1 Python correction

```python
# display images
import matplotlib.pyplot as plt

# ndimage defines a few filters
from scipy import misc, ndimage

# numeric calculation
import numpy as np

# measure time
import time

# read and save images
import imageio
```

1.1 First manipulations

1.1.1 Open, write images

The following file loads the ascent image and display it. The print function is optional.

```python
# load file ascent
ascent = misc.ascent()

# load file cerveau.jpg
brain = imageio.imread('cerveau.jpg')
print(type(brain))
print(brain.shape, brain.dtype)

# save file
imageio.imwrite('test.png', brain)
```

1.1.2 Display images

You can modify to previous example to add the following lines:

```python
plt.imshow(ascent)
plt.show()
```

Notice that you have to close the image window to write commands again. Also, the colormap is not the good one by default (see Fig. 1).

```
plt.figure(figsize=(10, 3.6))

# first subplot
plt.subplot(131)
plt.imshow(ascent)

# second subplot
plt.subplot(132)
plt.imshow(ascent, cmap=plt.cm.gray)
plt.axis('off')

# third subplot (zoom)
plt.subplot(133)
plt.imshow(ascent[200:220, 200:220], cmap=plt.cm.gray, interpolation='
    ↪ nearest')

plt.subplots_adjust(wspace=0, hspace=0.,
                    top=0.99, bottom=0.01,
                    left=0.05, right=0.99)
plt.show()
```

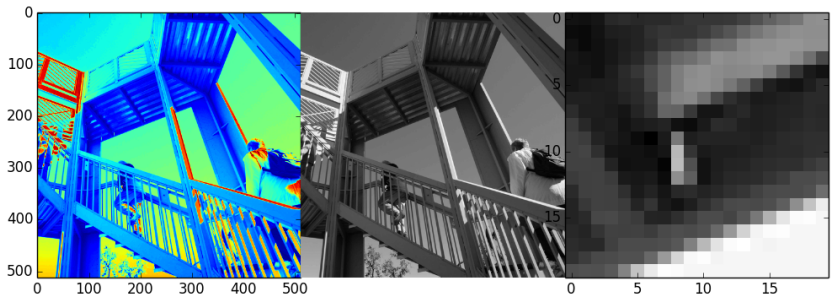


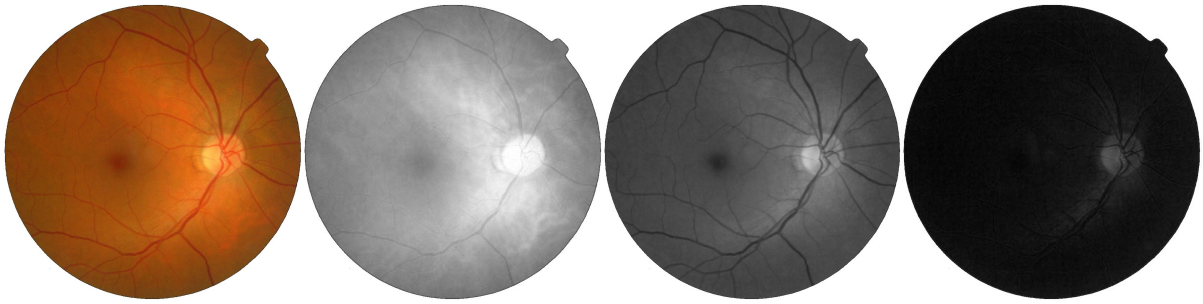
Figure 1: Displaying images with an adapted colormap.

1.1.3 Color channels

A color image is constituted of (generally) three channels. This representation follows the human visual perception principles: in the human retina, the sensitive cells (the cones) react to specific wavelength that correspond to red, green and blue. The sensors technology adopted the *same* characteristics and a so-called Bayer filter has 2 green filters for 1 red and 1 blue. Consequently, the green channel presents a better resolution than the other channels.

