# 1 Matlab correction

## 1.1 Generation of random noises

In order to convert the images into 8bits unsigned variables, the following function can be used:

```
function A=hist_stretch(B)
% histogram stretching.
% ensure the range is [0; 255]
A = B - min(B(:));
A = 255 * A / max(A(:));
A = uint8(A);
```

In order to display the results in Figs.1 and 2, a size $S = 32$ will be used to generate the noisy images.

### 1.1.1 Uniform noise

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

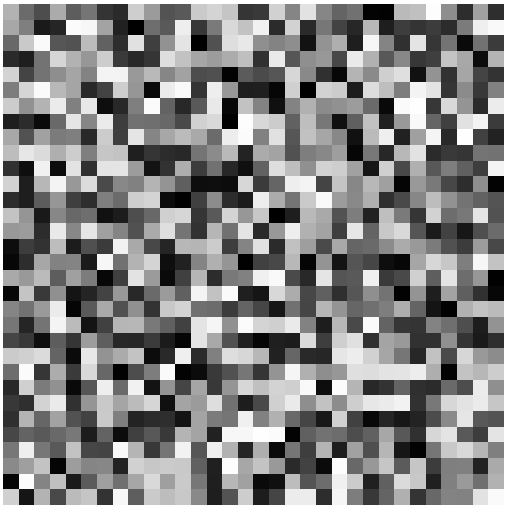
```
S=32; a=0; b=255;
R1=a+(b-a)*rand(S);
```

### 1.1.2 Gaussian noise

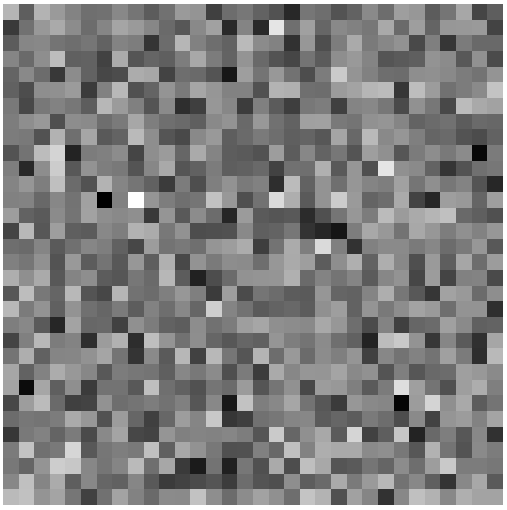
The matlab randn function generates values with normal centered distribution.

```
a=0; b=1;
R2=a+b*randn(S);
R2=hist_stretch(R2);
```

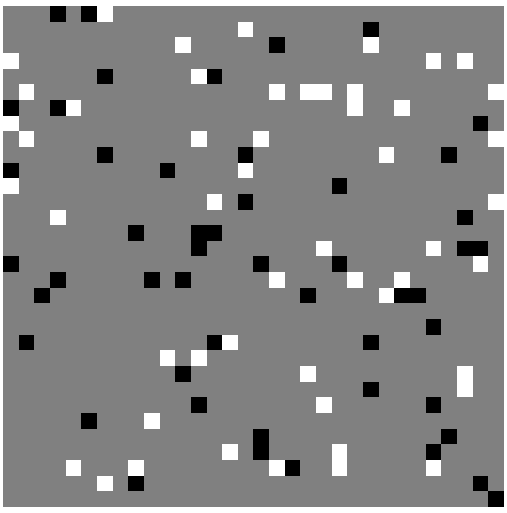
### 1.1.3 Salt and pepper noise



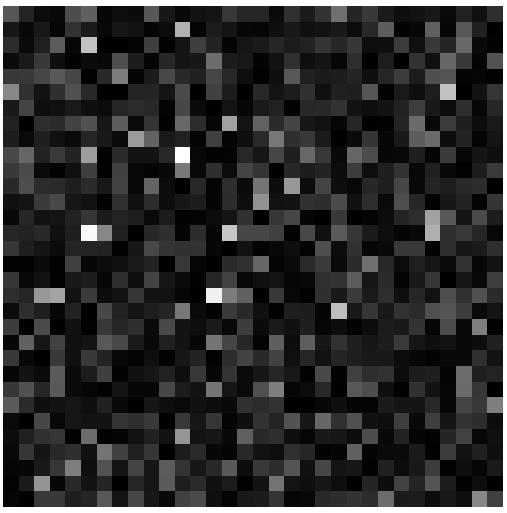
(a) Uniform noise.



(b) Gaussian noise.



(c) Salt and pepper noise.



(d) Exponential noise.

