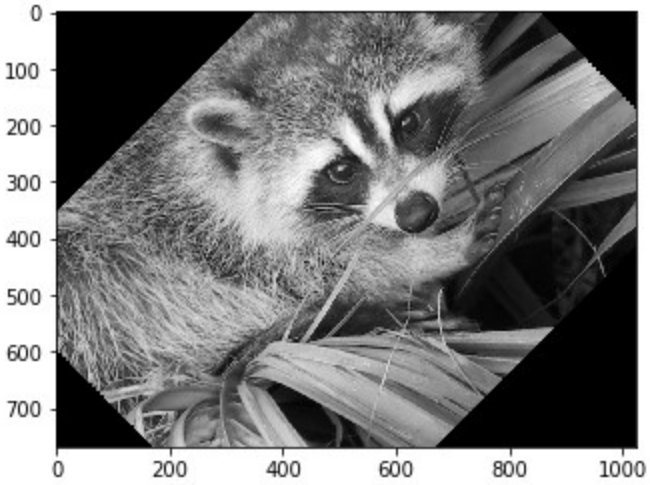
```
Out[4]: (numpy.ndarray, dtype('uint8'))
```

```
In [5]: image.shape  # dimensions de l'image (noir & blanc => 2D)
```

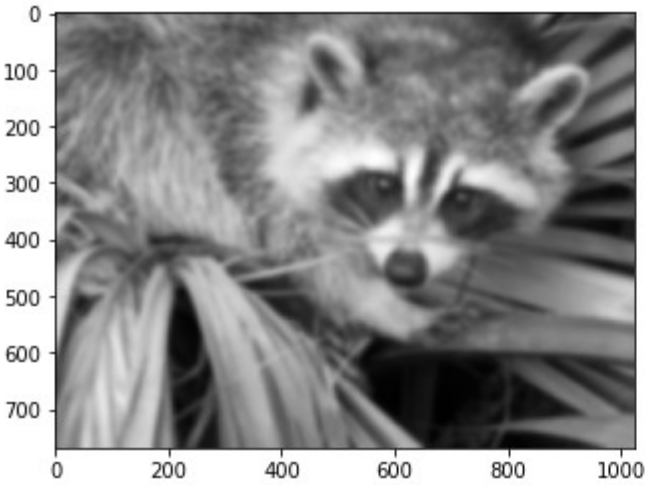
```
Out[5]: (768, 1024)
```

Traitement d'image

```
In [3]: from scipy import ndimage

In [4]: rotate_image = ndimage.rotate(image, 45, reshape=False)
        plt.imshow(rotate_image, cmap="gray")
Out[4]: <matplotlib.image.AxesImage at 0x230cd8341
```




```
In [5]: blurred_image = ndimage.gaussian_filter(image, sigma=5)
        plt.imshow(blurred_image, cmap="gray")
Out[5]: <matplotlib.image.AxesImage at 0x230cd905ac8>
```

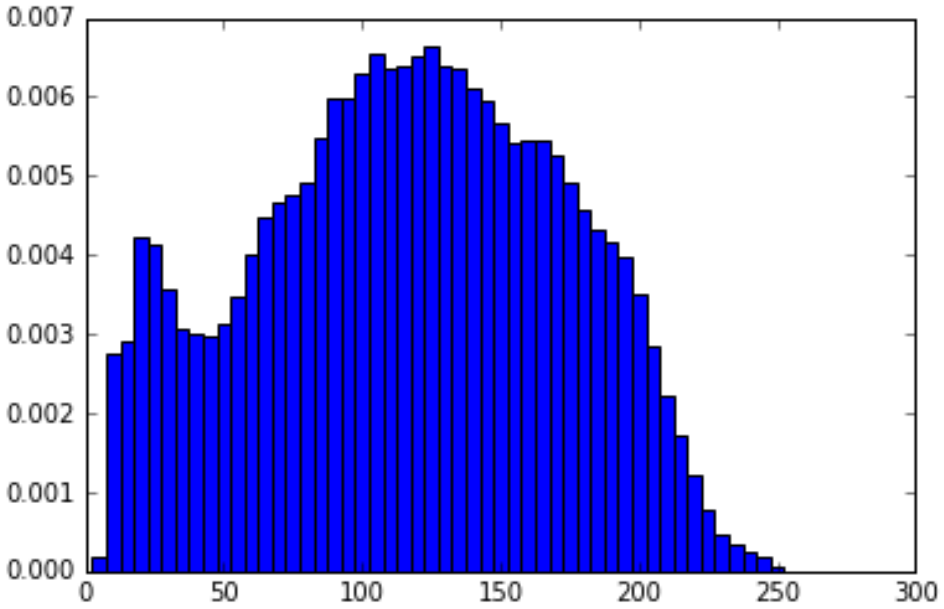



```
In [6]: import imageio

In [7]: imageio.imwrite('image.png', blurred_image)      # sauvegarder image

In [8]: loaded_image = imageio.imread('image.png')       # charger image
```

