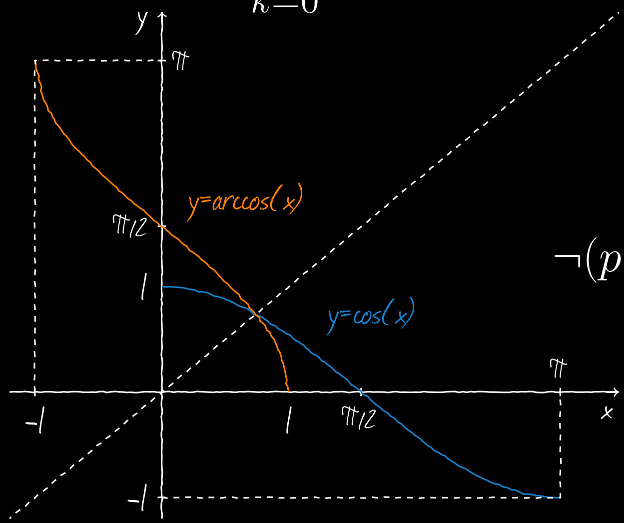
Report for Digital Image Processing

Fanyong Xue

version 1.1

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

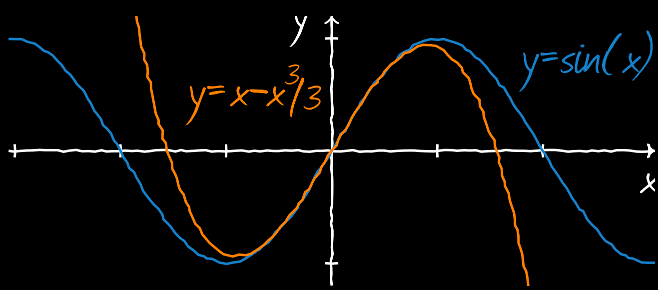


$$\zeta_k = |a|^{1/n} \mathrm{e}^{i(\arg(a)+2k\pi)/n}$$

$$\mathrm{e}^{i\pi} + 1 = 0$$

$$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



Contents

Contents

1.1 OverView

(a) Write a computer program for computing the histogram of an image.
(b) Implement the histogram equalization technique.
(c) Your program must be general to allow any gray-level image as its input.
As a minimum, your report should include the original image, a plot of its histogram, a plot of the transformation function, the enhanced image, and a plot of its histogram.

1.2 Generate the Histogram

1.2.1 Function

Generating the histogram of an image using following function:

$$H(i) = the\ number\ of\ pixel\ whose\ value\ euquals\ to\ i \tag{1.1}$$

1.2.2 Histogram

The histogram pictures of Fig1.jpg and Fig2.jpg are listed as follows:

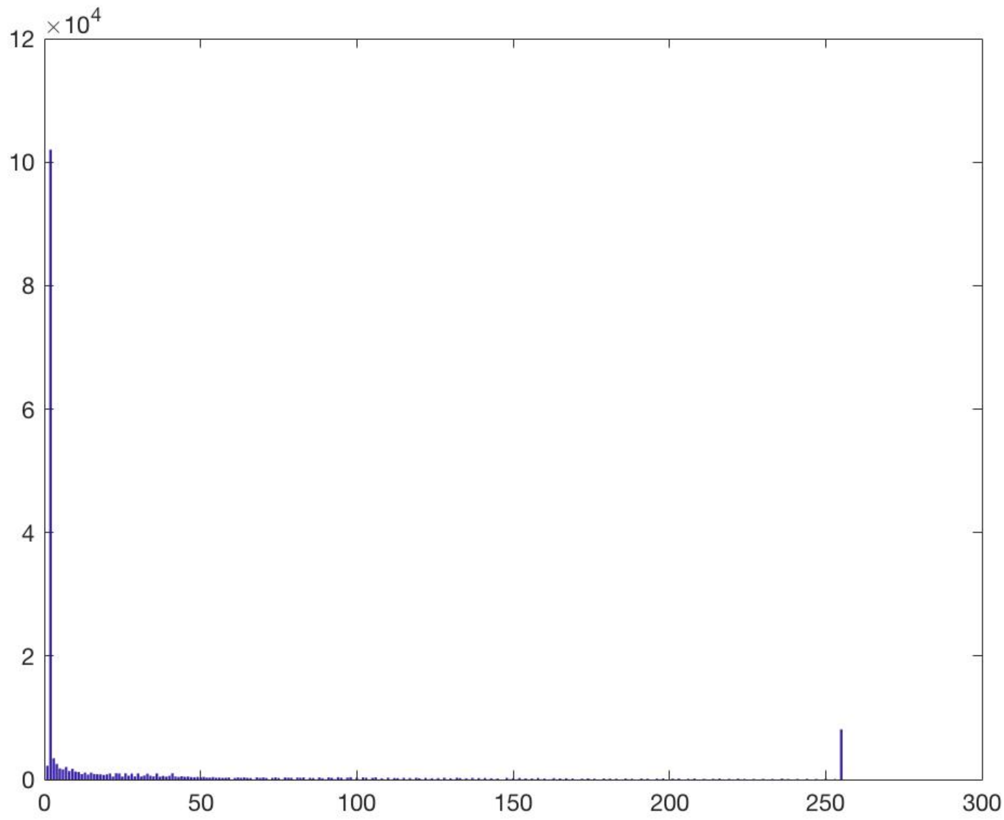


Figure 1.1: Histogram of fig1.jpg

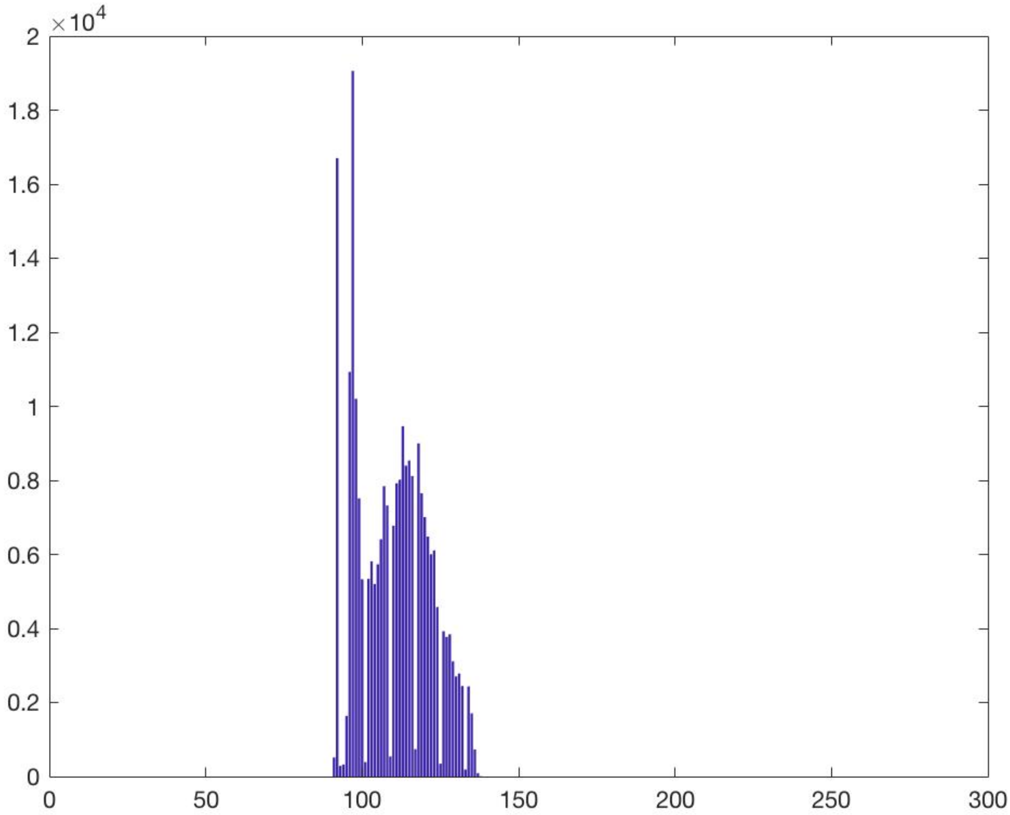


Figure 1.2: Histogram of fig2.jpg

1.3 Transfer Function

1.3.1 Implement the Histogram Equalization Technique

We use those functions to calculate the histogram equalization:

$$L = Max(image(r,c))\ \forall r \in [1, rows]\ and\ \forall c \in [1, cols] \tag{1.2}$$

$$s(r_k) = L * T(r_k) = L * \sum_{j=0}^{k} P_r(r_j) = L * \sum_{j=0}^{k} \frac{n_j}{n} \tag{1.3}$$

1.3.2 Transfer Function

The transfer function of Fig1.jpg and Fig2.jpg are listed as follows:

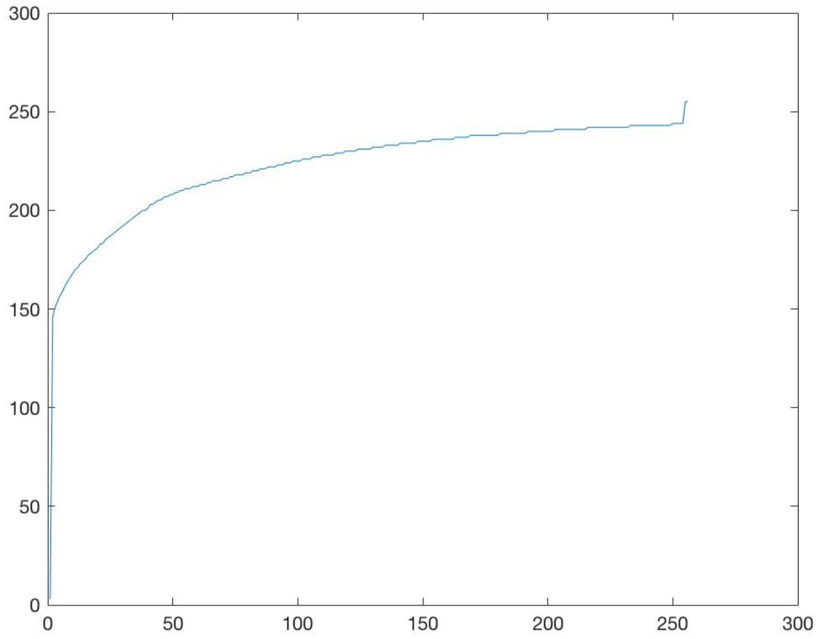


Figure 1.3: Transfer Function of fig1.jpg

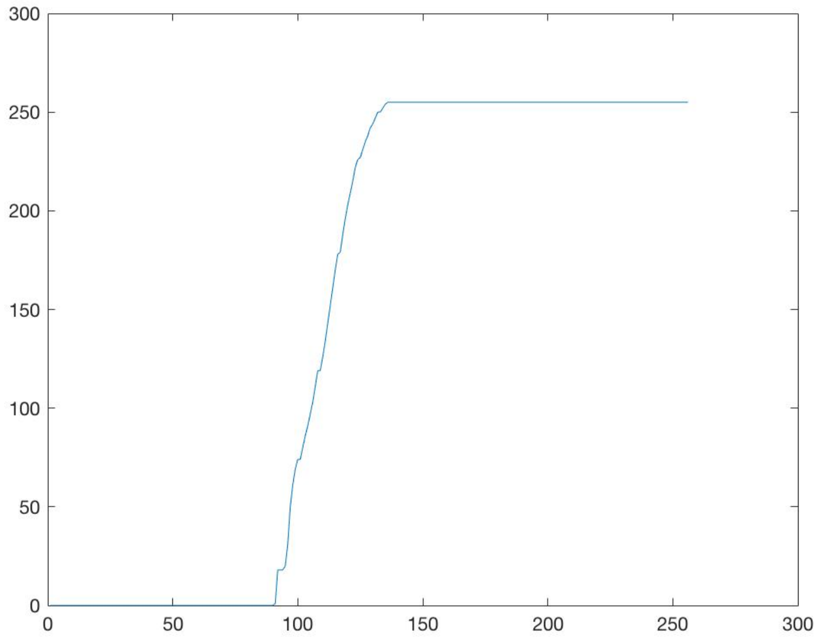


Figure 1.4: Transfer Function of fig2.jpg

1.4 Enhanced Images

1.4.1 Enhanced Function

We use the function

$$New\ Image(r,c) = Transfer\ Function(image(r,c))\ \forall r \in [1, rows]\ and\ \forall c \in [1, cols] \tag{1.4}$$

to enhance the original images.

1.4.2 Enhanced Images

The original images and enhanced images and histogram comparation are listed as follows.

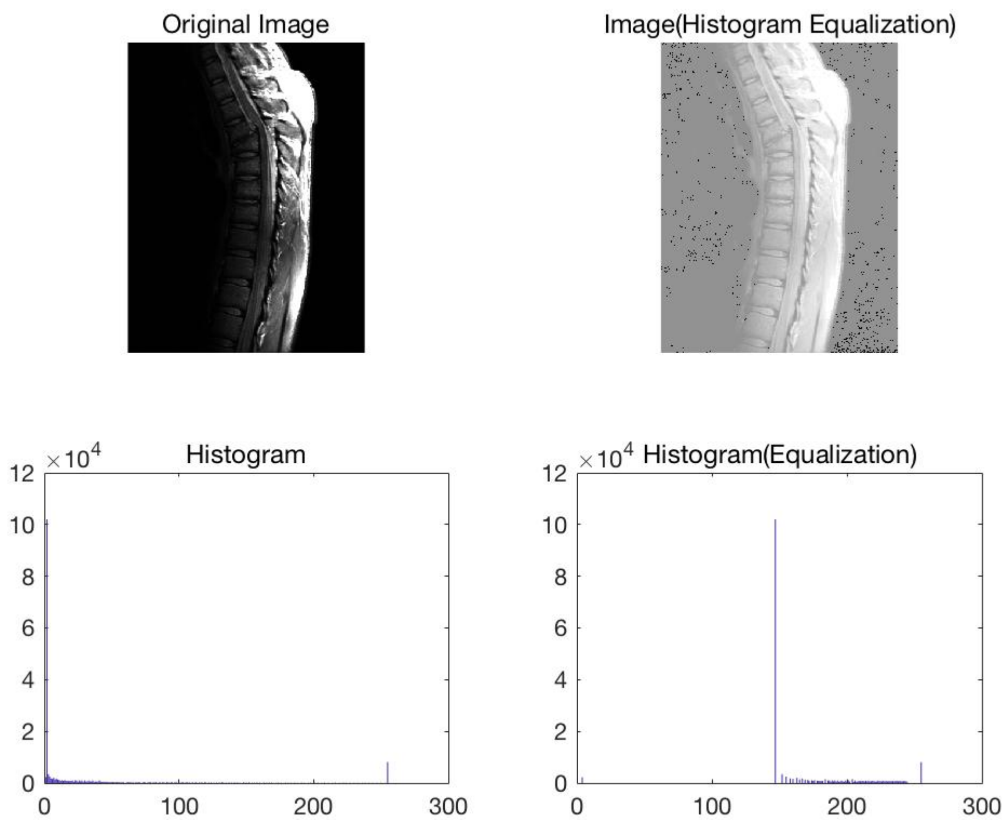


Figure 1.5: original image and enhanced image and histogram comparation of fig1.jpg

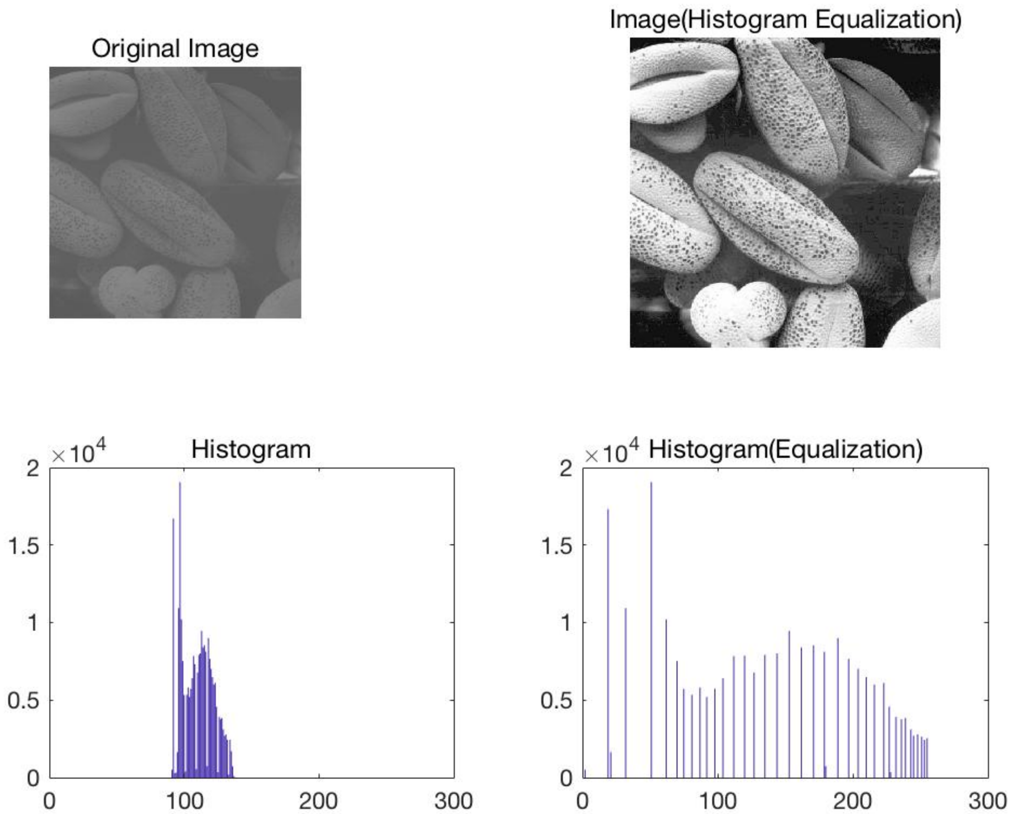


Figure 1.6: original image and enhanced image and histogram comparation of fig2.jpg

2.1 OverView

Implement the image enhancement task of Section 3.7 (Fig 3.43, page 171). The image to be enhanced is skeleton_orig.tif. You should implement all steps in Figure 3.43. (You are encouraged to implement all functions by yourself, not to directly use Matlab functions such as imfilter or fspecial.)

