1 Python correction

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

1.1 Generation of random noises

In order to convert stretch the images into the range [0;1], the following function can be used:

```python
def hist_stretch(I):
    # histogram stretching
    # returns values of I linearly stretched to range [0;1]
    I = I - np.min(I);
    I = I / np.max(I);
    return I;
```

Random noise illustrations are proposed in Figs.1 and 2, and a size $S = 32$ is used to generate the noisy images.

1.1.1 Uniform noise

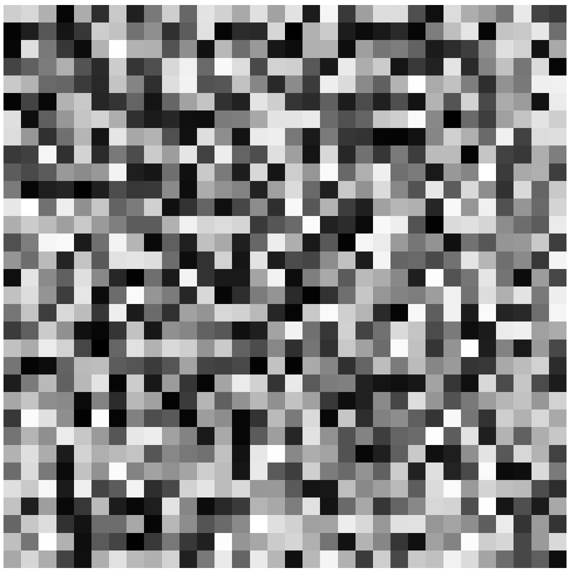
The python rand function generates values between 0 and 1 with uniform distribution.

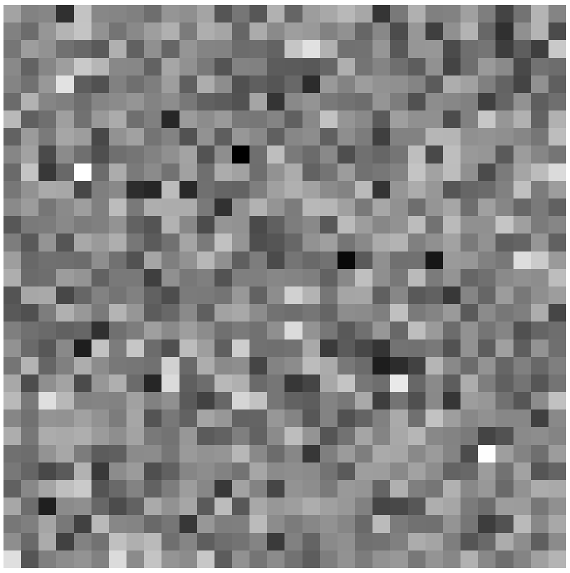
```python
S=32
a=0; b=255;
R1 = a + (b-a) * np.random.rand(S, S);
```

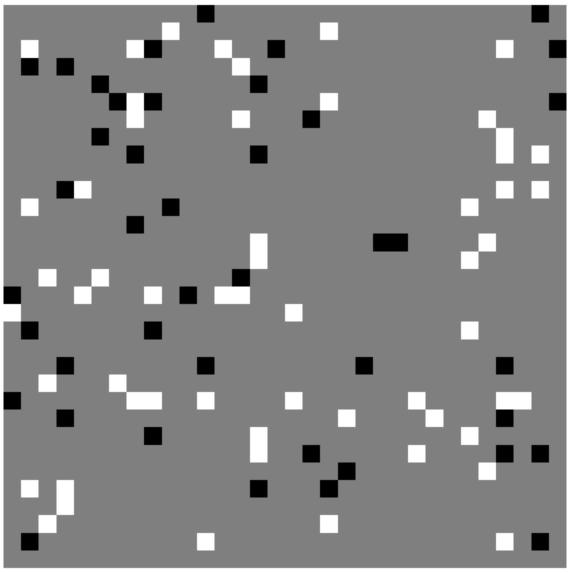
1.1.2 Gaussian noise

The python randn function generates values with normal centered distribution.

```python
a=0; b=1;
R2 = a + (b-a)*np.random.randn(S, S);
```


(a) Uniform noise.