Traitement d'image

- Histogramme

```
In [18]: n, bins = np.histogram(image, bins=50, normed=True)

In [29]: plt.bar(0.5*(bins[1:] + bins[:-1]), n, width=bins[1]-bins[0])

Out[29]: <Container object of 50 artists>
```



Exercice 1

- Charger une image couleur du disque dur
- Redimensionner cette image à 500 colonnes, en conservant les proportions (ndimage.zoom)
- Afficher l'image couleur avec matplotlib (*)
- Transformer l'image en monochrome : NB = 0.21*R + 0.72*G + 0.07*B
- Afficher l'image monochrome avec matplotlib(*)
- Transformer l'image en image pixellisée, par un moyennage 5x5 pixels
- Afficher l'image pixelisée avec matplotlib(*)
- Mettre un cadre noir de 10 pixels de largeur autour de l'image
- Afficher l'image encadrée avec matplotlib(*)

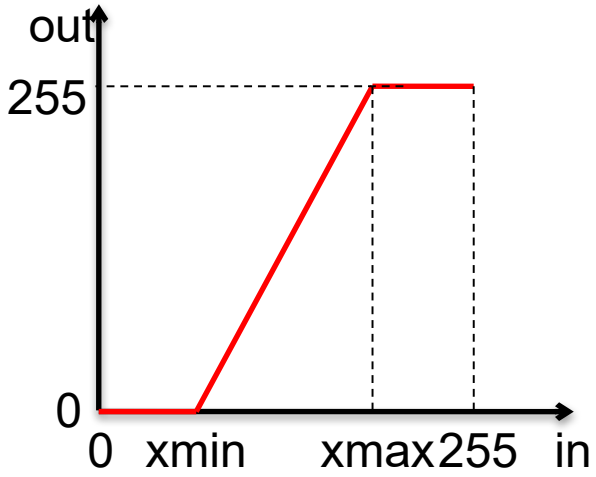
(*) Les 4 images doivent être affichées en disposition 2x2 (subplots)