Figure 1: Resulting noise images.

```
a=0.05;b=0.05;
R3=0.5*ones(S);
X=rand(S);
R3(X<=a)=0;
R3(X>a & X<=a+b)=1;
R3=hist_stretch(R3);
```

### 1.1.4 Exponential noise

```
a=1;
R4=-1/a*log(1-rand(S));
R4=hist_stretch(R4);
```

## 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:

```
A=imread('jambe.tif');
A=double(A);
A=A/255;
c=[160 200 200 160];
r=[200 200 240 240];
C=roipoly(A,c,r);
imhist(A(C));

% exponential noise
[m,n]=size(A);
expnoise=1/0.5*log(1-rand(m,n));
B=A-expnoise/max(abs(expnoise(:)));B=hist_stretch(B);
imwrite(B, 'im_exp.png');
figure;
subplot(1,2,1);viewImage(B);title('Image with exponential noise');
subplot(1,2,2);imhist(B(C));title('histogram in ROI');

% Gaussian noise
B=imnoise(A,'gaussian',0,0.004);
imwrite(B, 'im_gauss.png');
figure;
subplot(1,2,1);viewImage(B);title('image with Gaussian noise');
subplot(1,2,2);imhist(B(C));title('histogramin ROI');
```

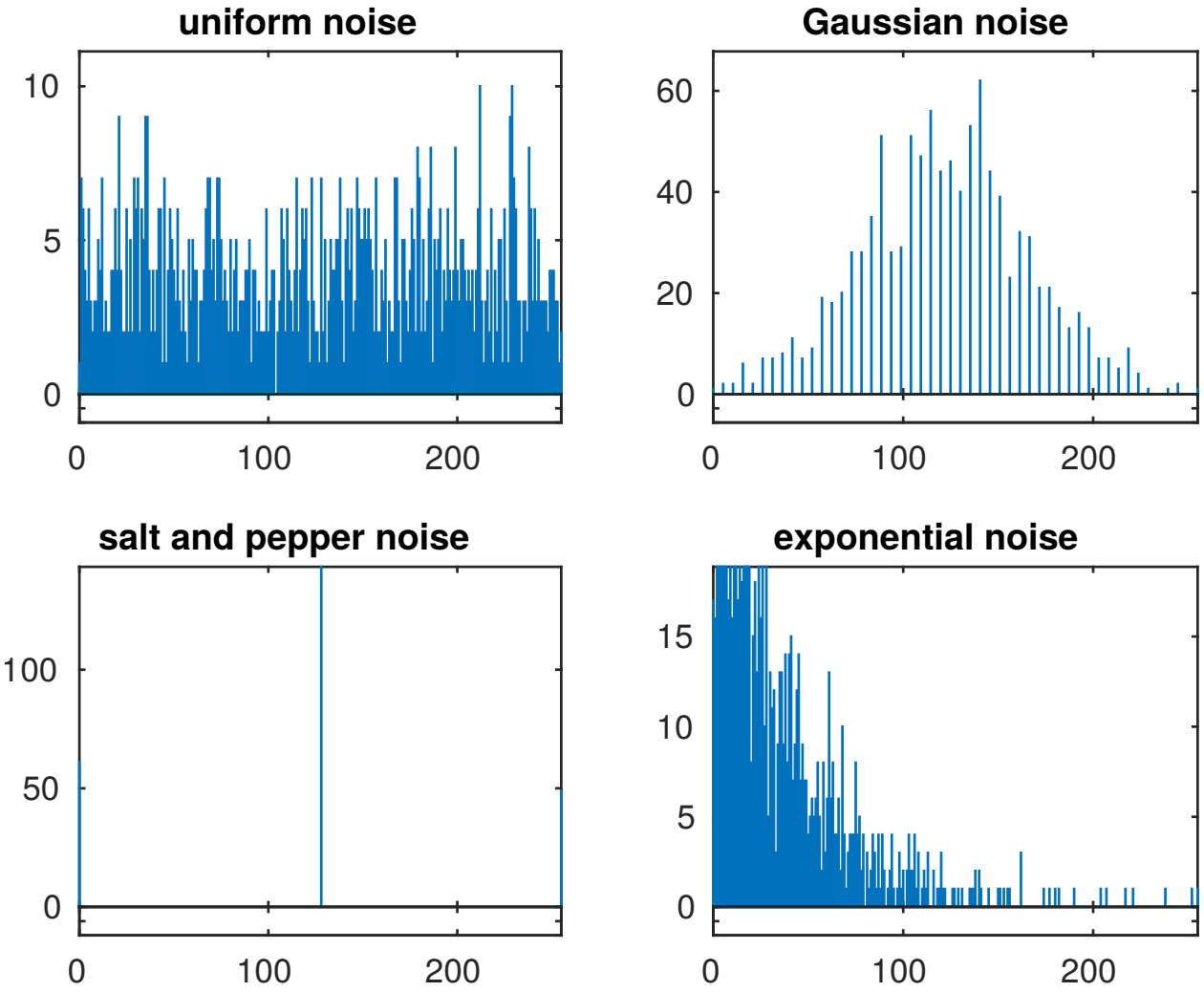
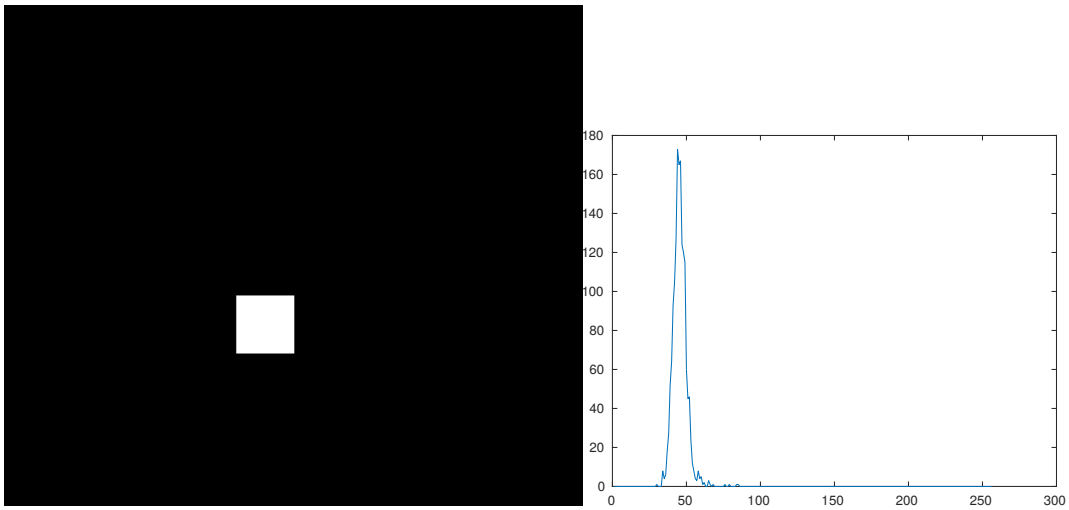



