Scientific tools in Python : some basics for image processing
(Documentation complète : numpy (docs.scipy.org/doc/numpy/) ; scipy ndimage
(docs.scipy.org/doc/scipy/reference/ndimage.html) ; skimage (scikit-image.org/) ; matplolib (matplotlib.org/))

The generic numpy.array : vectors (1D arrays), images (2D-nD arrays), ...

```
#Import of « numpy » with name « np »
import numpy as np

#Instanciation
my_2Darray=np.array([ [ 1,2], [3,4]], dtype=np.uint8) #cf numpy for all types (np.int16 , np.float32, …), genericity versus type (at runtime)
my_1Darray=np.asarray([1,2,3])          #from standard list to array

#Instanciation: some utils
my_array=np.zeros((2,3))                #array of « zeros » (2 rows, 3 columns)
my_array=np.ones((2,3))                 #array of « ones » (2 rows, 3 columns)
my_array_1D=my_array_nD.flatten()       #1D from nD array
my_array_1D=np.arange(3,7,2)            #1D array [3,5], i.e. integers from 3, spaced by 2, and lower than 7 (similar to range for integers)

#Array attributes/properties and function
my_array.shape                          #array « shape » : rows (my_array.shape[0]) and columns (my_array.shape[1]) for 2D array
len(my_array)                           #function returning the number of rows (== number of points if 1D array)
my_array.size                           #array size (number of points contained in the array)
my_array.dtype                          #type of points stored in the array (i.e. np.uint8, np.float,.... )

#Data type modification
my_array=my_array.astype(np.uint8)      #or np.int16 np.float …

#Array shape modification
my_array=my_array.reshape(n,m)          #e.g. If my_array is a (2,3) array, then my_array.reshape(3,2) changes it to 3 rows and 2 colums

#Array concatenation (along rows and cols)
my_array=np.hstack((A,B,C))             #horizontal stack : « adding B and C cols to A » → leads to [A B C] ; nb rows A == nb rows B and C
my_array=np.vstack((A,B,C))             #vertical stack : « adding B and C rows to A » → leads to [A, B,C] ; nb cols A == nb cols B and C

#Accessors
my_array[i,j]=1                         #accessing row i and column j (example for a 2D array)
my_sub_array=my_array[2:4,4:10]         #accessing a sub part of an array by indices intervals ([2,4[  for rows and [4,10[ for columns)
my_sub_array=my_array[2:4, :]           #accessing a sub part of an array by indices intervals ([2,4[  for rows and all columns)
my_sub_array=my_array[2:10]             #case of a 1D array (vector), could be done for nD arrays

#Loop over an array (case of a 2D array) : first example (2 nested loops)
for i in range(0,my_array.shape[0]):
    for j in range(0,my_array.shape[1]):
        print(i , ",", j , " = " ,my_array[i,j])

#Loop over an array (case of a 2D array) : second example (one loop over a 1D copy of the initial nD array)
for i in my_array.flatten(): print(i)

#Loop over an array (case of a 2D array) : third example (one loop over rows)
for p in my_array: print("line: " , p)
```

Some operators on numpy.array

```
#Basic element-wise arithmetic operations
my_array=my_array1+my_array2 ; my_array=my_array1*my_array2 ; ….
#Element-wise power operator
my_array=my_array**n
#Some other useful functions
value=np.min(my_array)                  #Min value stored in the array
value=np.max(my_array)                  #Max value stored in the array
value=np.mean(my_array)                 #Mean value ( over all values stored in the array)
value=np.median(my_array)               #Median value (over all values stored in the array)
value=np.sum(my_array)                  #Sum of all values stored in the array
indice_of_max=np.argmax(my_array)       #Index corresponding to the max value (first encountered) appearing in the flattened nD array
indice_of_min=np.argmin(my_array)       #Index corresponding to the min value (first encountered) appearing in the flattened nD array
my_array=np.abs(my_array)               #Resulting array contains absolute values of the input array
my_array=np.round(my_array,n)           #Resulting array contains rounded values of the input array (with n digits)
#Inner product
C=np.dot(A,B)                           #C is the inner product of A and B (matrix) : the number of rows of B == number of cols of A
```