2.2 Image b

2.2.1 Laplacian Transform Filter

We apply Laplacian Transform on the original image to get the image (b) using the following filter.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

2.2.2 Laplacian Transform

Image b:

Figure 2.1: Laplacian Transform of skeleton_orig.tif

2.3 Image c

The we add the Laplacian of the original image to the original image, we will get the new image c. The new image c is a rather noisy sharpened image.
Image c:

Figure 2.2: Laplacian Transform of skeleton_orig.tif

2.4 Image d

2.4.1 Sobel Gradient Masks

We will use two mask to separately get the components g_x and g_y. Then add the two components together, we will get the the sober gradient of the original image. The new image is as follows. As we can see, edges are much more dominant in this image than in the Laplacian image.

g_x:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

g_y:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

2.4.2 Image d

Image d:

Figure 2.3: Laplacian Transform of skeleton_orig.tif

2.5 Image e

Image e is formed by smoothing image d by 5*5 mean filter.
Image e:

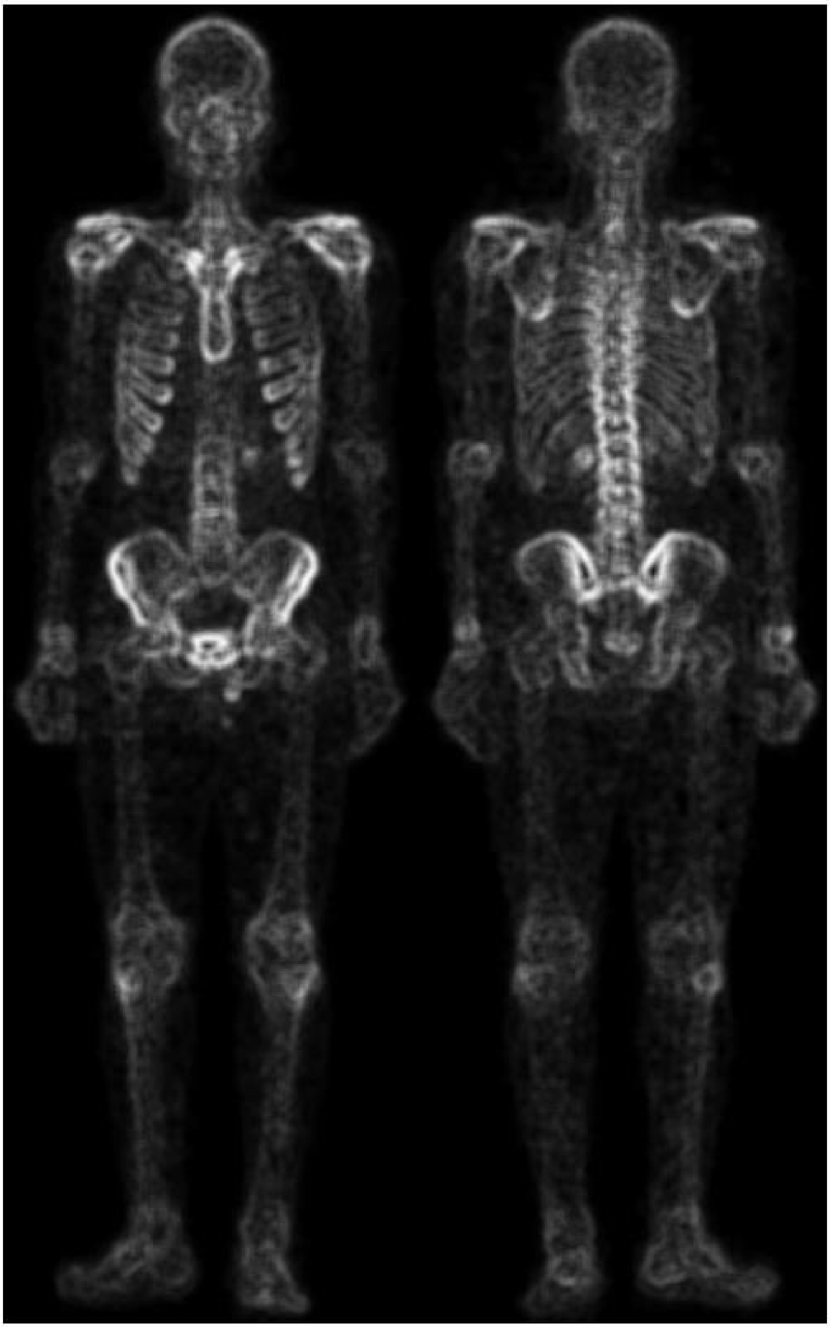


Figure 2.4: Laplacian Transform of skeleton_orig.tif

2.6 Image f

Image f is formed by the product of Laplacian and smoothed-gradient image. The dominance of the strong edges and the relative lack of visible noise, which is the key objective behind masking the Laplacian with a smoothed gradient image.
Image f:

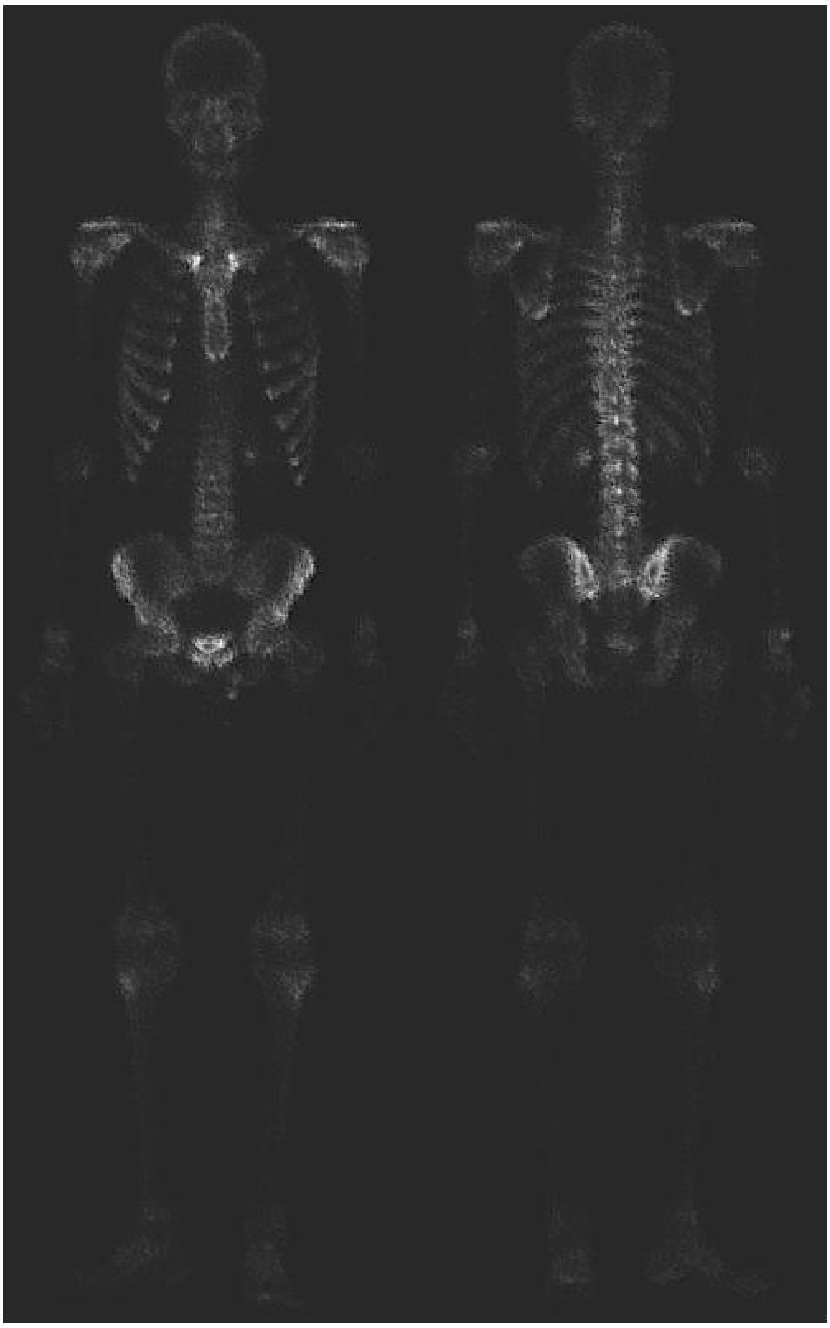


Figure 2.5: Laplacian Transform of skeleton_orig.tif

2.7 Image g

Adding the image f to the original image and then we get image g.
image g:

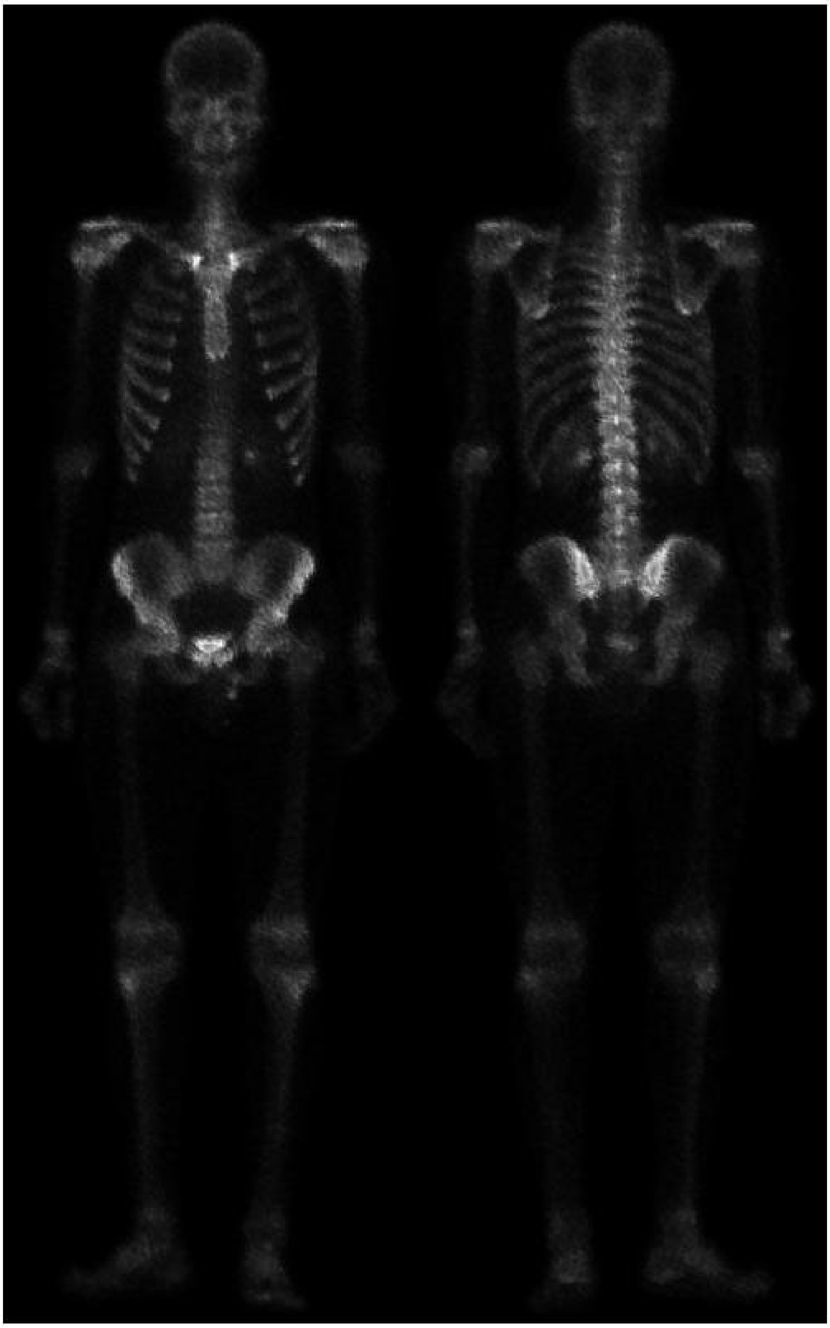


Figure 2.6: Laplacian Transform of skeleton_orig.tif

2.8 Image h

2.8.1 Power-Law Transformation

We use the following function to perform Power-Law Transformation on image g.

$$s = cr^{\gamma} \ (c \ = \ 1 \ and \ \gamma \ = \ 1) \tag{2.1}$$

2.8.2 Image h

Image h:

Figure 2.7: Laplacian Transform of skeleton_orig.tif

3.1 OverView

Implement the ideal, Butterworth and Gaussian lowpass and highpass filters and compare the results under different parameters using the image characters_test_pattern.tif (this image file can be found at the ftp server ftp://ftp.cs.sjtu.edu.cn:990/lu-ht/DIP/images) as the test pattern.

3.2 Fourier Transform

3.2.1 Transform Function

We first need transform the image to gray image and then use the 2D Fourier transformation belw to change it to the frequence domain.

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.1}$$

$$f(x,y)=\frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.2}$$

3.2.2 Fast Fourier Transform

The inverse funtion of FFT is as folows:

$$F(u+\frac{M}{2},v+\frac{N}{2})=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(\frac{ux+x\frac{M}{2}}{M}+\frac{vy+v\frac{N}{2}}{N})} \tag{3.3}$$

$$=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}*e^{-j\pi x}*e^{-j\pi y} \tag{3.4}$$

$$=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}-1^{x+y}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.5}$$

3.3 Filtering

- Expand image$(M*N)$ to image a$((2*M)*(2*N))$
- Perform the fourier transform on the new image and get image b
- Shift image b and get image c
- Perform the convolution on image c and filter and get image d
- Shift image d back and get image e
- Perform the inverse fourier transform on image e and get image f
- Cut image f to get the final result image$(M*N)$;

3.4 Ideal Filter

3.4.1 Ideal Low Pass

Ideal Low Pass Filter

The ideal low pass filter is defined as follows:

$$H(u,v) = \begin{cases} 1, & D(u,v) \le 0 \\ 0, & D(u,v) > 0 \end{cases} \tag{3.6}$$

$$D(u,v) = \sqrt{[(u-\frac{P}{2})^2 + (v-\frac{Q}{2})^2]} \tag{3.7}$$

Result

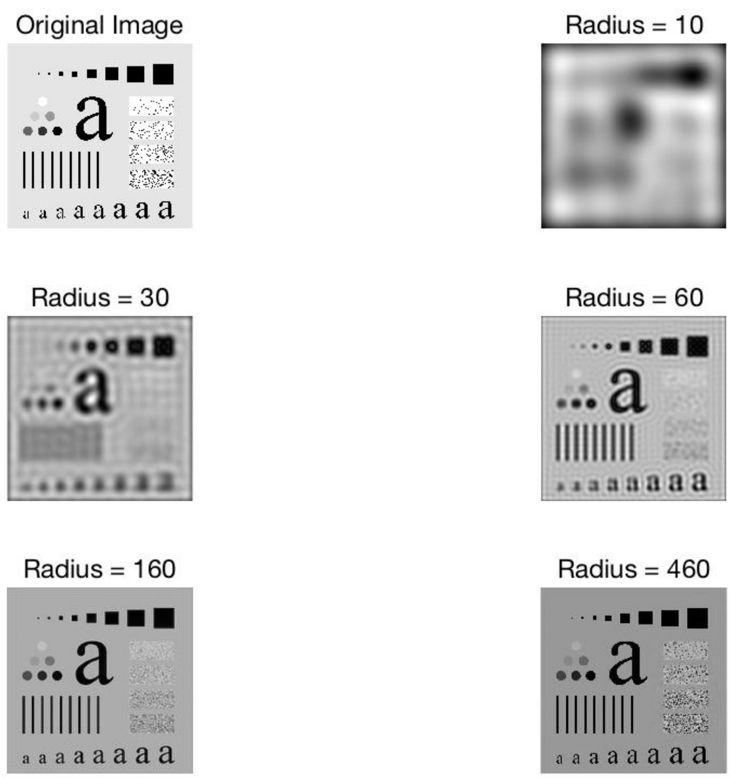


Figure 3.1: ILPF of characters_test_pattern.tif

3.4.2 Ideal High Pass

Ideal High Pass Filter

The ideal high pass filter is defined as follows:

$$H(u,v) = \begin{cases} 0, & D(u,v) \leq 0 \\ 1, & D(u,v) > 0 \end{cases} \tag{3.8}$$

$D(u,v)$ is defined in (3.7)

Result

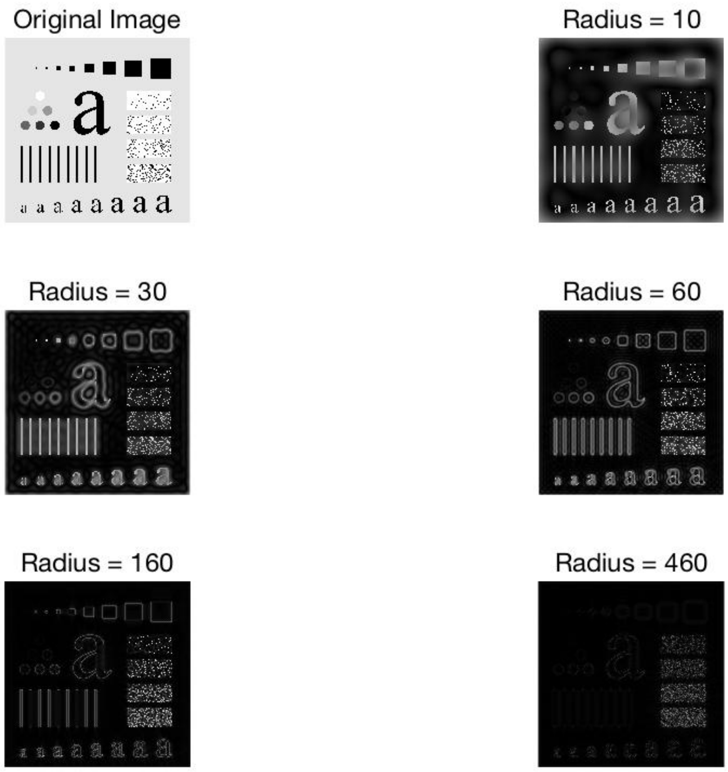


Figure 3.2: ILPF of characters_test_pattern.tif

3.5 Butterworth Filter

3.5.1 Butterworth Low Pass

Butterworth Low Pass Filter

The butterworth low pass filter is defined as follows:

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{3.9}$$

$D(u,v)$ is defined in (3.7)

Result

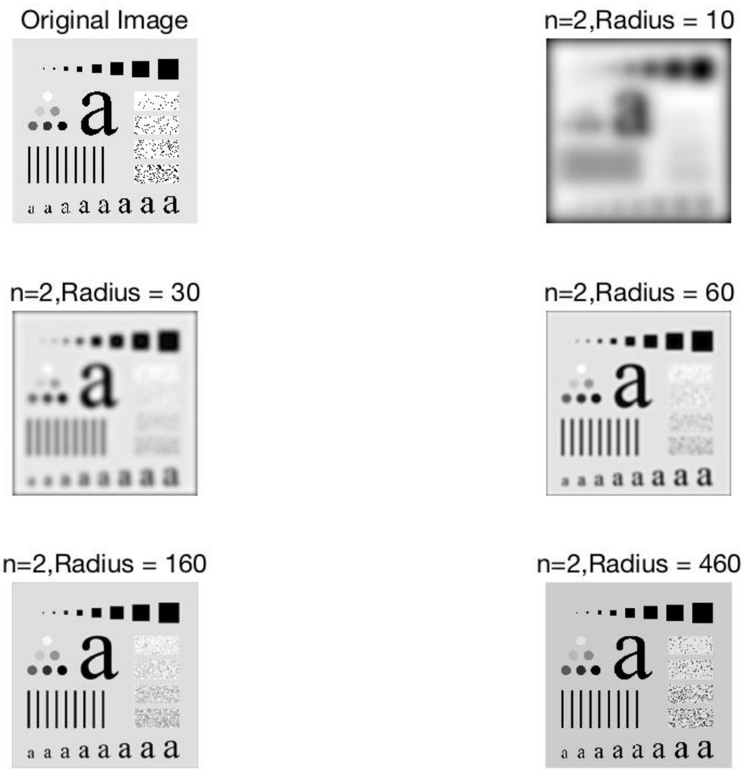


Figure 3.3: BLPF of characters_test_pattern.tif

3.5.2 Butterworth High Pass

Butterworth High Pass Filter

The butterworth high pass filter is defined as follows:

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}} \tag{3.10}$$

$D(u, v)$ is defined in (3.7)

Result

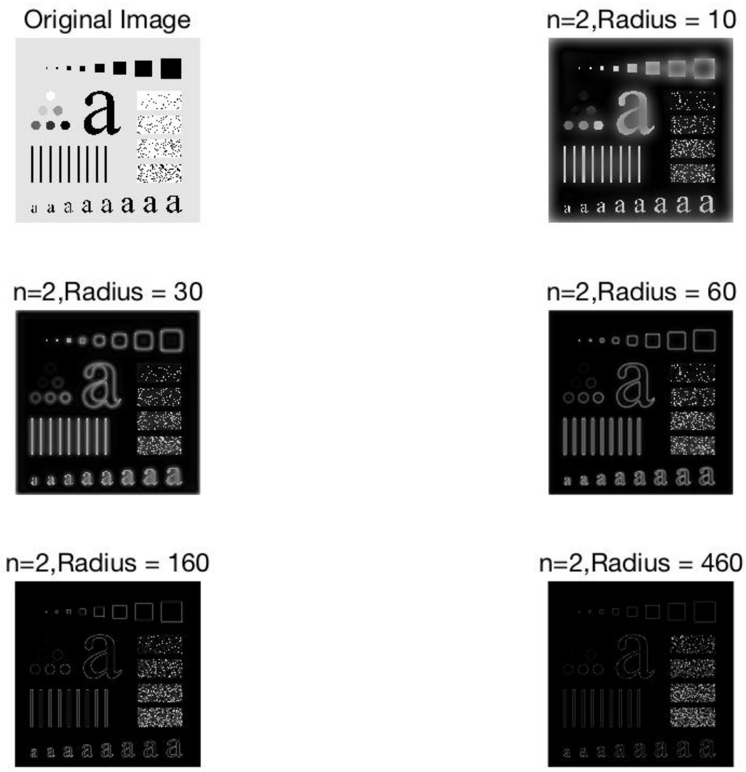


Figure 3.4: BLPF of characters_test_pattern.tif

3.6 Gaussian Filter

3.6.1 Gaussian Low Pass

Gaussian Low Pass Filter

The gaussian low pass filter is defined as follows:

$$H(u,v) = e^{-\frac{D^2(u,v)}{2\sigma^2}} \tag{3.11}$$

$D(u,v)$ is defined in (3.7)

Result

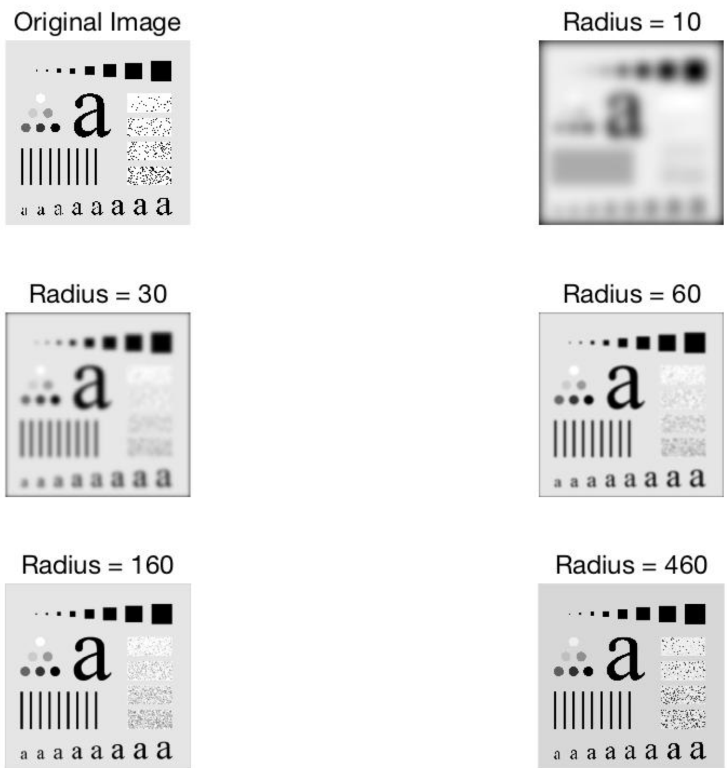


Figure 3.5: GLPF of characters_test_pattern.tif

3.6.2 Gaussian High Pass

Gaussian High Pass Filter

The gaussian high pass filter is defined as follows:

$$H(u,v) = 1 - e^{-\frac{D^2(u,v)}{2\sigma^2}} \tag{3.12}$$

$D(u,v)$ is defined in (3.7)

Result

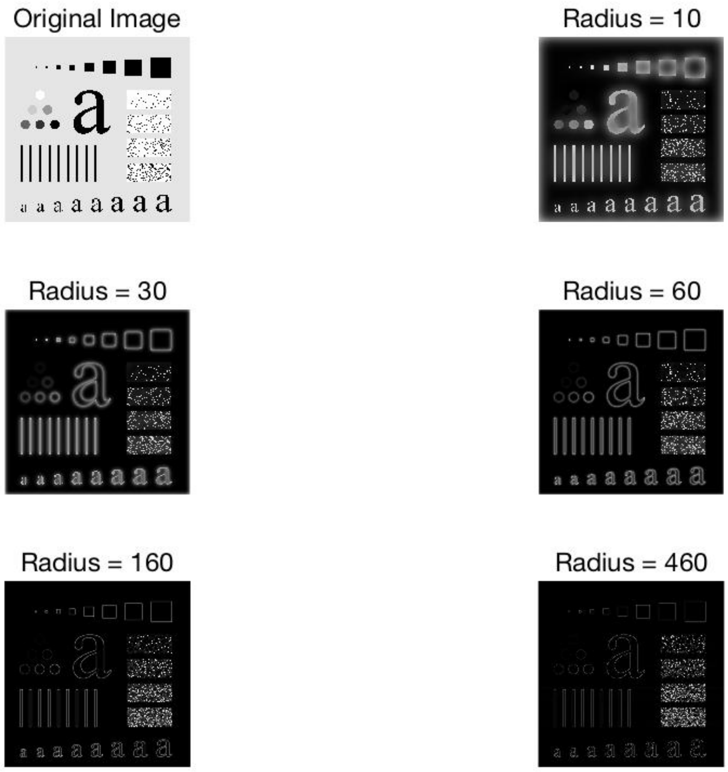


Figure 3.6: GLPF of characters_test_pattern.tif

4.1 OverView

In this problem, you are required to write a program to generate different types of random noise (Uniform, Gaussian, Rayleigh, Gamma, Exponential and Impulse, first started from the uniform noise and then use some functions to convert the uniform noise to Gaussian, Rayleigh, Gamma and Exponential; Impulse noise is generated in a different way, consulting the textbook and some other references) and then add these noises to the test patter image Fig0503(original_pattern).tif to compare the visual results of the noisy images.
Add some of these noises to the circuit image Circuit.tif (images can be found at ftp://ftp.cs.sjtu.edu.cn:990/lu-ht/DIP/images) and investigate the noise reduction results using different mean filters and order statistics filters as the textbook did at pages 344-352 (Pages 322-329 in the electronic version of the textbook).

