1 Python correction

```python
import numpy as np
from scipy import ndimage , misc
import matplotlib.pyplot as plt
```

1.1 Linear diffusion

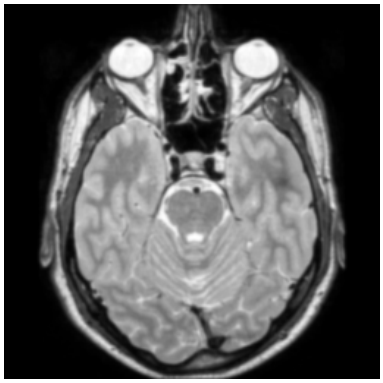
The linear diffusion filter is equivalent to the classical gaussian filter. The gradients are decomposed into the 4 directions so that the next developments (non linear diffusion filters) are made easy.

```python
def linearDiffusion(I, nbIter, dt):
    # linear diffusion
    # I: image
    # nbIter: number of iterations
    # dt: step time
    hW = np.array([[1, -1, 0]]);
    hE = np.array([[0,-1,1]]);
    hN = np.transpose(hW);
    hS = np.transpose(hE);

    Z = I.copy();
    for i in range(nbIter):
        gW = ndimage.convolve(Z, hW, mode='constant');
        gE = ndimage.convolve(Z, hE, mode='constant');
        gN = ndimage.convolve(Z, hN, mode='constant');
        gS = ndimage.convolve(Z, hS, mode='constant');
        Z = Z + dt*(gW+gE+gN+gS);
    return Z
```

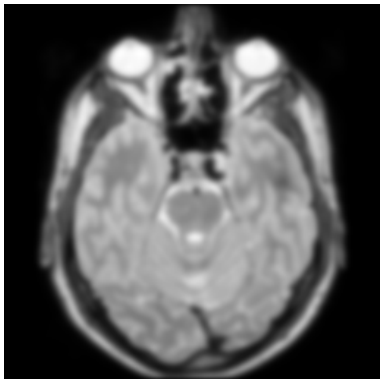
The code to filter images is presented as follows. Results are in Fig. 1.

```python
I = misc.imread("cerveau.png")/255.;
F = linearDiffusion(I, 50, .05);
F2= linearDiffusion(I, 200, .05);
plt.subplot(1,3,1);
plt.imshow(I, cmap=plt.cm.gray);
plt.subplot(1,3,2);
plt.imshow(F, cmap=plt.cm.gray);
plt.subplot(1,3,3);
plt.imshow(F2, cmap=plt.cm.gray);
```

(a) Original image.



(b) Linear diffusion filter, with $\alpha = 0.1$, $dt = 0.05$ and 10 iterations.



(c) Linear diffusion filter, with $\alpha = 0.1$, $dt = 0.05$ and 50 iterations.

Figure 1: Linear diffusion filter. Contours are not preserved: this is equivalent to a Gaussian filter.

1.2 Nonlinear diffusion

The nonlinear diffusion is an adaptation of the diffusion to the informations contained in the images. These informations are high frequency components. The following diffusion coefficient is proposed by Perona and Malik.

```
def c(I, alpha):
    # diffusion coefficient
    # I: image
    # alpha: diffusion parameter
    return np.exp(-(I/alpha)**2);
```

The gradients are evaluated in the 4 directions, a specific coefficient is thus applied to each one.

```python
def nonlinearDiffusion(I, nbIter, alpha, dt):
    # linear diffusion
    # I: image
    # nbIter: number of iterations
    # dt: step time
    hW = np.array([[1, -1, 0]]);
    hE = np.array([[0,-1,1]]);
    hN = np.transpose(hW);
    hS = np.transpose(hE);

    Z = I.copy();

    for i in range(nbIter):
        #print "%d" % i
        gW = ndimage.convolve(Z, hW, mode='constant');
        gE = ndimage.convolve(Z, hE, mode='constant');
        gN = ndimage.convolve(Z, hN, mode='constant');
        gS = ndimage.convolve(Z, hS, mode='constant');

        Z= Z + dt*(c(np.abs(gW), alpha)*gW + c(np.abs(gE), alpha)*gE
        + c(np.abs(gN), alpha)*gN + c(np.abs(gS), alpha)*gS);

    return Z
```

The script to filter the images, for $\alpha = 0.1$, $dt = 0.05$ and for 10 and 50 iterations, is the following. Results are presented in Fig. 2.

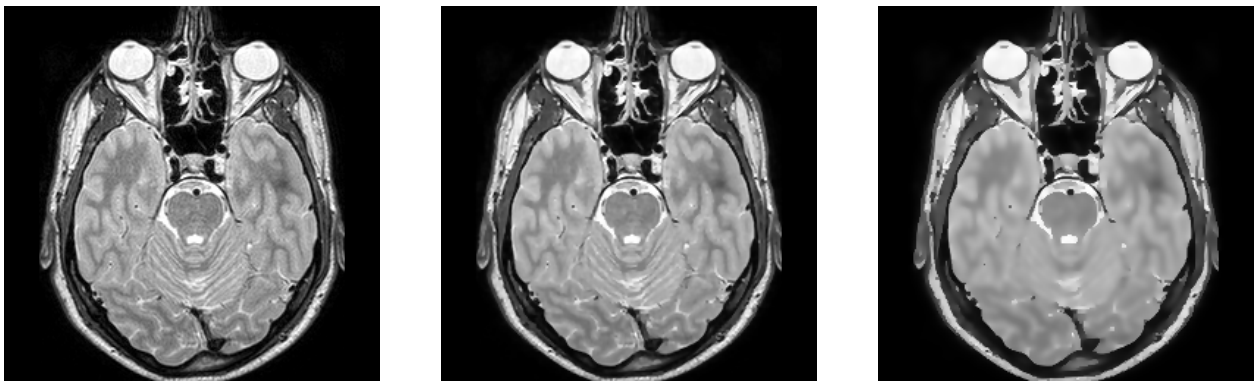
```python
alpha = 0.1;
dt = .05;
I = misc.imread("cerveau.png")/255.;

F = nonlinearDiffusion(I, 10, alpha, dt);
F2= nonlinearDiffusion(I, 50, alpha, dt);
```

1.3 Degenerate diffusion

Erosion and dilation can theoretically be coded with this numerical scheme. However, some numerical problems appear in the first version, which are corrected in the second version.