(b) Gaussian noise.


(c) Salt and pepper noise.

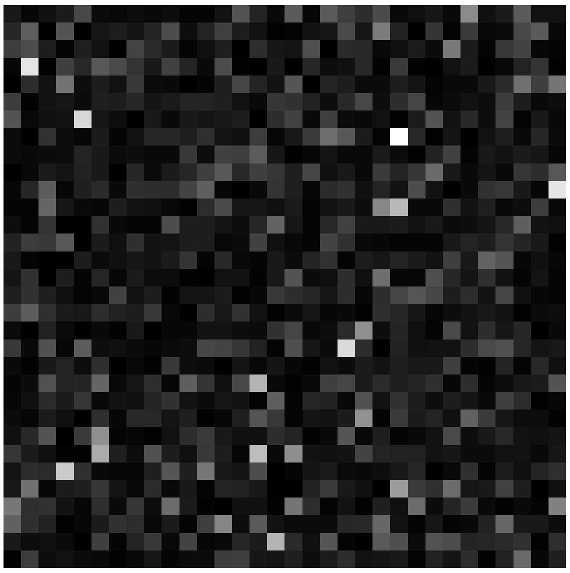

(d) Exponential noise.

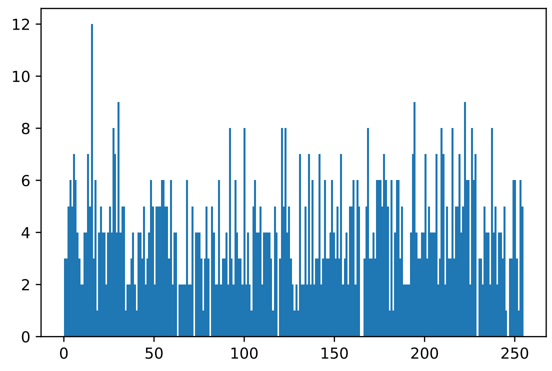
Figure 1: Resulting noise images.

1.1.3 Salt and pepper noise

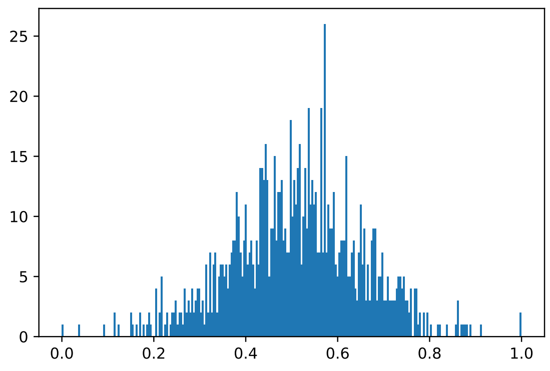
```python
a = 0.05; b = 0.1;
R3 = 0.5 * np.ones((S,S));
X = np.random.rand(S,S);
R3[X<=a] = 0;
R3[(X>a) & (X <= b)] = 1;
R3 = hist_stretch(R3);
```

1.1.4 Exponential noise

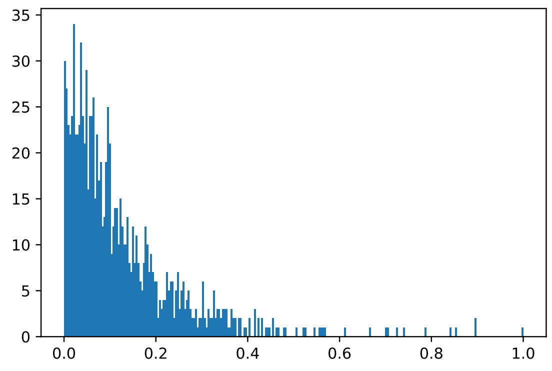
```python
a=1;
R4 = -1/a * np.log(1-np.random.rand(32, 32));
R4 = hist_stretch(R4);
```



