12/09/2018

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-get : **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 ]])
```

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 :

Propriétés d'un ndarray

- Accéder à un élément d'un tableau : opérateur [ ]

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]
```

Pour en savoir plus :

Propriétés d'un ndarray

- Sélection et manipulation conditionnelle d'items :

```
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]]
```

Opérations basiques

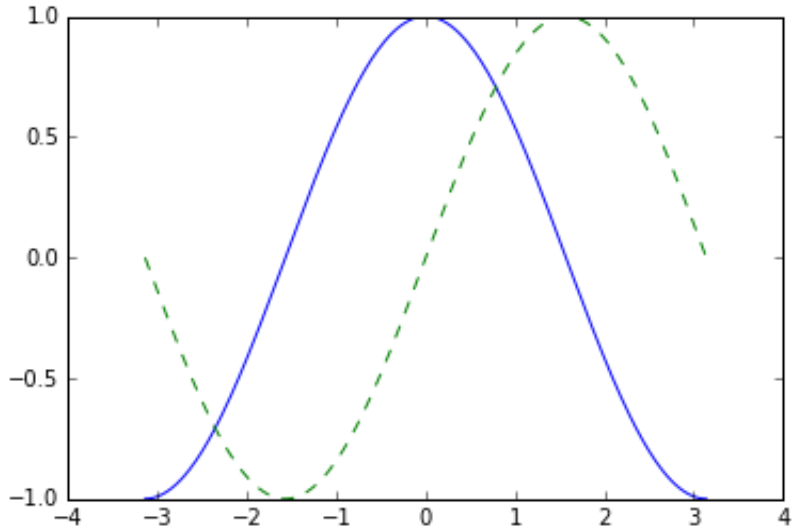
```
In [1]: import numpy as np

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
```

Pour en savoir plus :

```
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]]))
```

Tracer un graph

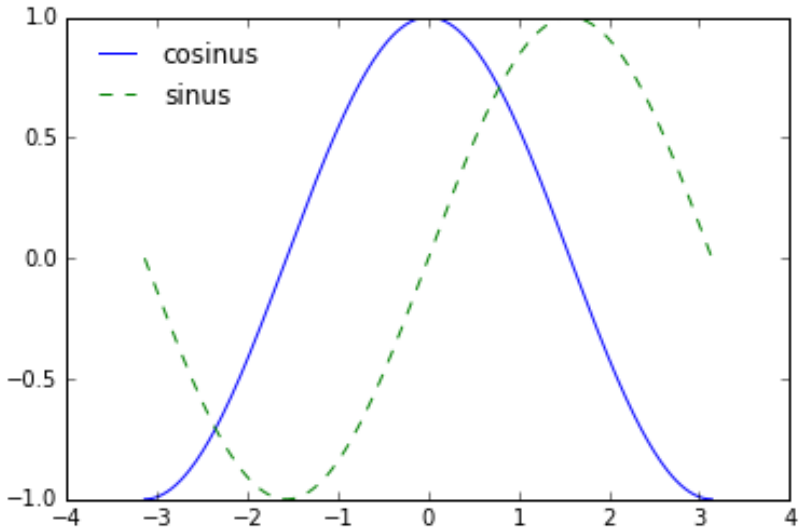
In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

