1 Python correction

```python
from scipy import ndimage , misc
import numpy as np

import matplotlib.pyplot as plt
```

1.1 Elementary operators

First of all, one have to create a structuring element. The following function can be used to create a circular structuring element. A square can also be used.

```python
def disk(radius):
    # defines a circular structuring element with radius given by 'radius
    ↪ '
    x = np.arange(-radius, radius+1, 1);
    xx, yy = np.meshgrid(x, x);
    d = np.sqrt(xx**2 + yy**2);
    return d<=radius;
```

The following code presents basic mathematical morphology operations. First of all, declare the structuring element.

```python
# read binary image (and ensure binarization)
B = imageio.imread("B.jpg");
B = B>100;

# Structuring element
square = np.ones((5,5));
```

Erosion and dilation are the two elementary function of mathematical morphology.

```python
# Erosion
Bsquare_erode = ndimage.morphology.binary_erosion(B, structure=square);
plt.subplot(231);
plt.imshow(Bsquare_erode); plt.title("erosion")
imageio.imwrite('erosion.png', Bsquare_erode);

# Dilation
Bsquare_dilate = ndimage.morphology.binary_dilation(B, structure=square);
plt.subplot(232);
plt.imshow(Bsquare_dilate);plt.title("dilation")
imageio.imwrite('dilation.png', Bsquare_dilate);
```

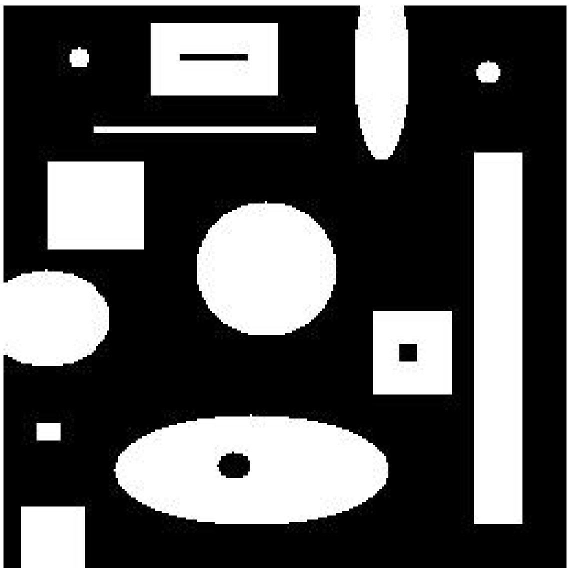
Opening and closing are a combination of the two previous functions.

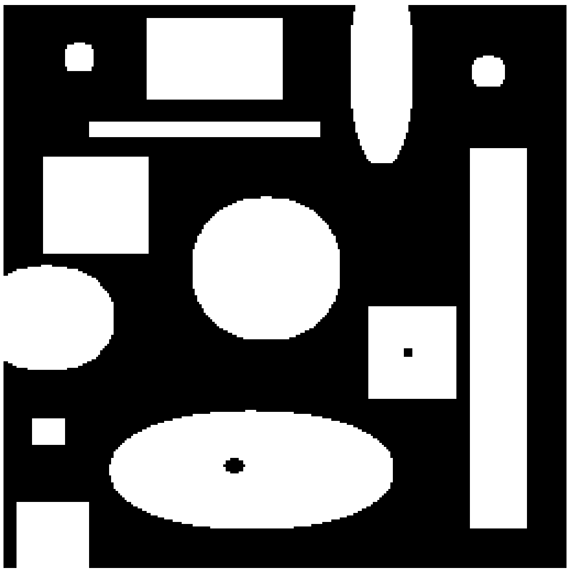
```python
# Opening
Bsquare_open = ndimage.morphology.binary_opening(B, structure=square);
plt.subplot(233);
plt.imshow(Bsquare_open);plt.title("opening")
imageio.imwrite('open.png', Bsquare_open);

# Closing
Bsquare_close = ndimage.morphology.binary_closing(B, structure=square);
plt.subplot(234);
plt.imshow(Bsquare_close);plt.title("closing")
imageio.imwrite('close.png', Bsquare_close);
```