4.2 Noise

4.2.1 Uniform Noise

The PDF of $uniformnoise$ is given by

$$p(z) = \begin{cases} \frac{1}{b-a}, & if\ a \le z \le b \\ 0, & otherwise \end{cases} \tag{4.1}$$

The mean of this density function is given by

$$\overline{z} = \frac{a+b}{2} \tag{4.2}$$

and its variance by

$$\sigma^2 = \frac{(b-a)^2}{12} \tag{4.3}$$

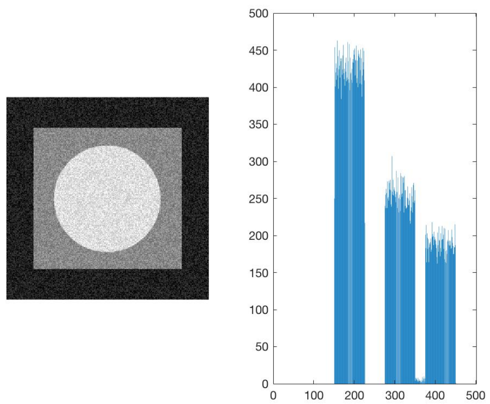


Figure 4.1: Uniform Noise

4.2.2 Gaussian Noise

The PDF of a $Gaussian$ random variable, z, is given by

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\bar{z})^2}{z\sigma^2}} \tag{4.4}$$

and it can generate form uniform noise by

$$X = \sqrt{-2lnU}cos(2\pi V) \tag{4.5}$$
$$Y = \sqrt{-2lnU}sin(2\pi V) \tag{4.6}$$

U and V come form uniform(0,1).

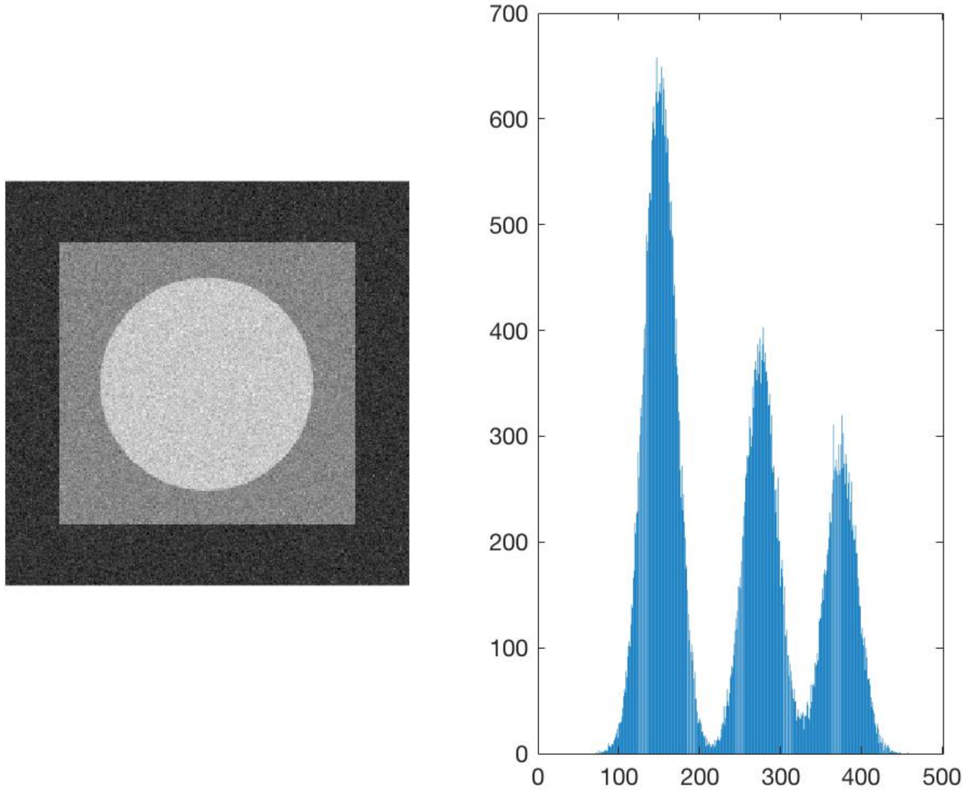


Figure 4.2: Gaussian Noise

4.2.3 Rayleigh Noise

The PDF of $Rayleighnoise$ is given by

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{-\frac{(z-a)^2}{b}}, & for\ z \geq a \\ 0, & for\ z < a \end{cases} \tag{4.7}$$

The mean and variance of this density are given by

$$\overline{z} = a + \sqrt{\frac{b\pi}{4}} \tag{4.8}$$

and

$$\sigma^2 = \frac{b(4-\pi)}{4} \tag{4.9}$$

and it can generate form uniform noise by

$$z = a + \sqrt{-bln[1-U(0,1)]} \tag{4.10}$$

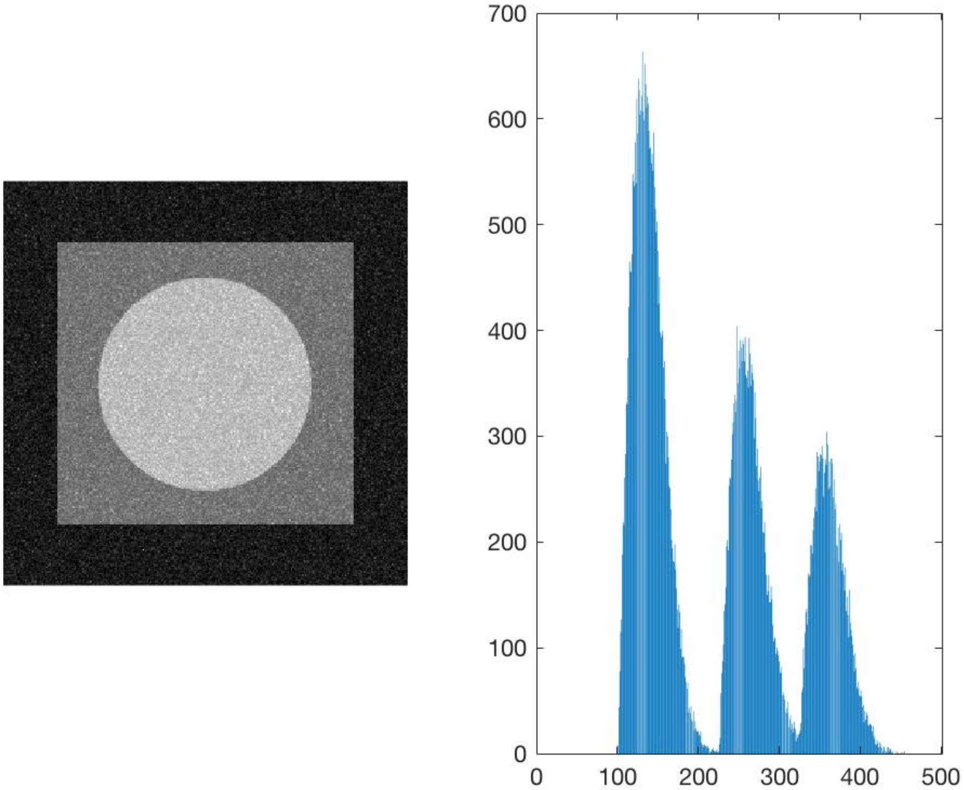


Figure 4.3: Rayleigh Noise

4.2.4 Erlang(Gamma) Noise

The PDF of $Erlang noise$ is given by

$$p(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)!} e^{-az}, & for\ z \geq 0 \\ 0, & for\ z < 0 \end{cases} \tag{4.11}$$

The mean and variance of this density are given by

$$\overline{z} = \frac{b}{a} \tag{4.12}$$

and

$$\sigma^2 = \frac{b}{a^2} \tag{4.13}$$

and it can generate form uniform noise by

$$E_i = -\frac{1}{a} ln[1 - U(0,1)] \tag{4.14}$$

$$z = E_1 + E_2 + \cdots + E_b \tag{4.15}$$

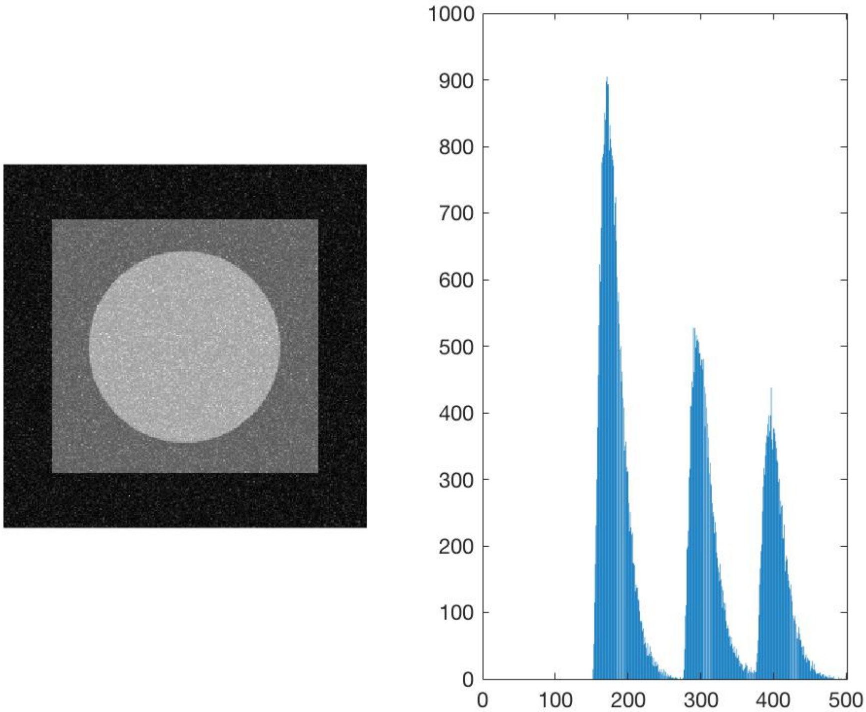


Figure 4.4: Gamma Noise

4.2.5 Exponential Noise

The PDF of $exponentialnoise$ is given by

$$p(z) = \begin{cases} ae^{-az}, & for\ z \geq 0 \\ 0, & for\ z < 0 \end{cases} \tag{4.16}$$