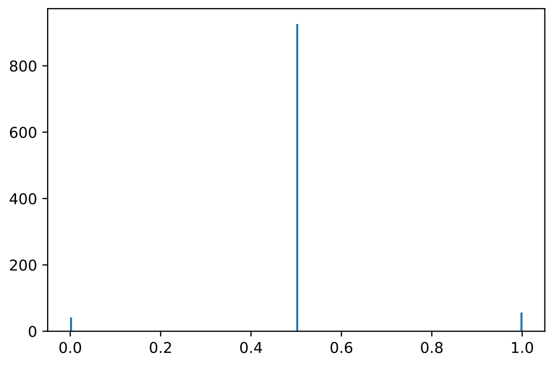
(a) Uniform noise.



(b) Gaussian noise.



(c) Exponential noise.



(d) Salt and pepper noise.

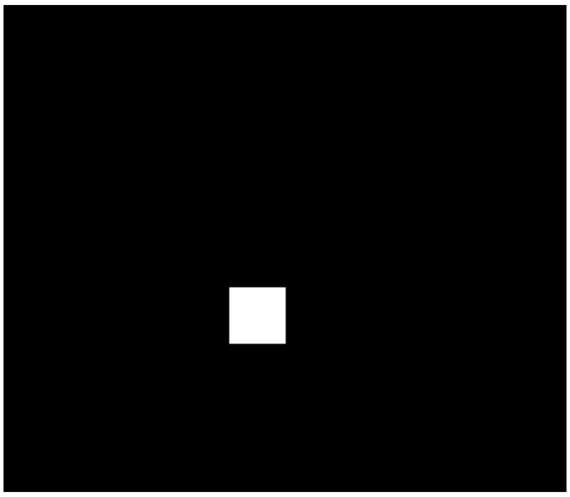
Figure 2: Histograms of the noise images generated with 32×32 pixels.

1.2 Noise estimation

In order to estimate the noise, a ROI of visually constant gray level is chosen, and its histogram is displayed. This is simulated by the following code, the result is displayed in Fig.3:

```python
# noise estimation
fig = plt.figure()
A = misc.imread('jambe.png');
roi = A[160:200, 200:240];
plt.hist(roi.flatten(),255);
fig.savefig("histo_roi_leg.pdf", bbox_inches='tight');
#plt.show()

# add exponential noise to image
nx, ny = A.shape;
expnoise = -1/.5 * np.log(1-np.random.rand(nx, ny));
expnoise = expnoise / np.max(expnoise);
B = A + 255*expnoise;
B = hist_stretch(B);
fig = plt.figure()
plt.imshow(B, cmap='gray');
misc.imsave("leg_exponential.png", B);
#plt.show()

# add gaussian noise to image
nx, ny = A.shape;
gaussnoise = 50*np.random.rand(nx, ny);
B = A + gaussnoise;
B = hist_stretch(B);
fig = plt.figure()
plt.imshow(B, cmap='gray');
misc.imsave("leg_gaussian.png", B);
```

In the case of exponential and Gaussian noise added to the image, The histograms are displayed in Figs.4 and 5.


(a) ROI.

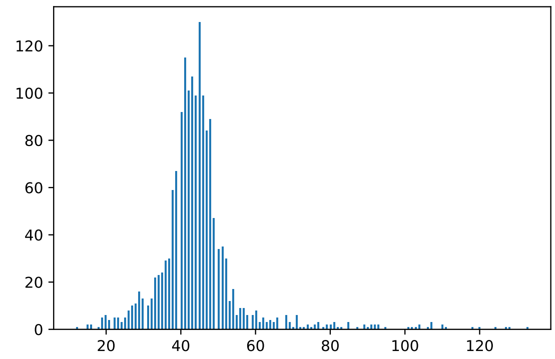
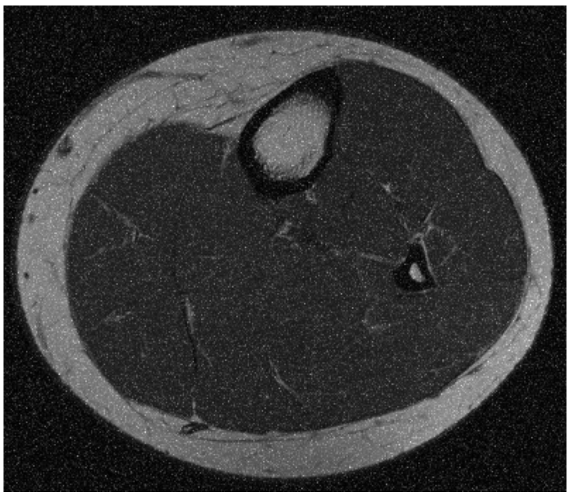

(b) Histogram.