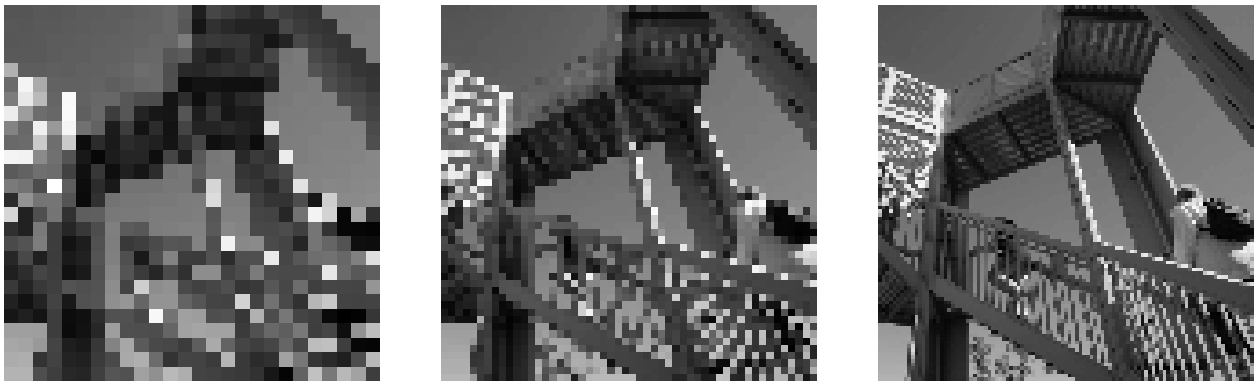
1.1.4 Image Resizing

The number of pixels is reduced by subsampling the image. Notice that there is no anti-aliasing filter applied to the image before reducing its size. The result is presented in Fig.3 with images of the same size, and in Fig.4 with images at the same resolution.



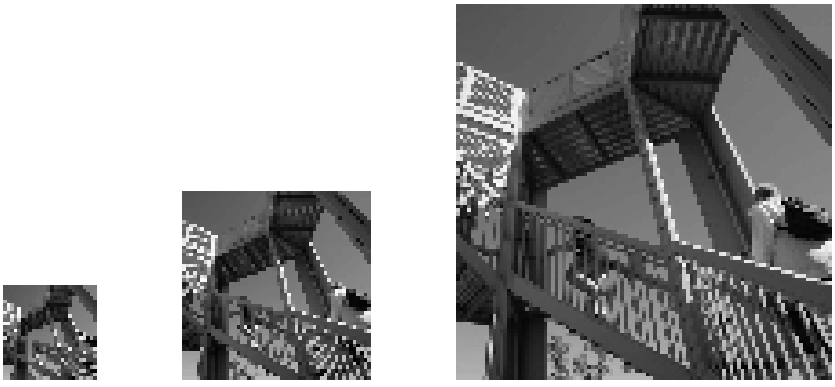
(a) Color image. (b) Red channel. (c) Green channel (d) Blue channel.

Figure 2: The green channel presents the best contrasts in the case of retina images.



(a) Scale divided by 20. (b) Scale divided by 10. (c) Scale divided by 5.

Figure 3: Reduction of the scale of the image on each axis, showing a so-called *pixellisation* effect. Represented at the same size, the density of pixels is thus reduced.



(a) (b) (c)

Figure 4: At a constant resolution, images with different definitions are represented with different sizes.

1.1.5 Color quantization

The following code uses the properties of integer operations to round values to the nearest integer. Illustration is presented Fig. 5.

```
q4 = ascent // 4*4;
q16= ascent //16*16;
q32= ascent //32*32;
```

(a) q4. (b) q16. (c) q32.

Figure 5: Reduction of the number of gray levels (quantization).

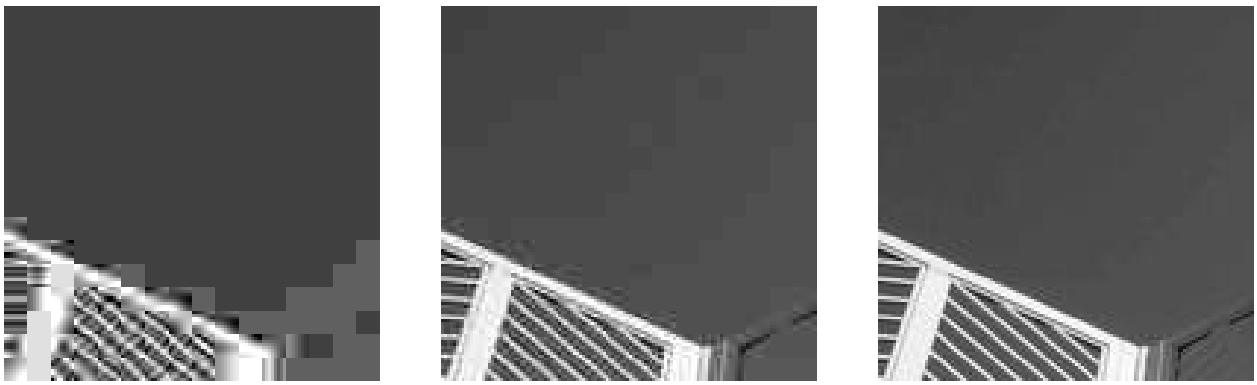
1.1.6 JPEG file format

In order to test the effect of jpeg compression, one can use the parameter quality. For the lowest quality, the loss of informations is really important (see Fig.6).

```
# test jpeg quality
imageio.imwrite("a_25.python.jpeg", ascent, quality=25);
imageio.imwrite("a_100.python.jpeg", ascent, quality=100);
imageio.imwrite("a_50.python.jpeg", ascent, quality=50);
imageio.imwrite("a_75.python.jpeg", ascent, quality=75);
imageio.imwrite("a_1.python.jpeg", ascent, quality=1);
```

1.2 Histogram

To plot the histogram of an image, you can use the hist function (see result in Fig. 7).