The mean and variance of this density function are

$$\overline{z} = \frac{1}{a} \tag{4.17}$$

and

$$\sigma^2 = \frac{1}{a^2} \tag{4.18}$$

and it can generate form uniform noise by

$$z = -\frac{1}{a} ln[1 - U(0,1)] \tag{4.19}$$

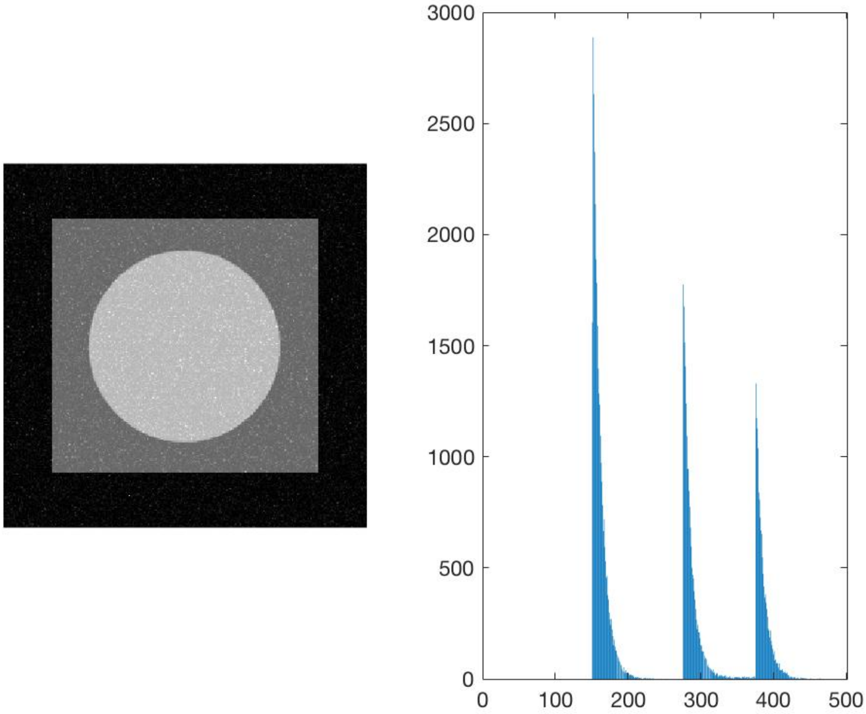


Figure 4.5: Exponential Noise

4.2.6 Impulse Noise

The PDF of $(bipolar) impulse noise$ is given by

$$p(z) = \begin{cases} P_a, & for\ z = a \\ P_b, & for\ z = b \\ 0, & otherwise \end{cases} \tag{4.20}$$