Figure 3: Histogram of the Region of Interest.


(a) Addition of exponential noise to the original image of the leg.

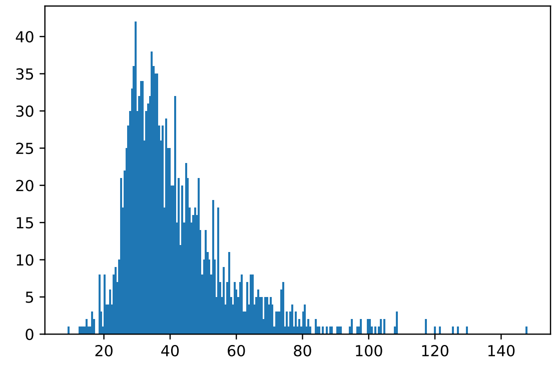
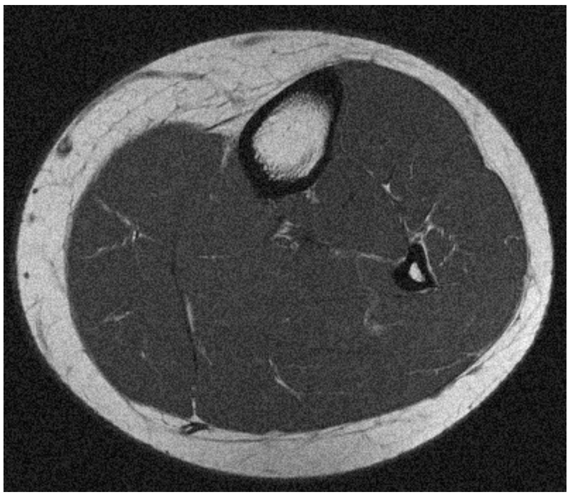
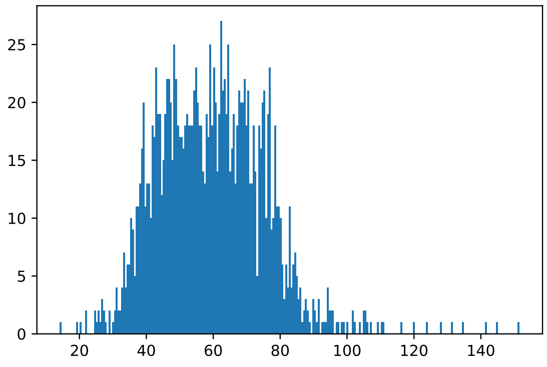

(b) Histogram of the ROI.

Figure 4: Exponential noise.


(a) Addition of Gaussian noise to the original image of the leg.



(b) Histogram of the ROI.

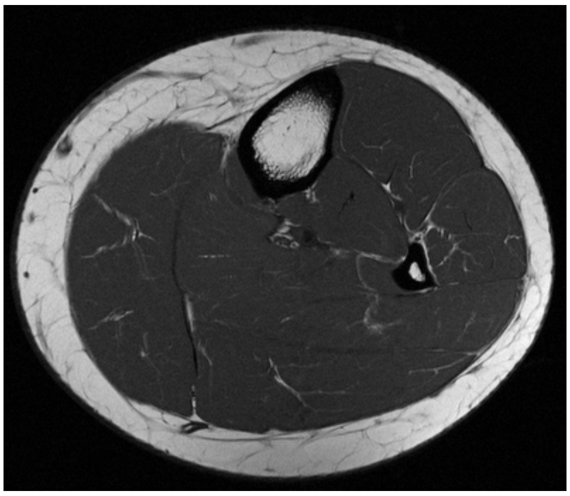
Figure 5: Gaussian noise.