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

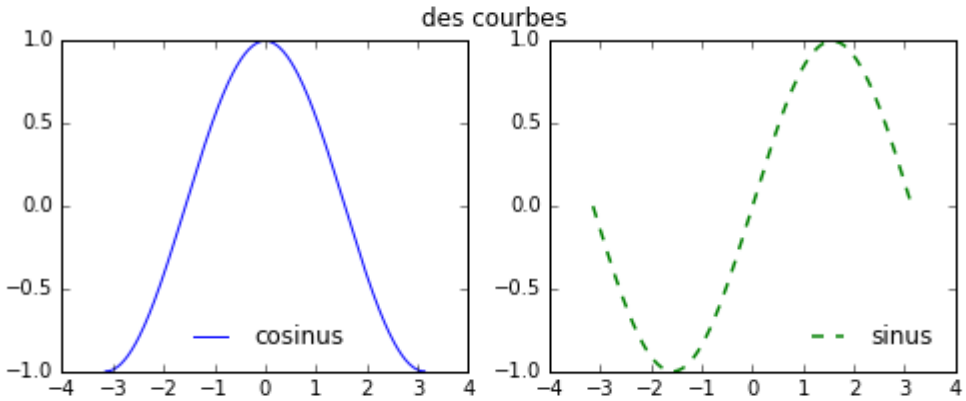
Ajouter une légende

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()
```



Matplotlib et Qt

In [21]:

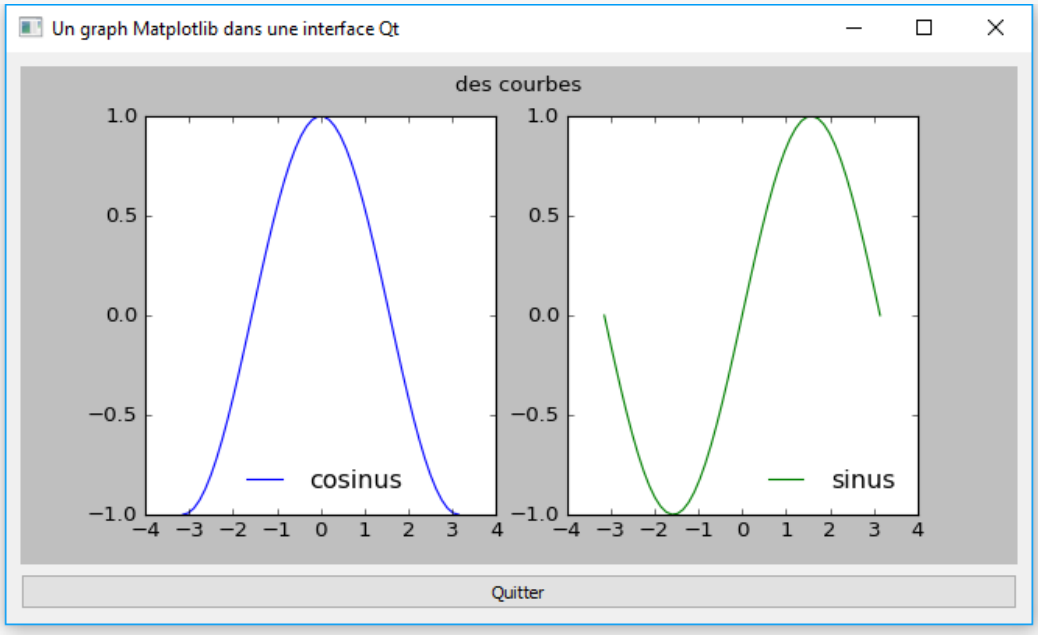
```
class MonGraph(QWidget):
    def __init__(self, parent):
        super().__init__(parent)

        self.figure = plt.figure()
        self.canvas = FigureCanvas(self.figure)

        self.bouton = QPushButton('Quitter')
        self.bouton.clicked.connect(self.quitter)

        layout = QVBoxLayout()
        layout.addWidget(self.canvas)
        layout.addWidget(self.bouton)
        self.setLayout(layout)

    def tracer(self):
        X = np.linspace(-np.pi, np.pi, endpoint=True)
        C, S = np.cos(X), np.sin(X)
        plot1 = self.figure.add_subplot(1, 2, 1)
        plot1.plot(X, C, label="cosinus")
        plot1.legend(loc="lower center", frameon=False)
        plot2 = self.figure.add_subplot(1, 2, 2)
        plot2.plot(X, S, color="green", label="sinus")
        plot2.legend(loc="lower right", frameon=False)
        self.figure.suptitle('des courbes', size=12)
        self.canvas.draw()

    def quitter(self):
        self.parent().close()
```

code identique à
une utilisation sans Qt

Matplotlib et Qt

```
from PyQt4.QtGui import *
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as FigureCanvas
import numpy as np
import matplotlib.pyplot as plt
```

```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Un graph Matplotlib dans une interface Qt')
        widget = MonGraph(self)
        widget.tracer()
        self.setCentralWidget(widget)

if __name__ == '__main__':
    app = QApplication([])
    fenetre = MaFenetre()
    fenetre.show()
    app.exec()
```



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
```

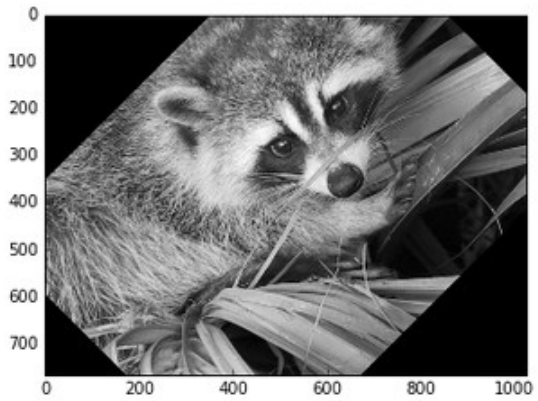
```
Out[4]: (numpy.ndarray, dtype('uint8'))
```