Jean-Baptiste Fasquel, ISTIA, Université d'Angers

Scientific tools in Python : some basics for image processing
(Documentation complète : numpy (docs.scipy.org/doc/numpy/) ; scipy ndimage (docs.scipy.org/doc/scipy/reference/ndimage.html) ; skimage (scikit-image.org/) ; matplolib (matplotlib.org/))

#Persistance of nD array
np.save("file.npy",my_array) AND my_array=np.load("file.npy")

Image I/O with Scipy (misc subpackage) : standard png, jpeg, … image file formats

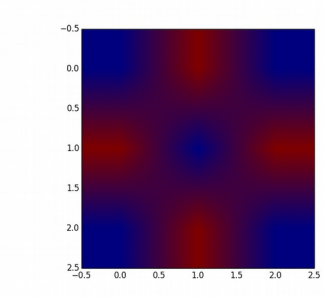
#Import of «scipy», and then import the subpackage « misc » of « scipy »
import scipy ; from scipy import misc
#2D numpy array from png
my_array=scipy.misc.imread(*"image.png"*)
#2D numpy array to png
scipy.misc.imsave(*"image.png"*,my_array)

Basic display with matplotlib

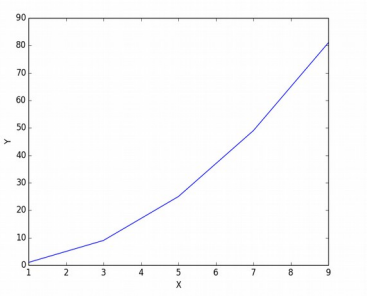
 # « numpy » import
import numpy as np
 # import matplotlib subpackage pyplot
import matplotlib.pyplot as plt

#2D array (example)
```
image=np.array([ [0,1,0],
                 [1,0,1],
                 [0,1,0]  ])
```

#Basic display of a 2D array
```
plt.imshow(image)    #Prepare image display
plt.show()           #Start displaying
```



Basic display of 2D array (image)

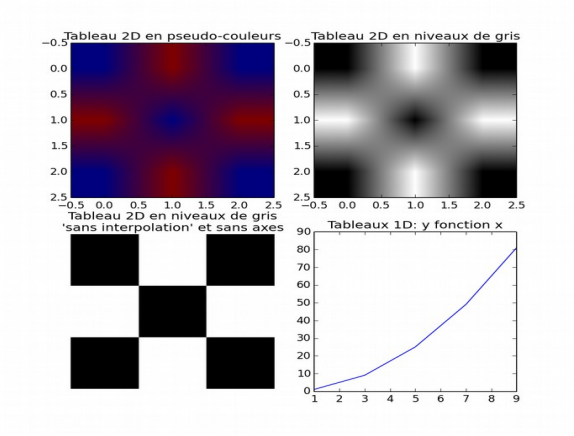


Basic display of « y=function(x) »

#1D Arrays : example of y as a function of x
```
x=np.array([1,3,5,7,9])
y=x**2               #x to the power of 2

plt.plot(x,y) #without x, abscissa are y indices
plt.show()
```

#Multi plot : plt.subplot(nmi)
```
#(n) defines the number of rows
#(n) defines the number of columns
#(i) defines the index (from 1 to nxm) of the current subplot
plt.subplot(221)
plt.imshow(image)
plt.subplot(222)
plt.imshow(image,"gray")
plt.subplot(223)
plt.imshow(image,"gray",interpolation="nearest")
plt.axis('off')
plt.subplot(224)
plt.plot(x,y)
plt.show()
```



Multi-display (plt.subplot(...))

Jean-Baptiste Fasquel, ISTIA, Université d'Angers

Scientific tools in Python : some basics for image processing
(Documentation complète : numpy (docs.scipy.org/doc/numpy/) ; scipy ndimage
(docs.scipy.org/doc/scipy/reference/ndimage.html) ; skimage (scikit-image.org/) ; matplolib (matplotlib.org/))