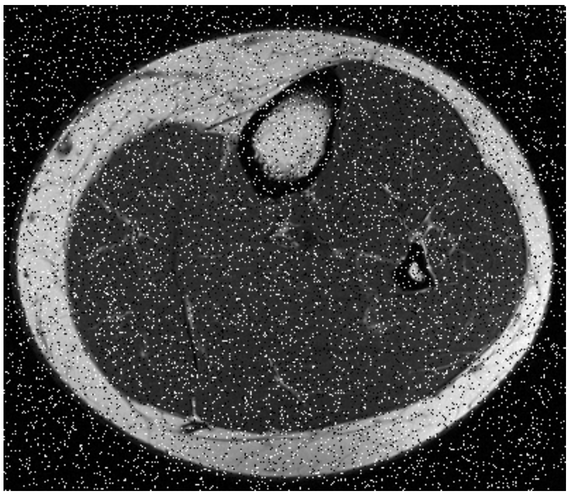
1.3 Image restoration by spatial filtering

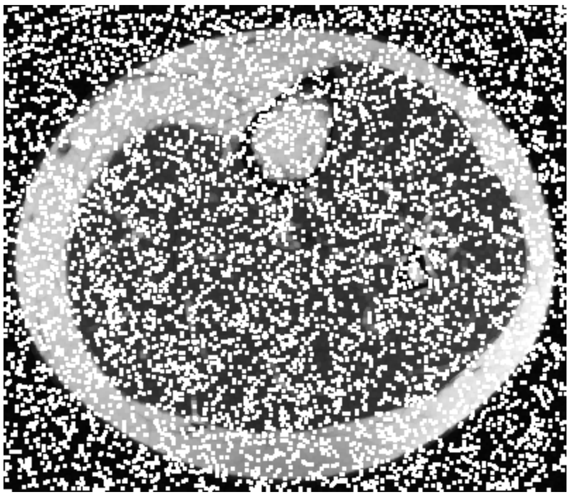
The following code is used to filter the images. The results are displayed in Fig.6.

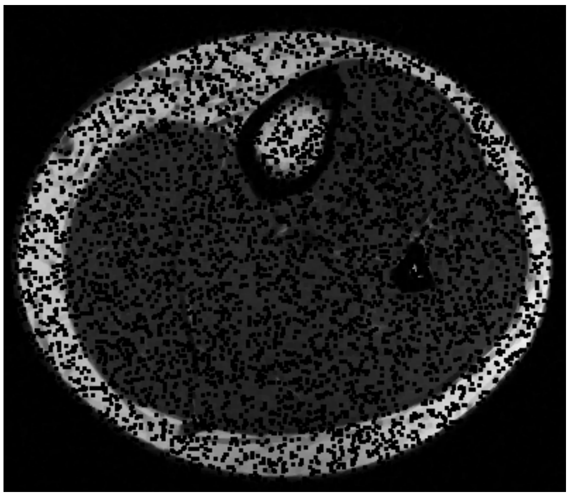
```
a = 0.05; b = 0.05;
spnoise = 0.5 * np.ones((nx,ny));
X = np.random.rand(nx,ny);
B = A.copy();
B[X<=a] = 0;
B[(X>a) & (X <= (a+b))] = 255;
fig = plt.figure()
plt.imshow(B, cmap='gray');
misc.imsave("leg_sp.png", B);
#plt.show()

### filtering by convolution
#average filtering
B1 = ndimage.uniform_filter(B,5);
misc.imsave("leg_uniform.png", B1);
B2 = ndimage.minimum_filter(B,3);
misc.imsave("leg_minimum.png", B2);
B3 = ndimage.maximum_filter(B,3);
misc.imsave("leg_maximum.png", B3);
B4 = ndimage.median_filter(B, 7);
misc.imsave("leg_median.png", B4);
B5 = amf(B, 7);
plt.imshow(B5, cmap='gray');
misc.imsave("leg_amf.png", B5);
```