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

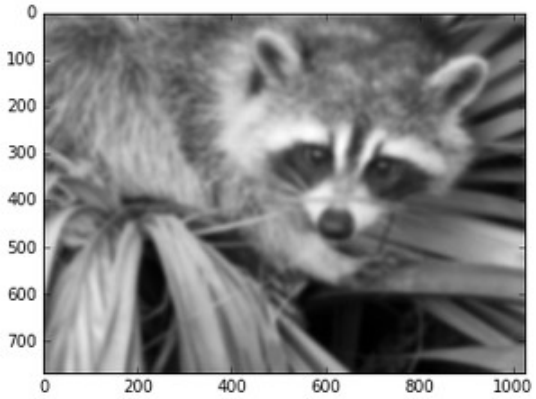
```
Out[5]: (768, 1024)
```

Traitement d'image

```
In [6]: from scipy import ndimage

In [7]: rotate_image = ndimage.rotate(image, 45, reshape=False)
        plt.imshow(rotate_image, cmap=plt.cm.gray)

Out[7]: <matplotlib.image.AxesImage at 0x1b159a5ccf8>
```




```
In [8]: blurred_image = ndimage.gaussian_filter(image, sigma=5)
        plt.imshow(blurred_image, cmap=plt.cm.gray)

Out[8]: <matplotlib.image.AxesImage at 0x1b15b91ca20>
```