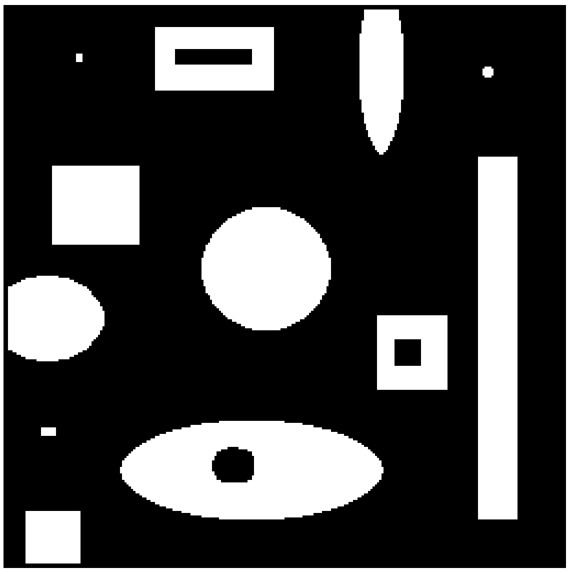
The results are presented in Fig. 1.

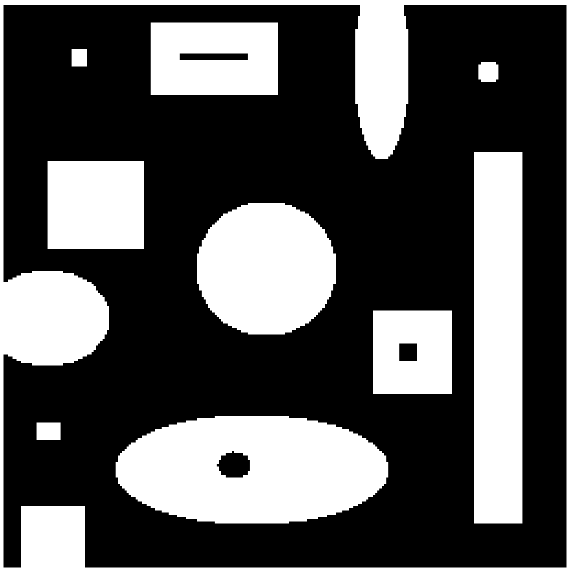
1.2 Morphological reconstruction

The algorithm of morphological reconstruction is coded like this in python:


(a) Original image.


(b) Dilation.


(c) Erosion.


(d) Opening.

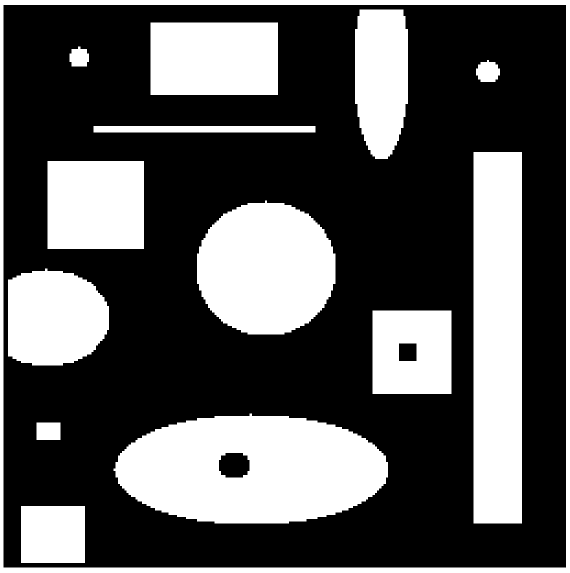

(e) Closing.

Figure 1: Basic mathematical morphology operations.

```python
def reconstruct(image, mask):
    # should be binary images
    M = np.minimum(mask, image);

    area = ndimage.measurements.sum(M);
    s=0

    se = np.array([[0, 1, 0], [1, 1, 1], [0, 1, 0]]);
    while (area != s):
        s = area;
        M = np.minimum(image, ndimage.morphology.binary_dilation(M,
            ↪ structure=se));
        area = ndimage.measurements.sum(M);

    return M
```

The Fig. 2 illustrates the morphological recontruction.