Figure 2: Histograms of the noise images generated with $32 \times 32$ pixels.

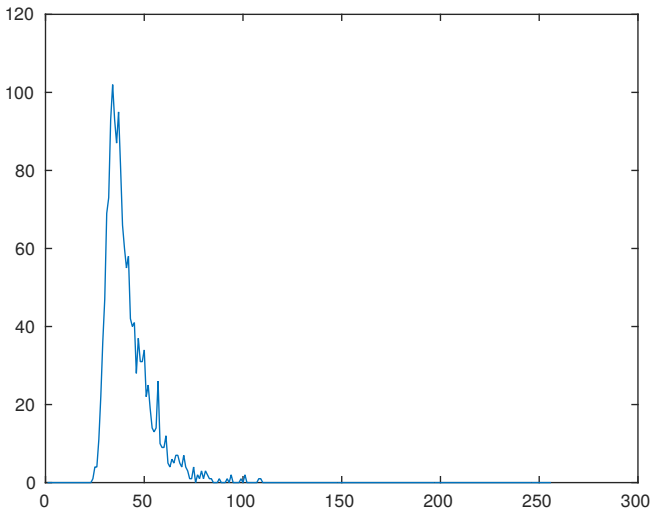


(a) ROI. (b) Histogram.

Figure 3: Histogram of the Region of Interest.