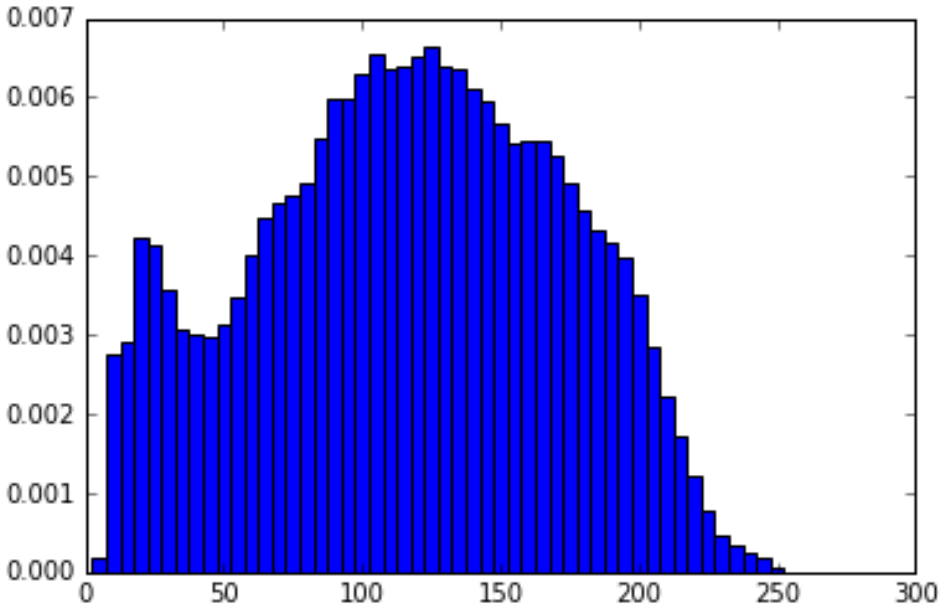



```
In [9]: scipy.misc.imsave('image.png', blurred_image)  # sauvegarder image

In [19]: image = scipy.misc.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 (imresize)
- 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
  - faire d'abord le traitement avec des boucles
  - puis utiliser une fonction de scipy (imresize)
  - comparer le temps de calcul des deux méthodes (en utilisant time.clock())
- 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(*)

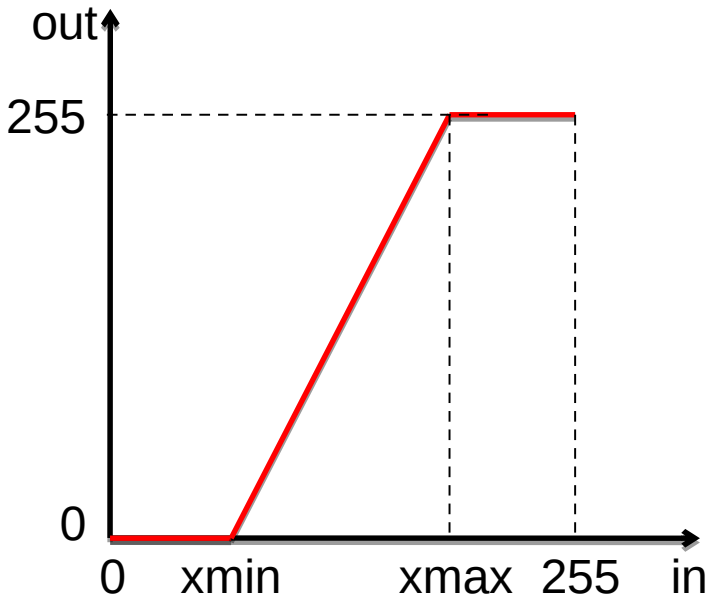
(*) Les 4 images doivent être affichées en disposition 2x2 (subplots)

Exercice 1

Exercice 2

1re partie

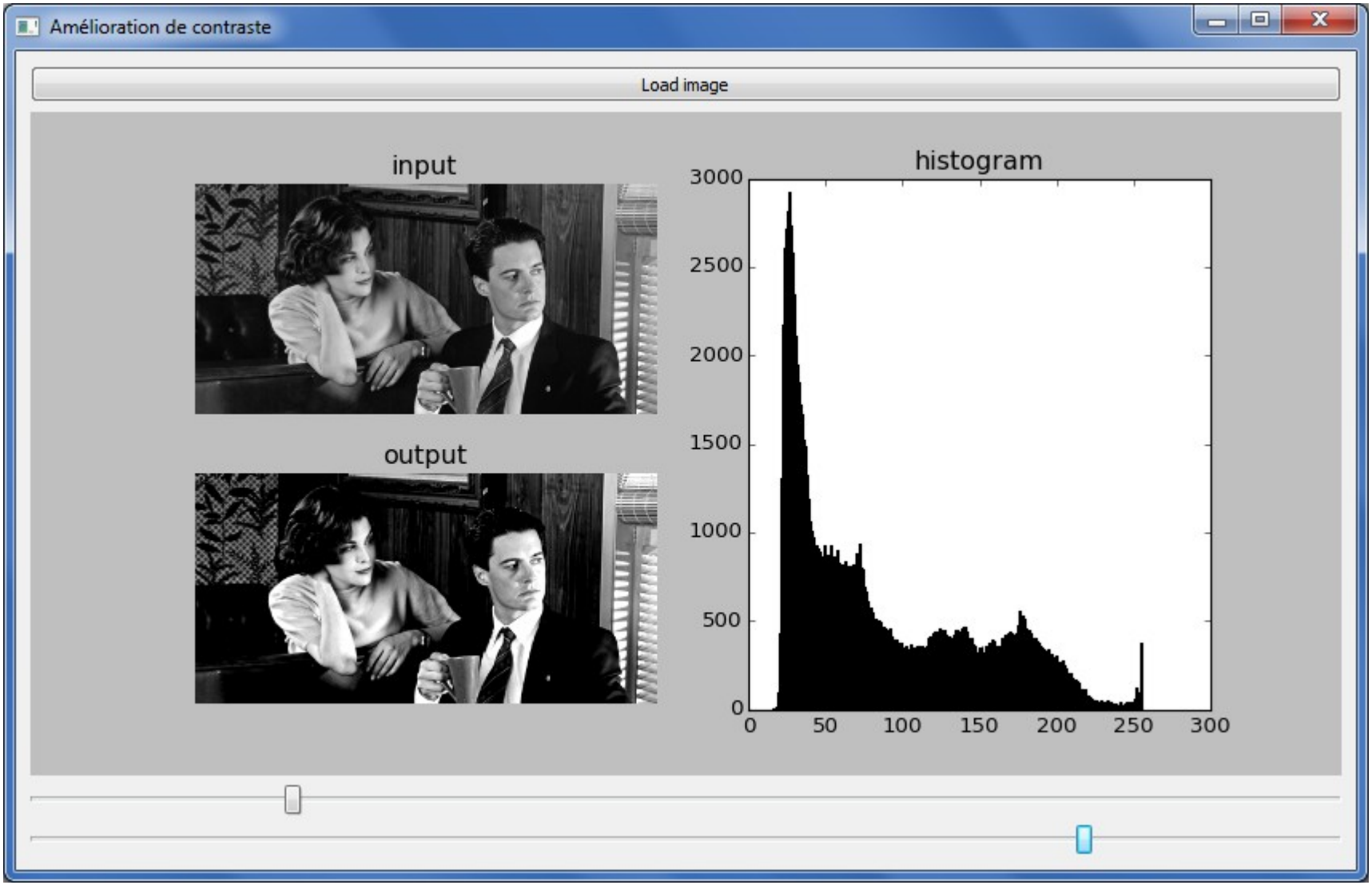
- Créer une classe permettant de :
  - Charger une image et la transformer en monochrome si c'est une image couleur
  - Appliquer sur cette image une loi linéaire d'augmentation de contraste définie ci-contre, où xmin et xmax sont des paramètres de cette loi.



2e partie

- Créer une IHM Qt permettant de :
  - Choisir et afficher une image
  - Afficher l'histogramme de cette image
  - Afficher l'image ayant subit une augmentation de contraste
- Ajouter dans l'IHM deux sliders permettant de définir les valeur xmin et xmax.