(a) Quality 1. (b) Quality 25. (c) Quality 75.

Figure 6: Different quality used to compress jpeg files.

```
plt.hist(ascent.flatten(), 256)
plt.show()
```

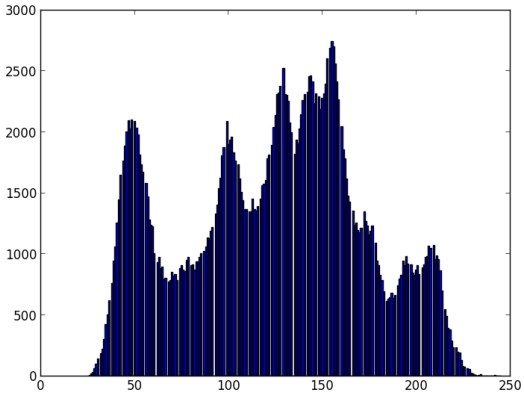


Figure 7: Histogram of ascent image.

You can also write your own code. The execution time is higher for these two functions than for the previous one.

```python
# Histogram function with 2D image
def compute_histogram(image):
    tab = np.zeros((256, ), dtype='I')
    X, Y = image.shape
    for i in range(X):
        for j in range(Y):
            tab[image[i,j]]+=1

    return tab
```

```python
# Histogram function with flatten image (vector)
def compute_histogram2(image):
    im = image.flatten()
    tab = np.zeros((256, ), dtype='I')
    for i in im:
        tab[i]+=1
    return tab
```

The following code presents a comparison of the different method for histogram computation.

```
# load ascent image and compute histograms
ascent = misc.ascent()
t0 = time.clock()
h = compute_histogram(ascent)
t1 = time.clock()
h2=compute_histogram2(ascent)
t2 = time.clock()

# .... plots
print "execution time 2D:%g" %(t1-t0)
plt.subplot(131)
plt.plot(h)
plt.title('2D function')

print "execution time 1D:%g" %(t2-t1)
plt.subplot(132)
plt.plot(h2)
plt.title('1D function')

# last plot: with matplotlib function
plt.subplot(133)
t3 = time.clock()
plt.hist(ascent.flatten(), 256)
t4 = time.clock()
print "execution time matplotlib:%g" %(t4-t3)

# display
plt.show()
```

The console outputs the following computation durations:

```
execution time 2D: 1.40284 s
execution time 1D: 1.27649 s
execution time matplotlib: 0.261095 s
```

1.3 Linear mapping of the image intensities

The linear mapping is a simple method that stretches linearly the histogram. If displayed with matplotlib, the images is linearly stretched, thus the modification cannot be observed.

```python
def image_stretch(image):
    # return image with new maximum and minimum at 255 and 0
    minimum = image.min();
    maximum = image.max();
    a = 255/(maximum-minimum);
    b = -255*minimum/(maximum-minimum);

    return a*image+b;
```

1.4 Aliasing effect

```python
# aliasing effect (Moire)
def circle(fs, f):
    # Generates an image with aliasing effect
    # fs: sample frequency
    # f : signal frequency
    t = np.arange(0,1,1./fs);
    ti,tj = np.meshgrid(t,t);
    C = np.sin(2*np.pi*f*np.sqrt(ti**2 +tj**2));
    return C
```

The image of Fig. 8 is generated with the following code.

```python
C = circle(300,50);
plt.imshow(C, cmap=plt.cm.gray);
plt.show()

imageio.imwrite('moire.png', C);
```

1.5 Low-pass filtering

The module scipy.ndimage.filters contains the usual filter functions.

The mean filter is illustrated in Fig. 9.