Some basic image processing functions from numpy

```
#Import
import numpy as np

#Thresholding (nD image/array)
thresholded_image=np.where(image > threshold, 255,0) #result: « 255 » where « image points » are > threshold, « 0 » otherwise

#Histogram (numpy histogram is generic (e.g. for floats)) : considered exemple focuses on an 2D integer image
image=np.array([[ 1, 2, 2],[3,1,2]])                                   # → image :[[ 1 , 2 , 2 ],
                                                                                  [ 3 , 1 , 2  ] ]
my_bins=np.arange(np.min(image),np.max(image)+2)            # → my_bins == [1,2,3,4]
histogram,intervals=np.histogram(image,bins=my_bins)        # → « my_bins » → occurrences in [1,2[ and [2,3[ and [3,4]
#Produce : intervals == [1,2,3,4], and histogram ==  [2,1,3]
#Intensity 1 (within interval [1,2[) ↔  2 points → histogram[0]==2
#Intensity 2 (within interval [2,3[) ↔  3 points → histogram[1]==3
#Intensity 3 (within interval [3,4]) ↔  1 points → histogram[2]==1
#Note : to remove the last elt of « intervals » (→ same size as histogram) → intervals=intervals[0:-1] #i.e. keeping all except the last « -1 »
```

Some basic image processing functions (neighborhood operators) from scipy.ndimage

```
#Import of scipy and its subpackage ndimage
import scipy as sp ; from scipy import ndimage

#Generic nD linear filtering (convolution et correlation) : exemple for a 2D array
filter=np.ones((3,3))  #Filter : 2D array
filter/=float(np.sum(filter))        #Normalization (optionnal)
result=sp.ndimage.convolve(image.astype(np.float), filter)  #Input to float for real numbers computations
result=sp.ndimage.correlate(image .astype(np.float), filter)  #Input to float for real numbers computations

#Predefined linear filtering: case of the « mean » filter
filter_size=3
result=sp.ndimage.filters.uniform_filter(image .astype(np.float), filter_size)

#Predefined linear filtering: case of the « sobel » filter (first derivative) – 2D image
direction=0                                  #0: along "x" (rows - vertical) ; 1: along "y" (columns - horizontal)
result=sp.ndimage.filters.sobel(image, direction)

#Predefined non-linear filtering: case of the « median » filter
size=3
result=sp.ndimage.filters.median_filter(image, size)

#Binary mathematical morphology (érosion, dilation, opening, closing)
structuring_element=np.ones((3,3))
result=sp.ndimage.morphology.binary_erosion(image,structuring_element)   #explicit structuring element
result=sp.ndimage.morphology.binary_erosion(image,iterations=1)          #number of «iterations » with the default square structure element
#idem for dilation (« binary_dilation »), opening (« binary_opening ») and closing (« binary_closing»):

#Connected components labeling
connectivity=np.ones((3,3))
result,nb_labels=sp.ndimage.measurements.label(image,connectivity)
#« 0 » in « result » identify background. Positive values > 0 correspond to labels components. « nb_labels » is the number of connected components

#Filling holes (based on binary mathematical morphology and connected components)
connectivity=np.ones((3,3))
resultat=scipy.ndimage.morphology.binary_fill_holes(image,connectivity)
```

Jean-Baptiste Fasquel, ISTIA, Université d'Angers

Scientific tools in Python : some basics for image processing
(Documentation complète : numpy (docs.scipy.org/doc/numpy/) ; scipy ndimage (docs.scipy.org/doc/scipy/reference/ndimage.html) ; skimage (scikit-image.org/) ; matplolib (matplotlib.org/))

Scikit-image : a third party package dedicated to image processing – some examples
(note : some functions hide (simplify) calls to scipy.ndimage algorithms)

#Imports
```
import skimage as sk
from skimage import exposure #for histogram in particular
from skimage import measure #for shape measurements in particular
```

#Histogram of integer images
```
histogram,gray_levels=sk.exposure.histogram(image)
```

#Automatic threshold determination (Otsu method) for thresholding purposes
```
import skimage as sk
from skimage import filter
threshold=sk.filter.threshold_otsu(image)
```

#Measure, for each connected composant (!=0), of some region properties (cf inlinge documentation for details)
```
measures=sk.measure.regionprops(image)
```

#Center of mass of the first connected component (0 index)
```
barycentre_x,barycentre_y=measures[0]['Centroid']
```

#Area of the first connected component (0 index)
```
surface=measures[0]['Area']
```