- Appliquer la loi d'augmentation de contraste « en temps réel », à chaque fois que la valeur d'un slider est modifiée.

Exercice 2



OpenCV

Dernière version
en date : 3.3 (08/2017)

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

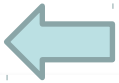
1) Télécharger le paquet au format wheel (.whl) sur le site suivant :

http://www.lfd.uci.edu/~gohlke/pythonlibs/

2) Choisir la version correspondant à votre version de Python :

OpenCV, a real time computer vision library.

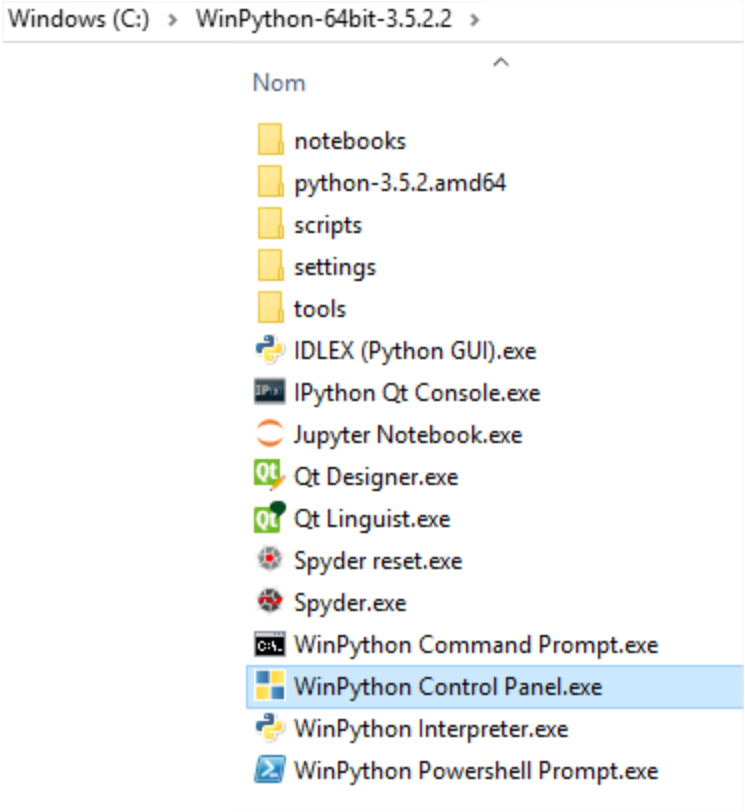
- opencv_python-2.4.13-cp27-cp27m-win32.whl
- opencv_python-2.4.13-cp27-cp27m-win_amd64.whl
- opencv_python-3.1.0+contrib_opencl-cp35-cp35m-win32.whl
- opencv_python-3.1.0+contrib_opencl-cp35-cp35m-win_amd64.whl
- opencv_python-3.1.0-cp27-cp27m-win32.whl
- opencv_python-3.1.0-cp27-cp27m-win_amd64.whl
- opencv_python-3.1.0-cp34-cp34m-win32.whl
- opencv_python-3.1.0-cp34-cp34m-win_amd64.whl
- opencv_python-3.1.0-cp35-cp35m-win32.whl
- opencv_python-3.1.0-cp35-cp35m-win_amd64.whl



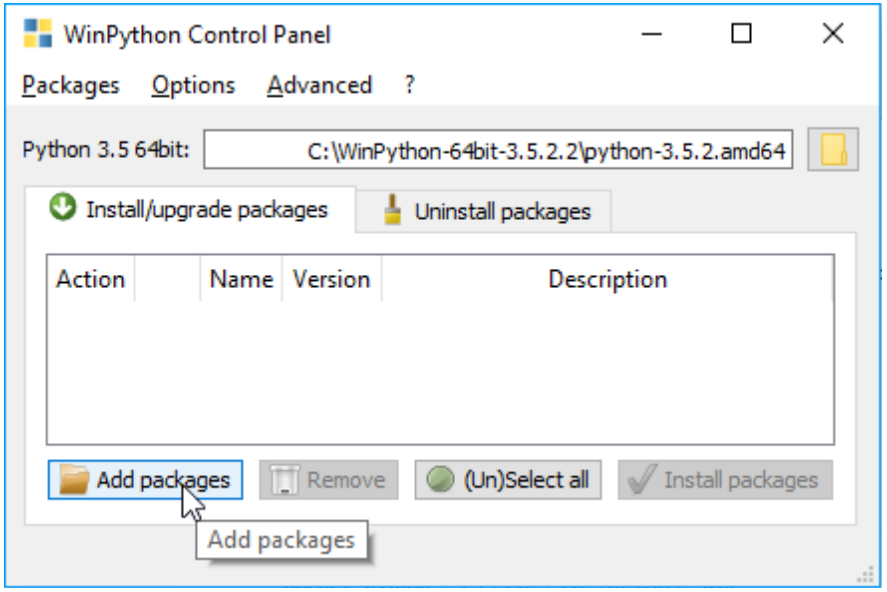
OpenCV 3.1 pour Python 3.5 Windows 64 bits

Installation