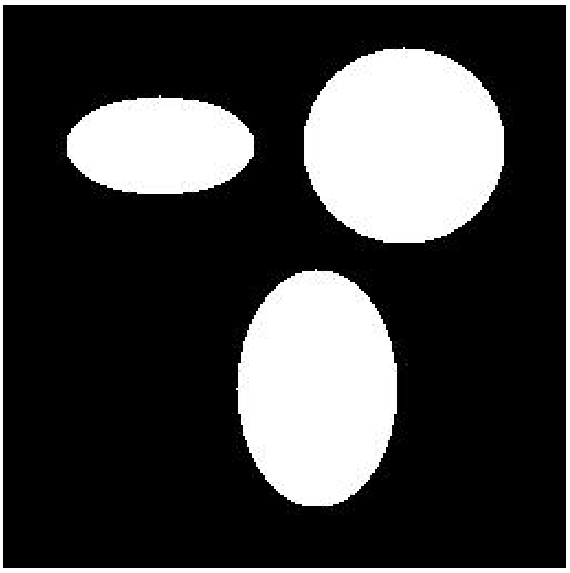
```python
A=imageio.imread('A.jpg');
A = A > 100;
M=imageio.imread('M.jpg');
M = M > 100;
# reconstruction de A par M
AM=reconstruct(A, M);

# display results
plt.subplot(1, 3, 1);
plt.imshow(A);
plt.subplot(1, 3, 2);
plt.imshow(M);
plt.subplot(1, 3, 3);
plt.imshow(AM);
plt.show();
```

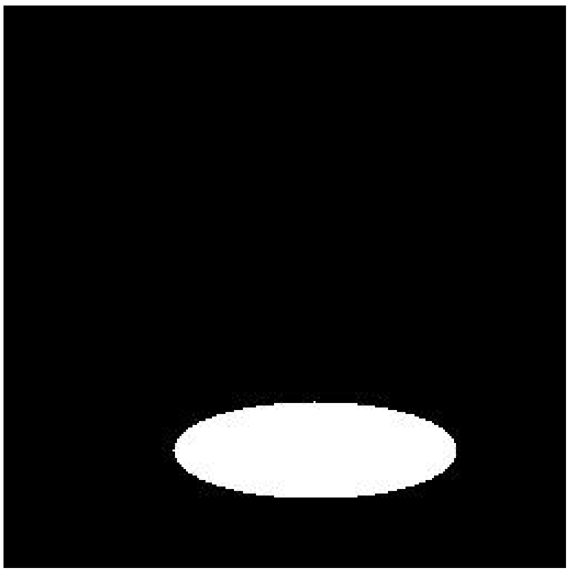
1.3 Operators by reconstruction

1.3.1 Remove objects touching the borders

The Fig. 3 illustrates the suppression of objects touching the borders of the image. If $\mathcal{B}$ represents the border of the image (create an array of the same size as the image, with zeros everywhere and ones at the sides), then this operation is defined by:

$$\text{killBorders}(I) = I \setminus \rho_I(\mathcal{B})$$


(a) Image A.


(b) Image M.

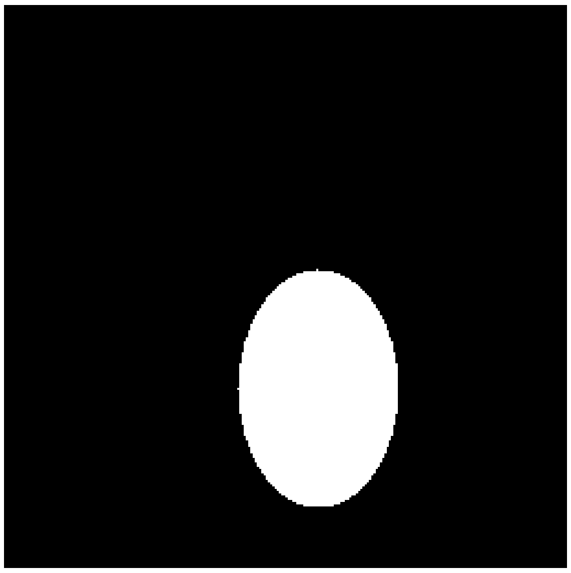

(c) AM=reconstruct(A, M);.