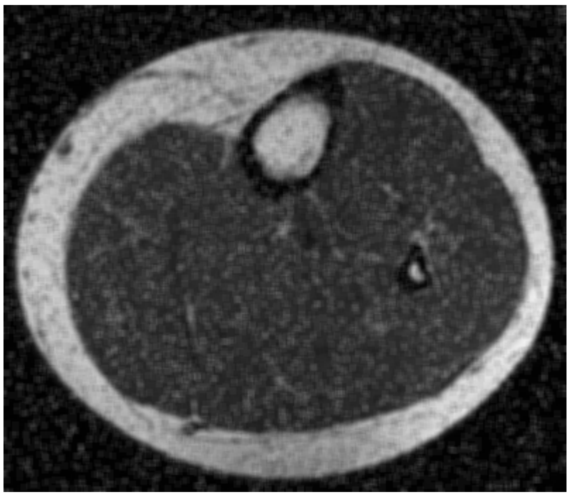
What can be noticed is that min and max filters are unable to restore the image (opening


(a) Original image.


(b) Noisy image (salt and pepper).


(c) Maximum filter.


(d) Minimum filter.


(e) Mean filter.

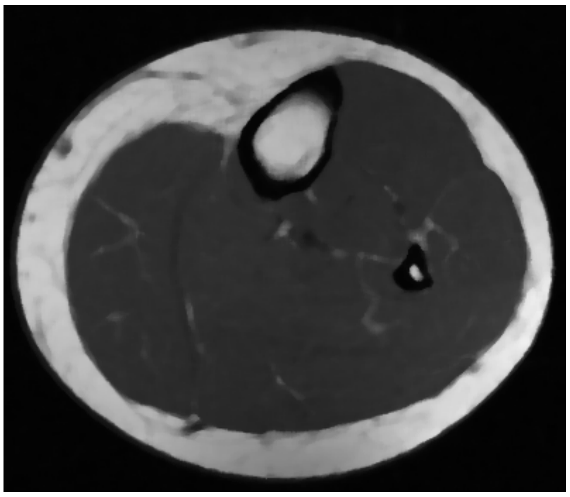

(f) Median filter.

Figure 6: Different filters applied to the noisy image. The median filter is particularly adapted in the case of salt and pepper noise (impulse noise), but still destroy the structures observed in the images.

and closing filters, from the mathematical morphology, could be a solution to explore). The mean filter is a better solution, but an average value is highly modified by an impulse noise. The median filter is the optimal solution is order to suppress the noise, but fine details are lost. An interesting solution is to apply an adaptive median filter.