3) Installer ce paquet en passant par le « WinPython Control Panel » que vous trouverez dans le répertoire d'installation de WinPython :



Cliquer sur « Add packages » pour sélectionner et installer le paquet précédemment téléchargé :



Installation

- Sous Windows ou macOS, avec Anaconda

  Dans une fenêtre de command Anaconda :

  conda install -c conda-forge opencv

Documentation

- Site officiel : http://opencv.org

- Tutoriels OpenCV-Python : http://docs.opencv.org/3.3.0/d6/d00/tutorial_py_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 travailler sur un fichier vidéo (avi, mp4, etc.)

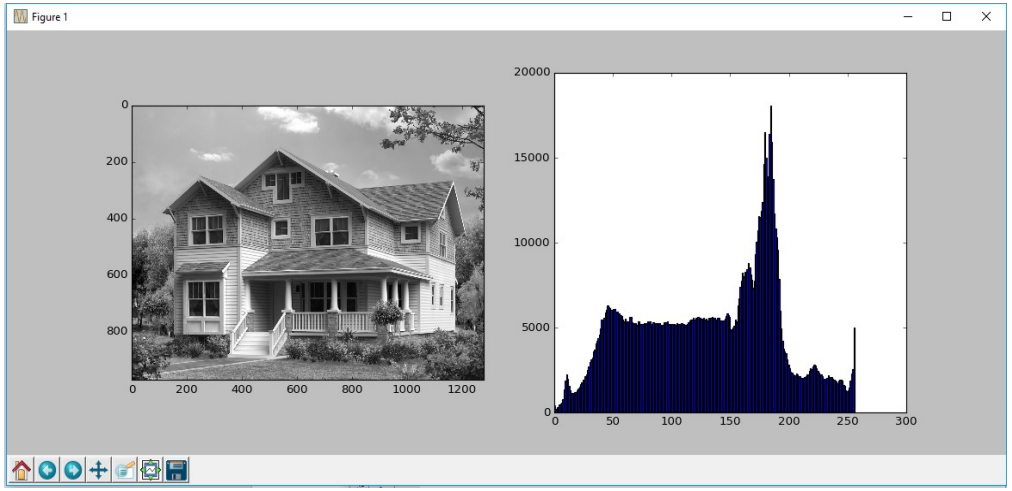
```
import cv2
cap = cv2.VideoCapture(0)  #0=numéro de webcam (si plusieurs : 0 ou 1 ou 2…)
# cap = cv2.VideoCapture('video.avi')  # pour lire un fichier vidéo
while True:
    ret, frame = cap.read()
    cv2.imshow('webcam avec opencv et python', frame)
    if cv2.waitKey(40) & 0xFF == ord('q'):
    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

Exemple 2

- Utilisation conjointe de OpenCV, Matplotlib, Numpy

```
import cv2
from matplotlib import 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 (beaucoup) plus rapides que celle de Scipy, et donc plus adaptées au traitement d'un flux vidéo en temps réel.