Exercice 1

Exercice 2

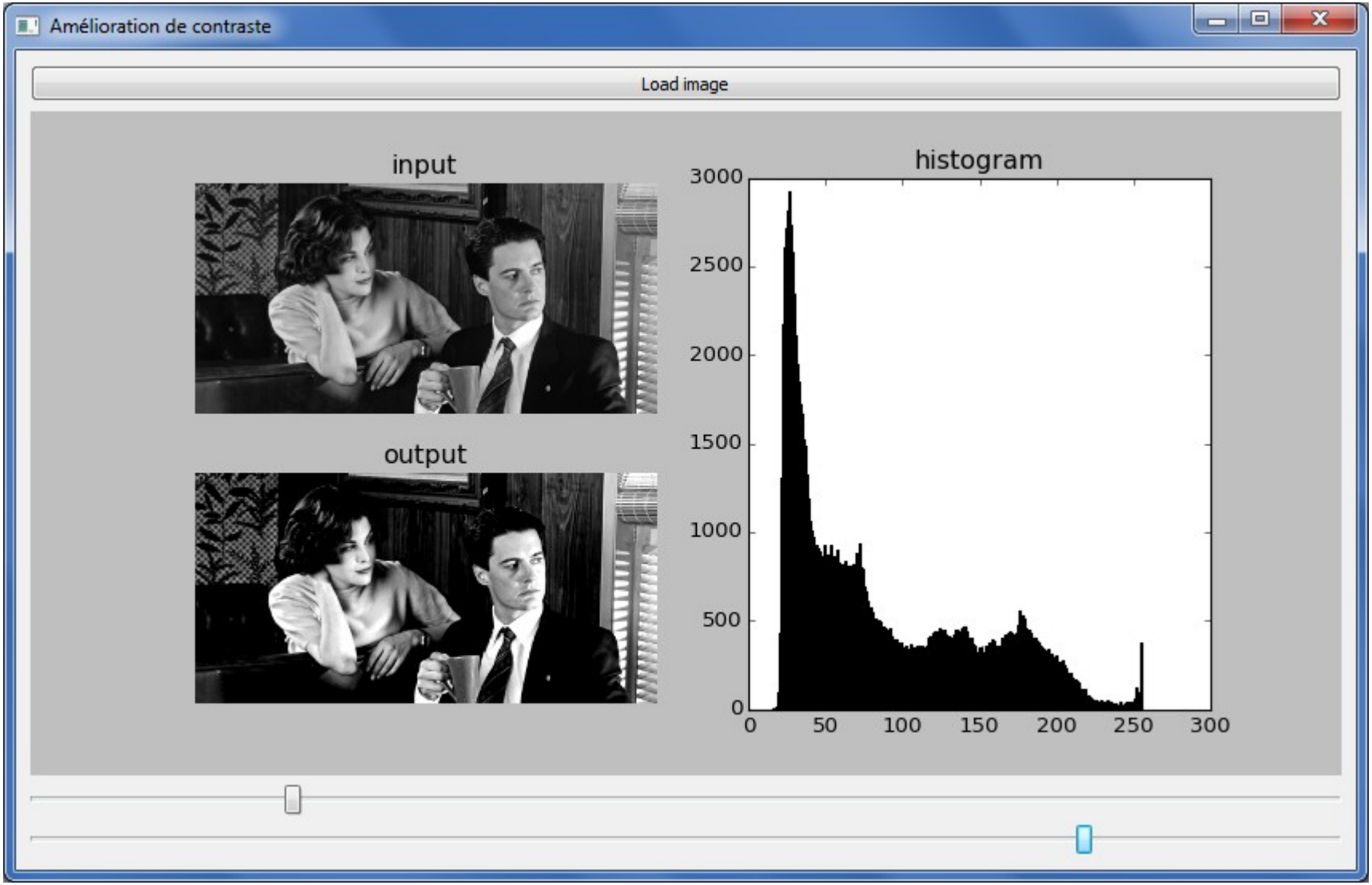
- Créer une classe permettant de :
  - Charger une image et la transformer en monochrome si c'est une image couleur
  - Appliquer sur cette image une loi linéaire d'augmentation de contraste définie ci-contre, où xmin et xmax sont des paramètres de cette loi.

  Tester en affichant les images côte-à-côte avec Matplotlib.



- Ajouter une méthode pour calculer automatiquement xmin et xmax de la loi précédente, tels que Pmin% des pixels soient < xmin et Pmax% des pixels soient > xmax. Pmin et Pmax sont des paramètres du programme.
  - Indice : utiliser l'histogramme cumulé de l'image et utiliser np.where()

Exercice 2



OpenCV

Dernière version
en date : 4.8 (2023)

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision.