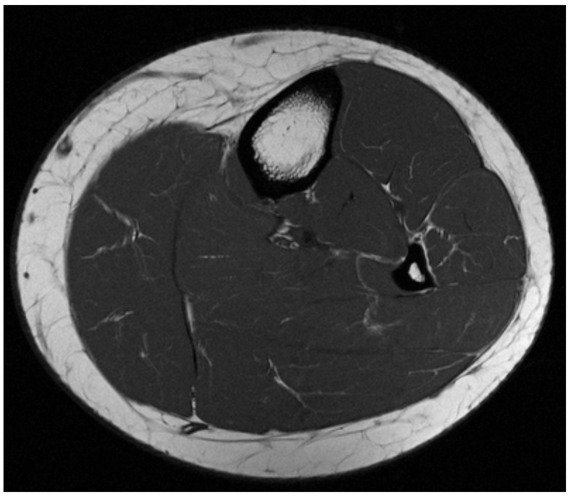
1.3.1 Adaptive median filter

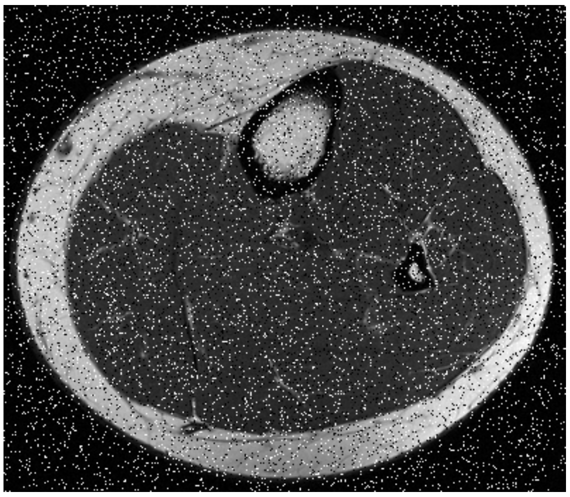
The following code is a simple implementation of the algorithm previously presented. It has not been optimized in order to have a simple presentation. A more sophisticated version can be found in [1]. The results are illustrated in Fig.7 for $S_{max} = 7$.

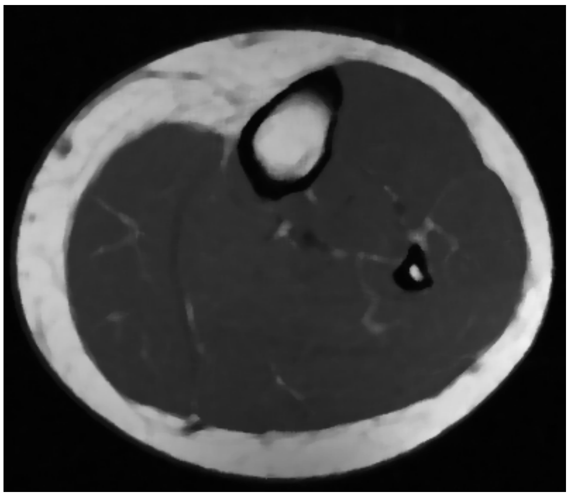
```python
def amf(I, Smax):
    """
    Adaptive median filter
    I: grayscale image
    Smax: maximal size of neighborhood. Limits the effect of median
        ↪ filter
    """
    f = np.copy(I);
    nx, ny = I.shape;

    sizes = np.arange(1, Smax, 2);
    zmin = np.zeros((nx, ny, len(sizes)));
    zmax = np.zeros((nx, ny, len(sizes)));
    zmed = np.zeros((nx, ny, len(sizes)));

    for k,s in enumerate(sizes):
        zmin[:,:,k] = ndimage.minimum_filter(I, s);
        zmax[:,:,k] = ndimage.maximum_filter(I, s);
        zmed[:,:,k] =  ndimage.median_filter(I, s);

    isMedImpulse = np.logical_or(zmin==zmed,zmax==zmed);

    for i in range(nx):
        for j in range(ny):
            k = 0;
            while k<len(sizes)-1 and isMedImpulse[i,j,k] :
                k+=1;

            if I[i,j] == zmin[i,j,k] or I[i,j]==zmax[i,j,k] or k==len(
                ↪ sizes):
                f[i,j] = zmed[i,j,k];
    return f;
```

References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002. 9


(a) Original image.


(b) Noisy image (salt and pepper).


(c) Median filter of size 7×7.

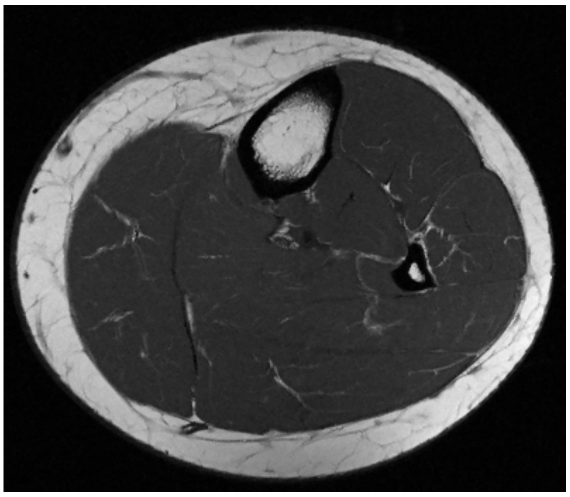

(d) Adaptive median filter, $S_{max} = 7$.

Figure 7: Illustration of the adaptive median filter.