Figure 2: Morphological reconstruction.

```python
def killBorders(A):
    # remove cells touching the borders of the image
    m, n = A.shape
    M = np.zeros((m,n));
    M[0,:] = 1;
    M[m-1,:] = 1;
    M[:,0] = 1;
    M[:,n-1] = 1;
    M = reconstruct(A, M);
    return A-M
```

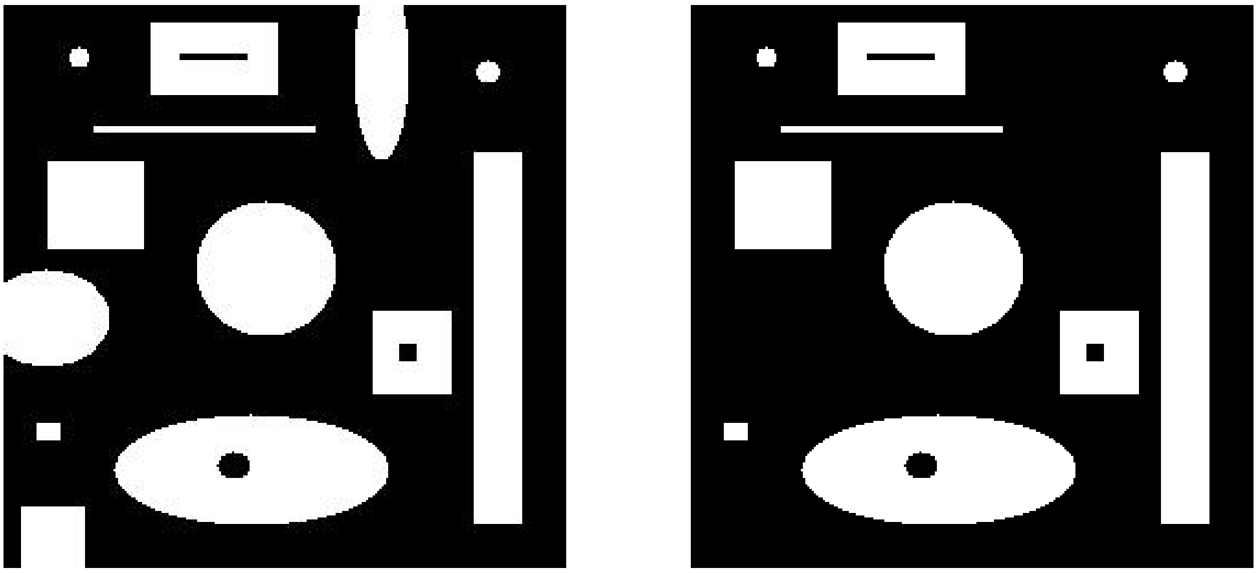

(a) Original image. (b) Objects removed.

Figure 3: Suppress border objects.

1.3.2 Remove small objects

This is illustrated in Fig. 4. It consists in an erosion followed by a reconstruction. The structuring element used in the erosion defines the objects considered as “small”.

$$\text{killSmall}(I) = \rho_I(\varepsilon(I))$$

```python
def killSmall(A, n):
    # destroy small objects (size smaller than n)
    se = np.ones((n, n));
    M = ndimage.morphology.binary_erosion(A, structure=se);
    return reconstruct(A, M);
```