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

Installation

- Sous Windows, avec WinPython

  Dans une invite de commande WinPython :

  ```
  pip install opencv-python
  ```

- Sous Windows ou macOS, avec Anaconda

  Dans une invite de commande Anaconda :

  ```
  conda install -c conda-forge opencv
  ```

- Sous Linux Ubuntu, avec apt

  Dans un terminal :

  ```
  sudo apt install python3-opencv
  ```

Documentation

- Site officiel :
  http://opencv.org

- Tutoriels OpenCV-Python :
  https://docs.opencv.org/4.5.5/d9/df8/tutorial_root.html

Exemple 1

- Utilisation de la webcam
  - OpenCV permet de récupérer les images d'une webcam avec la fonction VideoCapture()
  - VideoCapture(fichier) permet également de lire un fichier vidéo (avi, mp4, etc.)

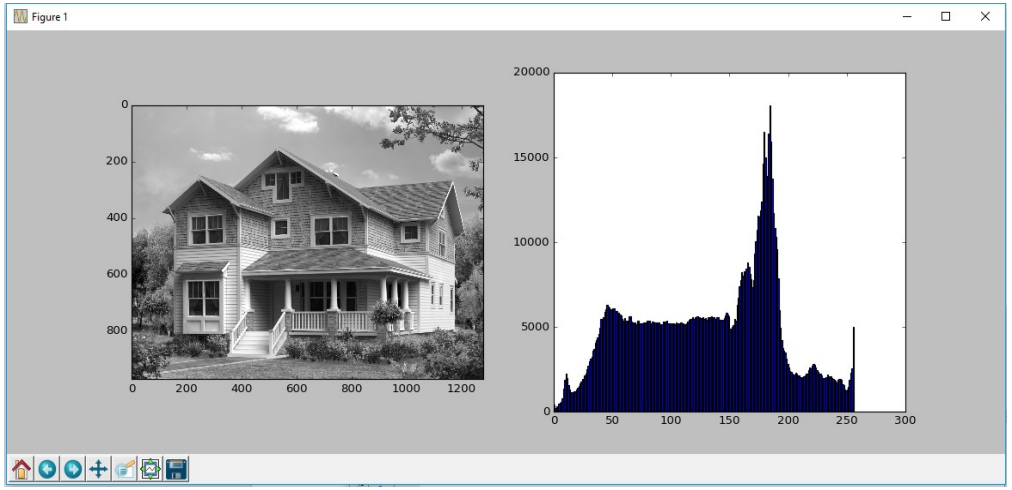
```
import cv2
cap = cv2.VideoCapture(0)  #0=numéro de webcam (si plusieurs : 0 ou 1 ou 2…)
# cap = cv2.VideoCapture('video.avi')  # pour lire un fichier vidéo
while True:
    ret, frame = cap.read()
    cv2.imshow("webcam avec opencv et python", frame)
    if cv2.waitKey(10) == ord("q"):
        break
cap.release()
cv2.destroyAllWindows()
```

  - cap.read() retourne une matrice au format Numpy (ndarray).
  - **On peut donc combiner l'utilisation de Numpy, Scipy, Matplotlib et OpenCV.**
  - waitKey(x) attend l'appui d'une touche pendant x ms.

Exemple 2

- Utilisation conjointe de OpenCV, Matplotlib, Numpy

```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('home.jpg', 0)
fig = plt.figure()
plt1 = fig.add_subplot(1, 2, 1)
plt1.imshow(img, cmap='gray')
plt2 = fig.add_subplot(1, 2, 2)
plt2.hist(img.ravel(), 256, [0,256])
plt.show()
cv2.waitKey(0)
```



Les fonctions de traitement d'image d'OpenCV sont généralement plus rapides que celle de Scipy, et donc plus adaptées au traitement d'un flux vidéo en temps réel.

Exercice

- Utiliser OpenCV pour lire les frames d'un fichier vidéo (exemple : mp4) et les afficher (avec imshow)
  - Indice : utiliser VideoCapture, s'inspirer de l'exemple avec webcam

- Utiliser OpenCV pour détecter les visages dans cette vidéo et dessiner un rectangle autour des visages détectés
  - Indice : utiliser la classe CascadeClassifier et le détecteur de visages Haar Cascade pré-entraîné fourni par OpenCV