# 1 Python correction

```python
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from scipy.ndimage.morphology import distance_transform_edt
from scipy.ndimage.morphology import distance_transform_cdt
from skimage import morphology
```

## 1.1 Watershed and distance map

### 1.1.1 Regional maxima

The following code is an implementation of the regional maxima (from mathematical morphology definition). It deals with the case of float images as input.

```python
def rmax(I):
    """
    Own version of regional maximum
    This avoids plateaus problems of peak_local_max
    I: original image, int values
    returns: binary array, with True for the maxima
    """
    I = I.astype('float');
    I = I / np.max(I) * 2**31;
    I = I.astype('int32');
    h = 1;
    rec = morphology.reconstruction(I, I+h);
    maxima = I + h - rec;
    return maxima>0
```

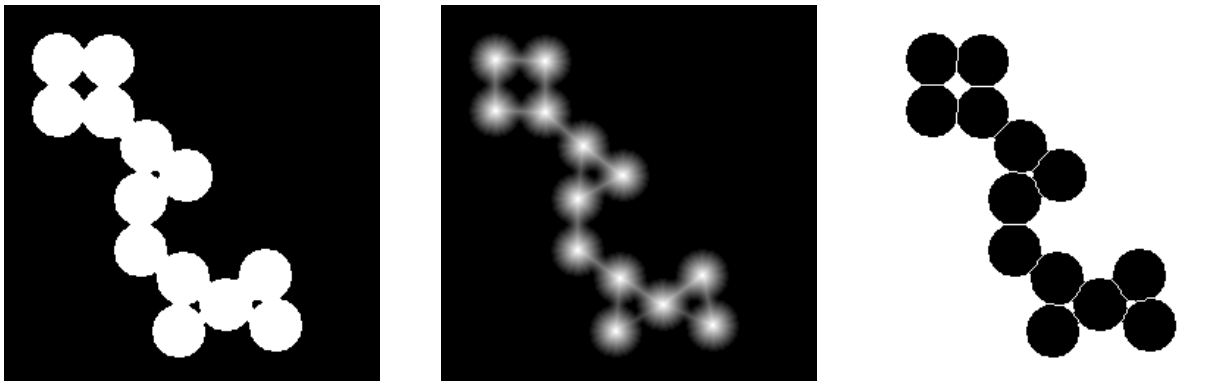
### 1.1.2 Distance map followed by a watershed

This method is a classical way of performing the separation of some objects by proximity or influence zones. It is illustrated in Fig.1.

```
A = misc.imread('circles.tif');

# chamfer distance gives integer distances
dm = distance_transform_edt(A);

# regional maxima
local_maxi = rmax(dm);
```

The local maxima will be used as a marker for the watershed operation, in order to perform the separation of the grains, see Fig.1.

```
# watershed segmentation for separating the circles
markers = ndi.label(local_maxi, np.ones((3, 3)))[0]

W = morphology.watershed(-dm, markers, watershed_line=True)

# separation of the grains
B = A & W==0;
separation = ndi.label(B, np.ones((3,3)))[0];
```



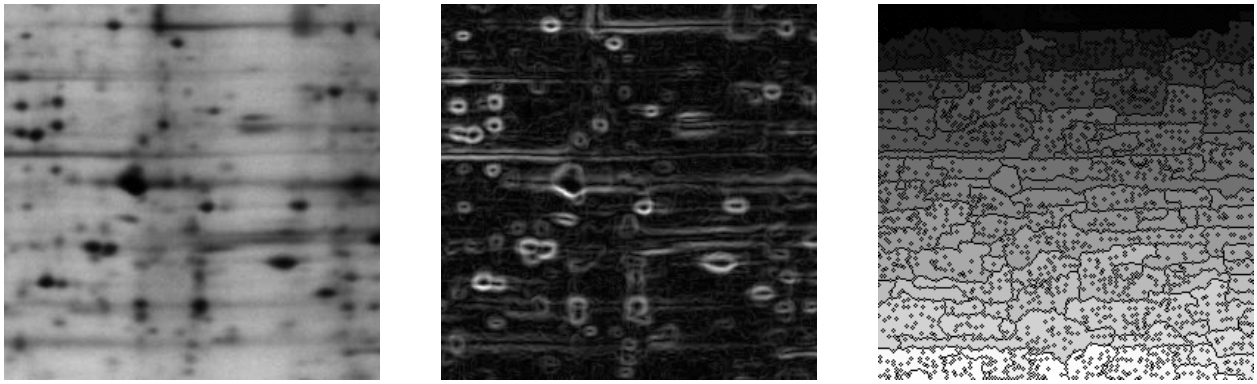
(a) Original image. (b) Distance map. (c) Separation of the grains.