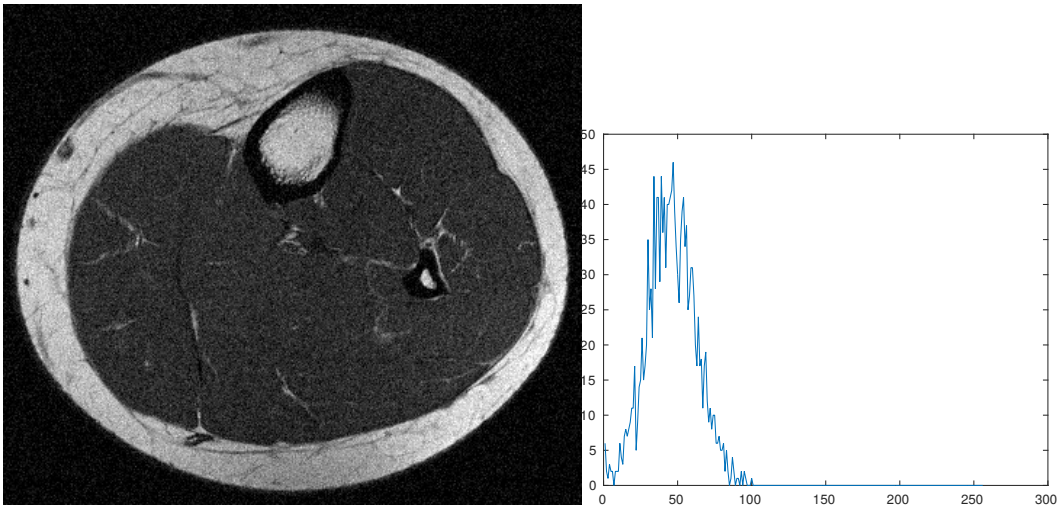
In the case of exponential and Gaussian noise added to the image, The histograms are displayed in Figs.4 and 5.

(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: Exponential noise.

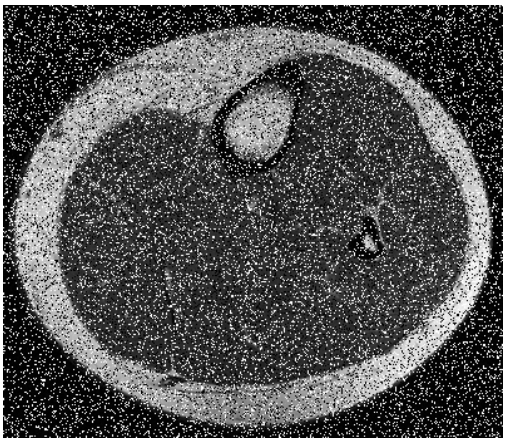
## 1.3 Image restoration by spatial filtering

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

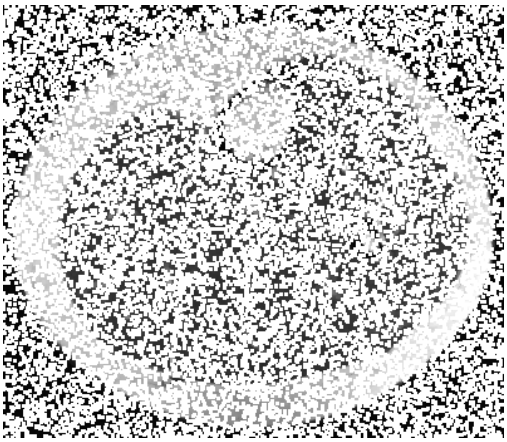
```
A=imread('jambe.tif');
A=double(A);
A=A/255;
[m,n]=size(A);
% add salt and pepper noise
B=imnoise(A,'salt & pepper', 0.25);
% filtering
% mean
w=fspecial('average',5);
B1=imfilter(B,w);
% max
B2=ordfilt2(B,9,ones(3,3));
% min
B3=ordfilt2(B,1,ones(3,3));
% median
B4=medfilt2(B,[7,7]);
```

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

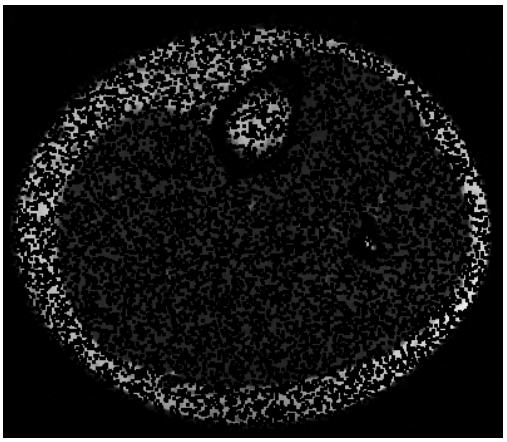
(a) Original image.



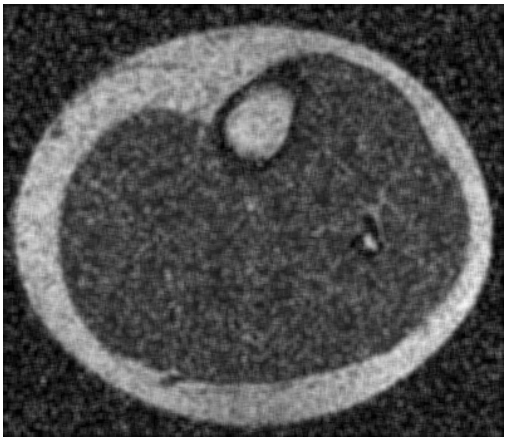
(b) Noisy image (salt and pepper).