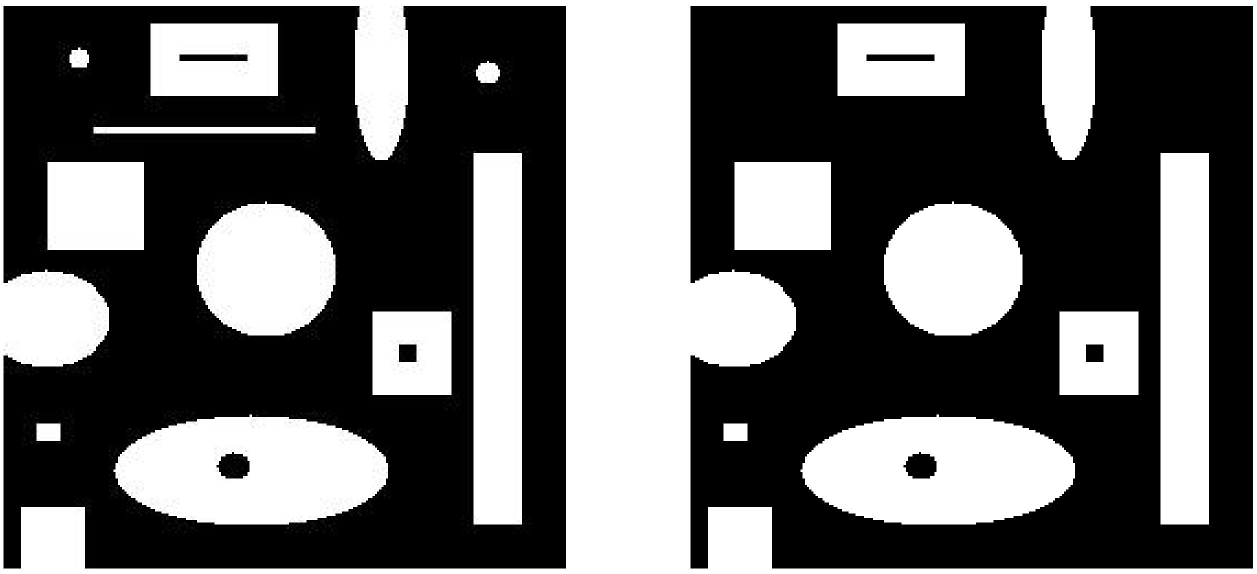

(a) Original image. (b) Small objects are removed.

Figure 4: Small objects removal.

1.3.3 Close holes in objects

This is illustrated in Fig. 5. The operation is given by the following equation, with $\mathcal{B}$ the border of image I, and X^C denoting the complementary of set X:

$$\text{removeHoles(I)} = \{\rho_{I^C}(\mathcal{B})\}^C$$

```python
def closeHoles(A):
    # close holes in objects
    Ac = ~A;
    m,n = A.shape;
    M = np.zeros((m,n));
    M[0,:] = 1;
    M[m-1,:] = 1;
    M[:,0] = 1;
    M[:,n-1] = 1;
    M = reconstruct(Ac, M);
    return ~M
```

1.4 Application

The application shown in Fig. 6 presents the segmentation of blood cells after removing small cells, closing holes and removing the cells touching the borders of the image.