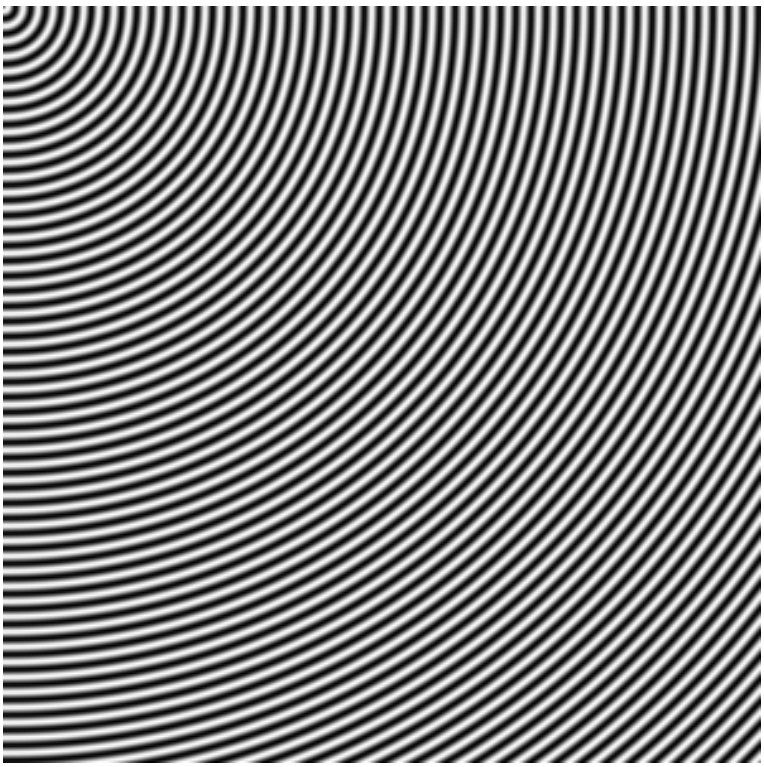


Figure 8: Moiré effect, generated with $f_s = 300$ and $f = 50$.

```
# mean on a 3x3 neighborhood
m3 = ndimage.filters.uniform_filter(ascent)
m25= ndimage.filters.uniform_filter(ascent, 25)

plt.subplot(121)
plt.imshow(m3, cmap=plt.cm.gray)
plt.axis('off')
plt.title('3x3 mean filter')

plt.subplot(122)
plt.imshow(m25, cmap=plt.cm.gray)
plt.axis('off')
plt.title('25x25 mean filter')

plt.show()
```

1.5.1 Gaussian filter

The gaussian filter is presented in Fig. 10.

(a) Neighborhood of size 3x3.

(b) Neighborhood of size 25x25.

Figure 9: Mean filters.

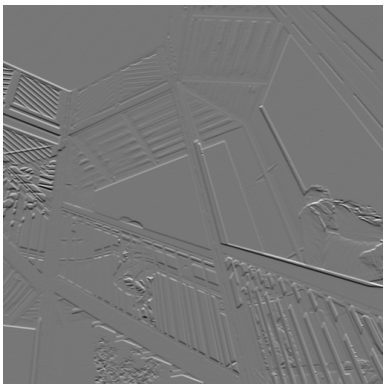
```python
# ascent image
ascent = misc.ascent()

# Gaussian filter
gaussian= ndimage.filters.gaussian_filter(ascent, 5)
```

Figure 10: Gaussian filter of size 5.

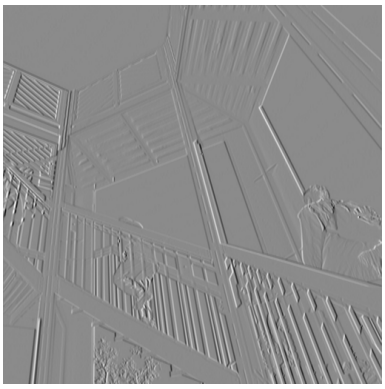
1.6 Derivative filters

Derivative filters (Prewitt, Sobel...) use a finite derivation approximation. They are very sensitive to noise (as every system using a derivation). The gradient is defined as a vector, and a norm should be used to display a resulting image. Notice that with these filters, the connexity of the contours is not preserved. Illustration is proposed in Fig. 11.

```python
# ascent image
ascent = misc.ascent()
ascent.astype('int32');

# Prewitt filter
prewitt0 = ndimage.filters.prewitt(ascent, axis=0)
prewitt1 = ndimage.filters.prewitt(ascent, axis=1)

# Sobel filter
dy = ndimage.filters.sobel(ascent, axis=0) # vertical
dx = ndimage.filters.sobel(ascent, axis=1) # horizontal
mag = np.hypot(dx, dy)  # magnitude
sobel = mag * 255.0 / mag.max()  # normalize (Q&D)

# display results
plt.subplot(131)
plt.imshow(prewitt0, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Prewitt filter axis 0')

plt.subplot(132)
plt.imshow(prewitt1, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Prewitt filter axis 1')

plt.subplot(133)
plt.imshow(sobel, cmap=plt.cm.gray)
plt.axis('off')
plt.title('Sobel filter')

plt.show()
```



(a) Prewitt filter for axis x.



(b) Prewitt filter for axis y.

Figure 11: Prewitt filter.