Figure 1: Steps of the separation of the grains.

## 1.2 Watershed and image gradients

The gradient image amplifies the noise. Thus, the watershed operator directly applied to the gradient of the image produces an over-segmented image (see Fig.2).

```python
def sobel_mag(im):
    """
    Returns Sobel gradient magnitude
    im: image array of type float
    returns: magnitude of gradient (L2 norm)
    """
    dx = ndi.sobel(im, axis=1)   # horizontal derivative
    dy = ndi.sobel(im, axis=0)   # vertical derivative
    mag = np.hypot(dx, dy)  # magnitude
    return mag;
```



(a) Original image. (b) Amplitude of the gradient (Sobel). (c) Watershed segmentation.

Figure 2: Performing the watershed on the gradient image is usually not a good idea.

In fact, this method produces as many segments as there are minima in the gradient image Fig.3.

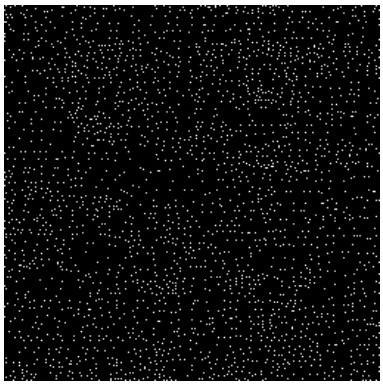


Figure 3: Local minima of the gradient image.

### 1.2.1 Solution: filtering the image

Before evaluating the gradient, the image is filtered. The number of minima is lower and this leads to a less over-segmented image (Fig.4).

```
SE = morphology.disk(2);
O = morphology.opening(gel, selem=SE);
F = morphology.closing(O, selem=SE).astype('float');
g = sobel_mag(F).astype('float');
```

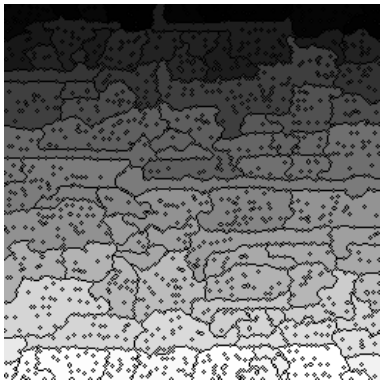
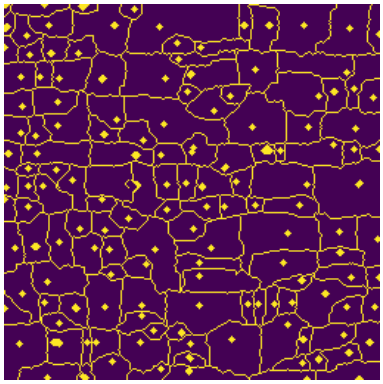


Figure 4: Even if the gradient is performed on the filtered image, there is still a high over-segmentation.

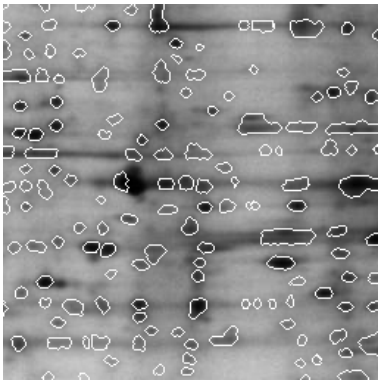
## 1.3 Watershed constrained by markers

The watershed can be constrained by markers: the markers can provide the correct number of regions. This method imposes both the background (external markers) and the objects (internal markers). The results are illustrated in Fig.5. The ultimate erosion of the internal markers is used to deconnect these markers from the external markers.

```
local_maxi = rmax(255-F);

markers = ndi.label(local_maxi, np.ones((3, 3)))[0]

W = morphology.watershed(F, markers, watershed_line=True)

markers2 = local_maxi | (W==0);
M = ndi.label(markers2, np.ones((3, 3)))[0]
segmentation = morphology.watershed(g, M, watershed_line=True);

gel[segmentation==0] = 255;
plt.imshow(gel);
plt.show();
misc.imsave("segmentation.python.png", gel);
```



(a) Markers of the background and of the objects.



(b) Final segmentation.

Figure 5: Watershed segmentation by markers.