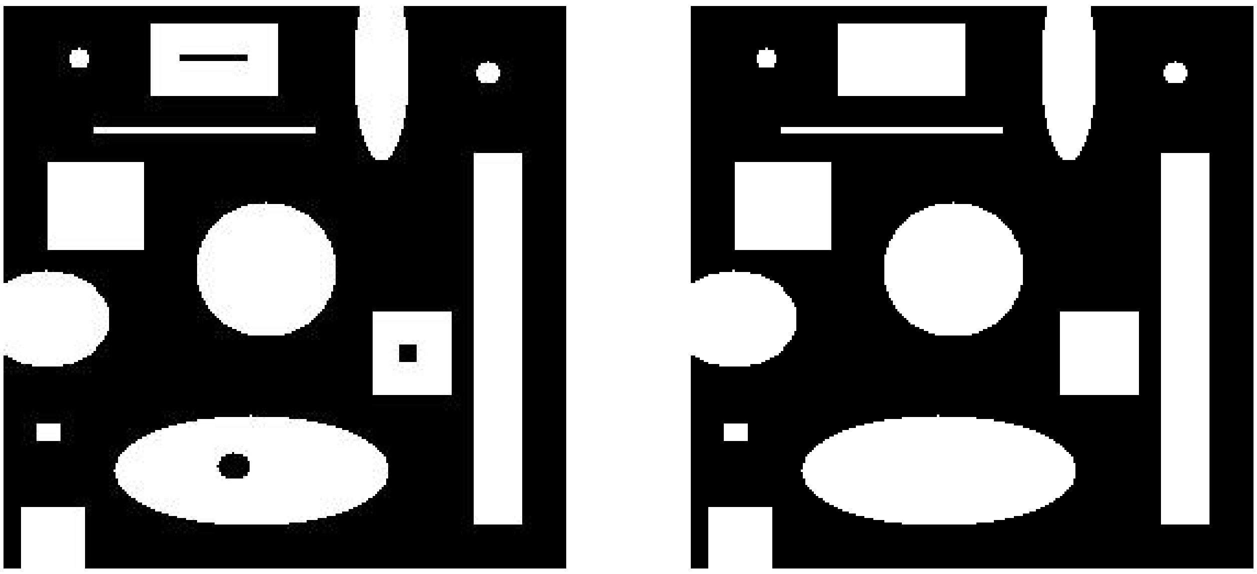
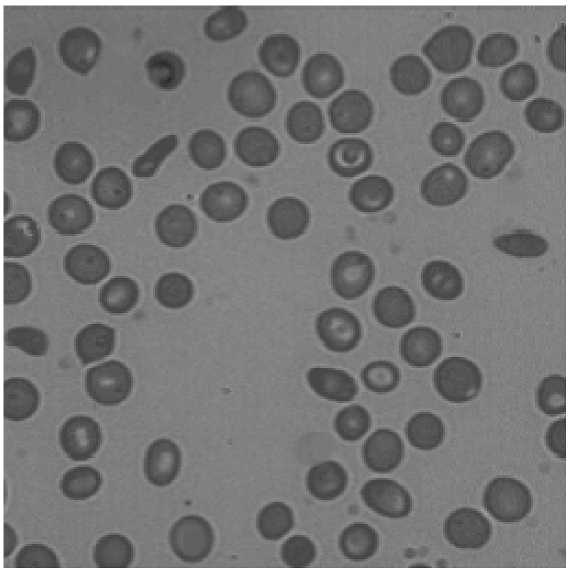
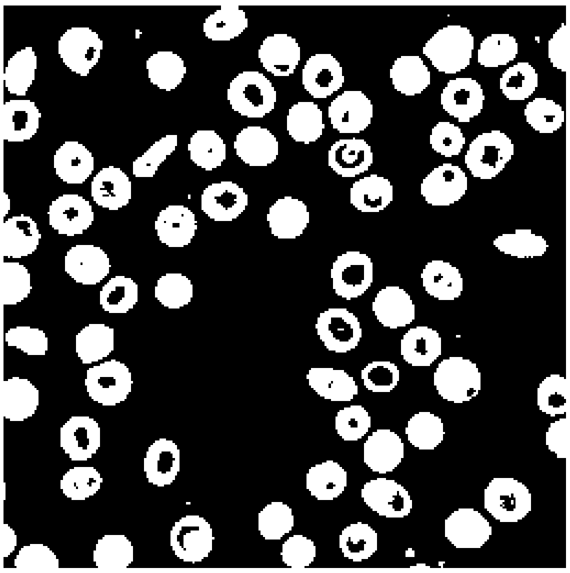

(a) Original image. (b) Holes in objects are closed.

Figure 5: Hole filling.

```python
cells = imageio.imread('cells.jpg')<98;
imageio.imwrite('cellsbw.png', cells);
B = closeHoles(cells);
B = killBorders(B);
B = killSmall(B, 5);
plt.subplot(1,2,1);
plt.imshow(cells);
plt.subplot(1,2,2);
plt.imshow(B);
plt.title('clean image')
plt.show()
imageio.imwrite('clean.png', B);
```


(a) Original image.


(b) Binarization.

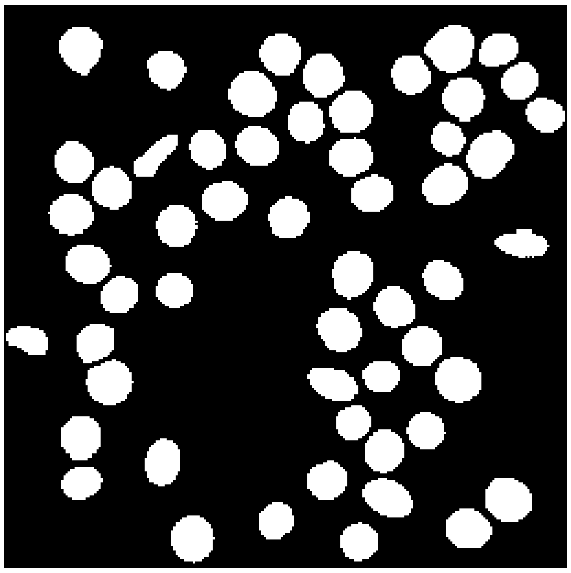

(c) Final segmentation of the cells.

Figure 6: Segmentation of the cells.