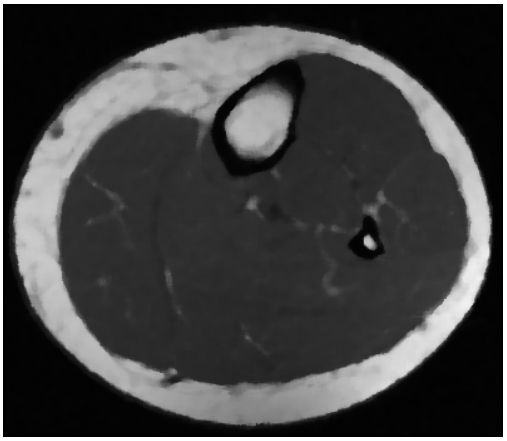
(c) Maximum filter.



(d) Minimum filter.



(e) Mean filter.



(f) Median filter.

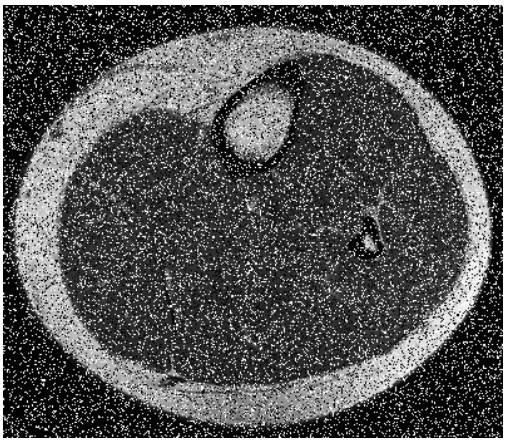
Figure 6: Different filters applied to the noisy image. The median filter is particularly adapted in the case of salt and pepper noise (impulse noise), but still destroy the structures observed in the images.

### 1.3.1 Adaptive median filter

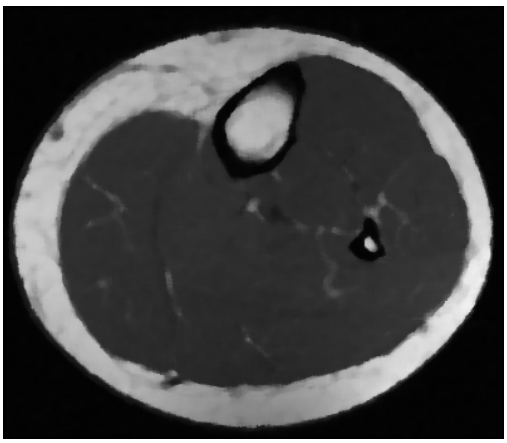
The following code is a simple implementation of the algorithm previously presented. It has not been optimized in order to have a simple presentation. A more sophisticated version can be found in [1]. The results are illustrated in Fig.7 for $S_{max} = 7$.

```
function f = amf(I, Smax)
% adaptive median filter
% I: original image
% Smax: size maxi of neighborhood

f = I;

sizes=1:2:Smax;
zmin = zeros([size(I) length(sizes)]);
zmax = zeros([size(I) length(sizes)]);
zmed = zeros([size(I) length(sizes)]);

for k=1:length(sizes),
    zmin(:,:,k) = ordfilt2(I, 1, ones(sizes(k)), 'symmetric');
    zmax(:,:,k) = ordfilt2(I, sizes(k)^2, ones(sizes(k)), 'symmetric');
    zmed(:,:,k) = medfilt2(I, [sizes(k) sizes(k)], 'symmetric');
end

% determines for all scales at the same time if zmed is an impulse noise.
% this enables the choice of the scale.
isMedImpulse = (zmin==zmed) | (zmax==zmed);

for i=1:size(I,1)
    for j=1:size(I,2)

        % finds the right scale
        % determines k (neighborhood size) where the median value is not
            ↪ an
        % impulse noise
        k=1;
        while isMedImpulse(i,j,k) && k<length(sizes)
            k = k+1;
        end

        % if the value of the pixel I(i,j) is an impulse noise, it is
        % replaced by the median value at scale k, if not, it is kept
        % untouched (already set)
        if I(i,j)==zmin(i,j,k) || I(i,j)==zmax(i,j,k) ||  k == length(
            ↪ sizes)
            f(i,j) = zmed(i,j,k);
        end

    end
end
```

(a) Original image.



(b) Noisy image (salt and pepper).



(c) Median filter of size $7 \times 7$.

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

Figure 7: Illustration of the adaptive median filter.

# References

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