1.3.1 First version

This first version is the direct transcription of the numerical scheme. As observed in Fig.3, shocks (peaks) appear after a few iterations.



(a) Original image.

(b) Non linear diffusion filter, with $\alpha = 0.1$, $dt = 0.05$ and 10 iterations.

(c) Non linear diffusion filter, with $\alpha = 0.1$, $dt = 0.05$ and 50 iterations.

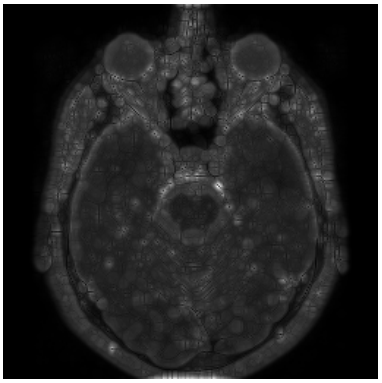
Figure 2: Nonlinear diffusion filter. Contours are preserved.

```python
def degenerateDiffusion1(image, nbIter, dt):
    # degenerate diffusion
    # I: image
    # nbIter: number of iterations
    # dt: step time
    h = np.array([[-1,0,1]]);
    ht= np.transpose(h);

    Zdilation = image;
    Zerosion  = image;
    for i in range(nbIter):
        gH = ndimage.convolve(Zdilation, h , mode='constant');
        gV = ndimage.convolve(Zdilation, ht, mode='constant');

        jH = ndimage.convolve(Zerosion, h , mode='constant');
        jV = ndimage.convolve(Zerosion, ht, mode='constant');

        Zdilation = Zdilation + dt*np.sqrt(gV**2 + gH**2);
        Zerosion  = Zerosion + dt*np.sqrt(jV**2 + jH**2);

    return (Zdilation, Zerosion)
```

1.3.2 More sophisticated version

Another scheme, more numerically stable, can be preferred to the previous one. Results are presented in Fig. 4.

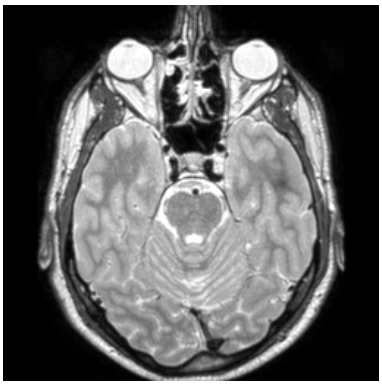
(a) Dilation by diffusion, for $dt = 0.05$ and $nbIter = 10$.



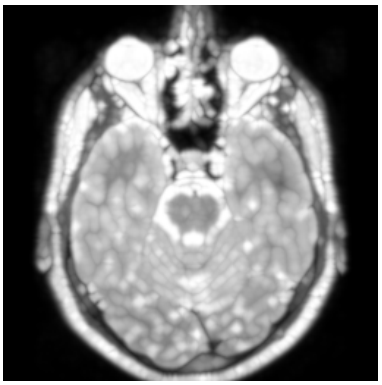
(b) Dilation by diffusion, for $dt = 0.05$ and $nbIter = 50$.

Figure 3: Mathematical morphology operations by diffusion.

```python
def degenerateDiffusion2(image, nbIter, dt):
    # degenerate diffusion
    # I: original image
    # nbIter: number of iterations
    # dt: step time
    hW = np.array([[1,-1,0]]);
    hE = np.array([[0, -1, 1]]);
    hN = np.transpose(hW);
    hS = np.transpose(hE);
    Zdilation = image;
    Zerosion  = image;

    for i in range(nbIter):
        gW = ndimage.convolve(Zdilation, hW, mode='constant');
        gE = ndimage.convolve(Zdilation, hE, mode='constant');
        gN = ndimage.convolve(Zdilation, hN, mode='constant');
        gS = ndimage.convolve(Zdilation, hS, mode='constant');

        jW = ndimage.convolve(Zerosion, hW, mode='constant');
        jE = ndimage.convolve(Zerosion, hE, mode='constant');
        jN = ndimage.convolve(Zerosion, hN, mode='constant');
        jS = ndimage.convolve(Zerosion, hS, mode='constant');

        g = np.sqrt( np.minimum(0, -gW)**2 + np.maximum(0, gE)**2 + np.
            ↪ minimum(0, -gN)**2 + np.maximum(0, gS)**2);
        j = np.sqrt( np.maximum(0, -jW)**2 + np.minimum(0, jE)**2 + np.
            ↪ maximum(0, -jN)**2 + np.minimum(0, jS)**2);

        Zdilation = Zdilation + dt * g;
        Zerosion = Zerosion - dt * j;

    return Zdilation, Zerosion
```



(a) Dilation by diffusion, for $dt = 0.05$ and nbIter=10.



(b) Dilation by diffusion, for $dt = 0.05$ and nbIter=50.

Figure 4: Mathematical morphology operations by diffusion, sophisticated version.