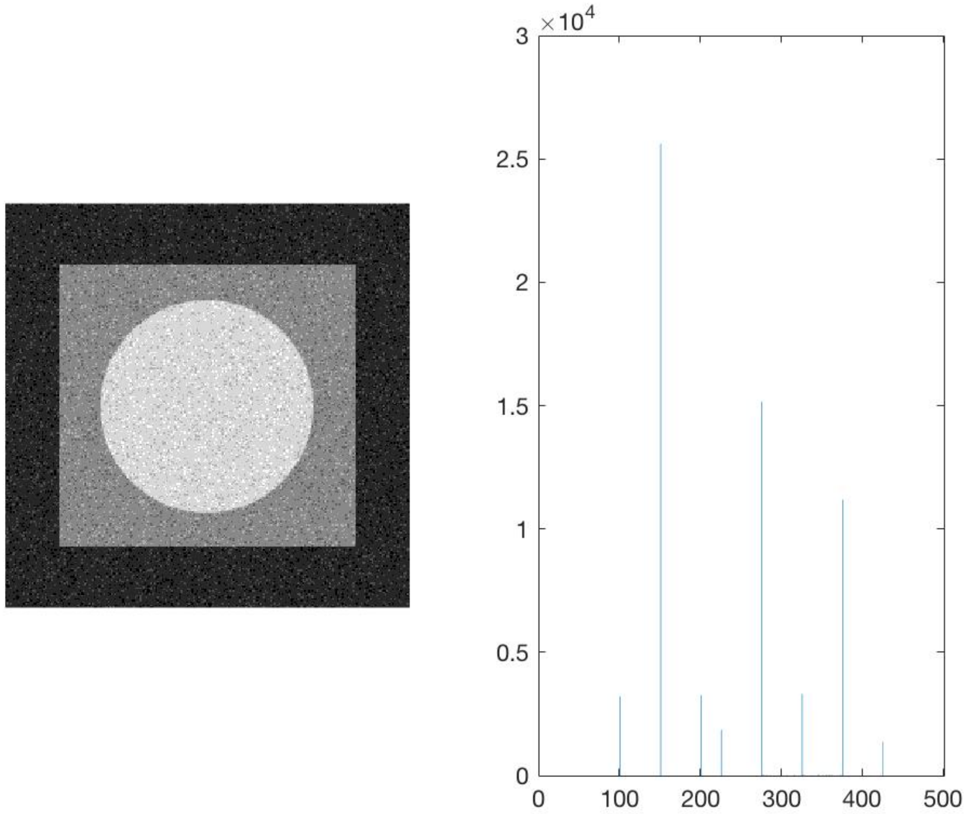


Figure 4.6: Impulse Noise

4.3 Mean Filters

4.3.1 Filters

Arithmetic mean filter

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t) \tag{4.21}$$

Geometric mean filter

$$\hat{f}(x,y) = [\prod_{(s,t)\in S_{xy}} g(s,t)]^{\frac{1}{mn}} \tag{4.22}$$

Harmonic mean filter

$$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}} \tag{4.23}$$

Contraharmonic mean filter

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}} \tag{4.24}$$

4.3.2 Results

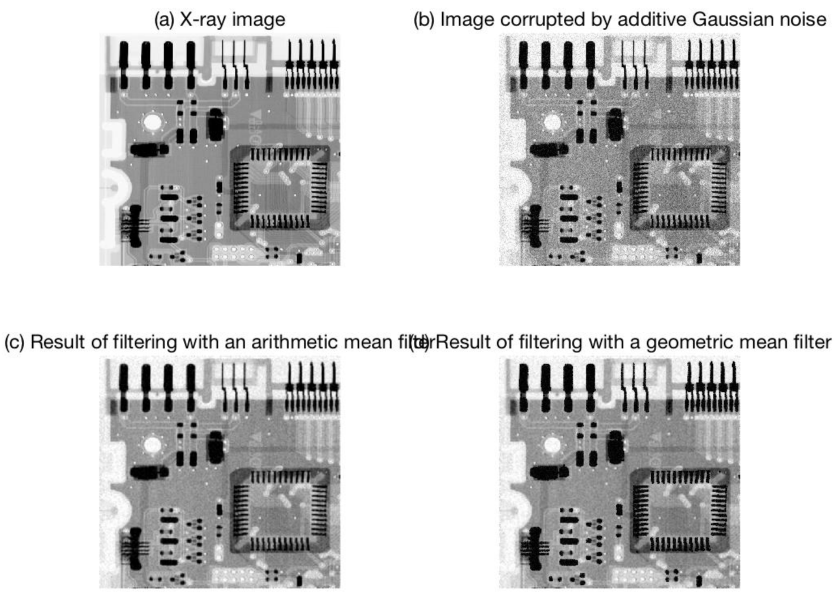


Figure 4.7: Original Book FIGURE 5.7

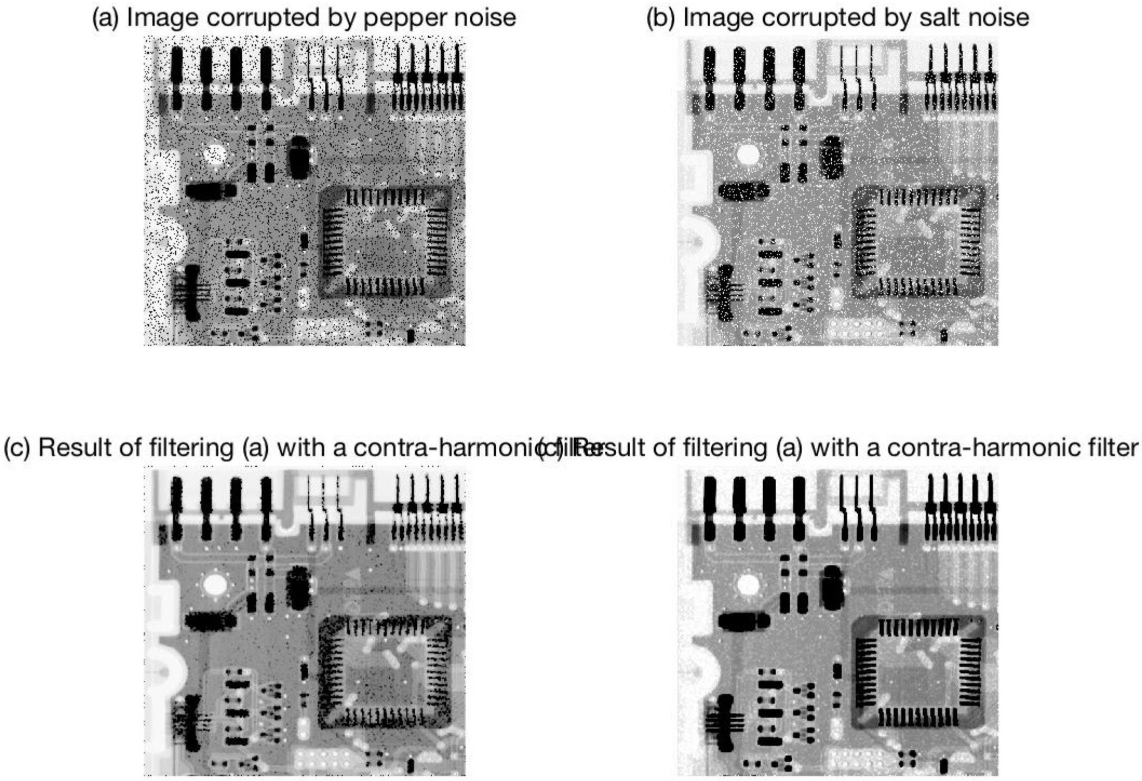


Figure 4.8: Original Book FIGURE 5.8

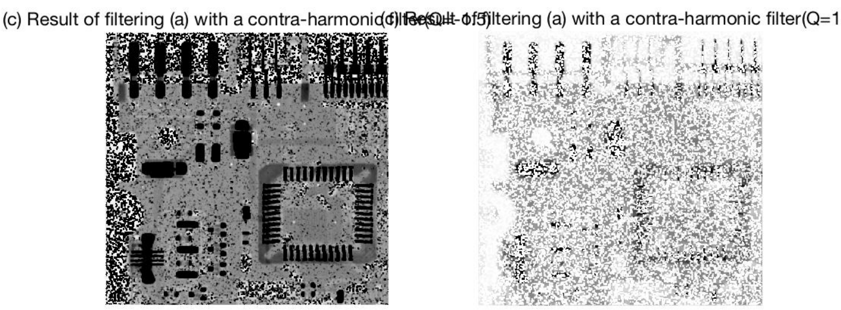


Figure 4.9: Original Book FIGURE 5.9

4.4 Order-Statistic Filters

4.4.1 Filters

Median filter

$$\hat{f}(x, y) = median_{(s,t)\in S_{xy}} g(s, t) \tag{4.25}$$

Max filters

$$\hat{f}(x, y) = max_{(s,t)\in S_{xy}} g(s, t) \tag{4.26}$$

Min filters

$$\hat{f}(x, y) = min_{(s,t)\in S_{xy}} g(s, t) \tag{4.27}$$

Midpoint filter

$$\hat{f}(x, y) = \frac{1}{2}[max_{(s,t)\in S_{xy}} g(s, t) + min_{(s,t)\in S_{xy}} g(s, t)] \tag{4.28}$$

Alpha-trimmed mean filter

$$\hat{f}(x, y) = \frac{1}{mn - d} \sum_{(s,t)\in S_{xy}} g_r(s, t) \tag{4.29}$$

4.4.2 Results

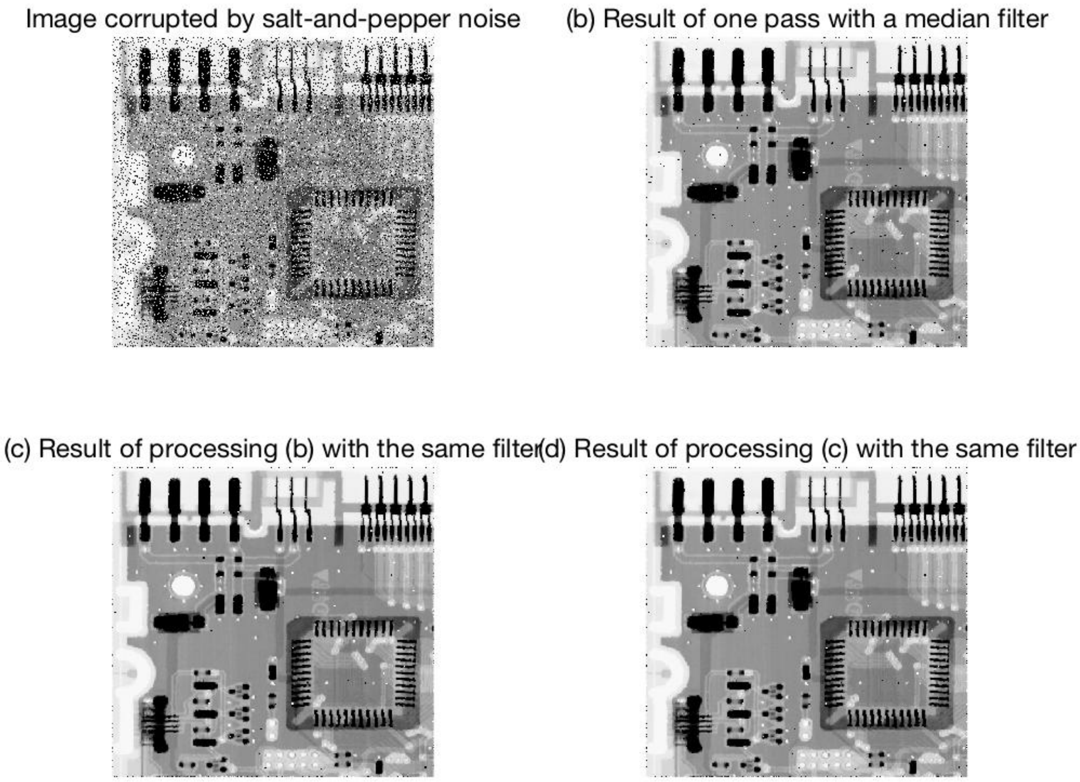


Figure 4.10: Original Book FIGURE 5.10

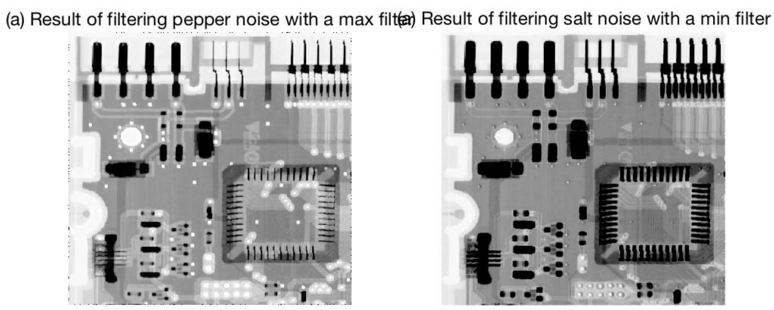
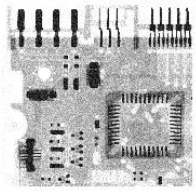
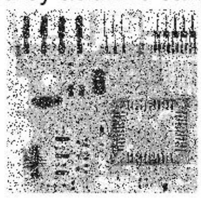


Figure 4.11: Original Book FIGURE 5.11

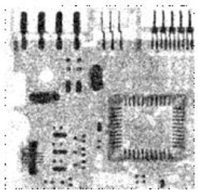
(a) Image corrupted by additive uniform noise



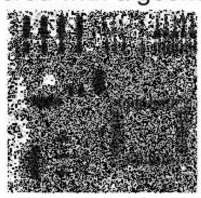
(b) Image corrupted by additive salt-and-pepper noise



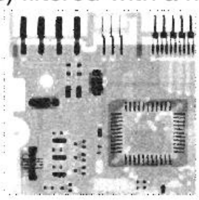
(c) Image (b) filtered with an arithmetic mean filter



(c) Image (b) filtered with a geometric mean filter



(c) Image (b) filtered with a median filter



(c) Image (b) filtered with an alpha-trimmed mean filter

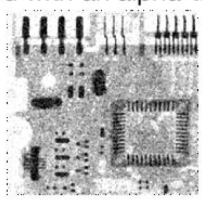


Figure 4.12: Original Book FIGURE 5.12

5.1 OverView

5.2 This is a section

5.2.1 This is a subsection

5.2.2 This is a subsection

A.1 Problem1

```
% Problem 1
% by Xue Fanyong
% Student ID:515030910443
% Histogram Equalizatio

%% Main Part
image1 = imread('Image Path/Fig1.jpg');
image2 = imread('Image Path/Fig2.jpg');

[histogram1,histogram_e1,transfer_f1,image_e1] =
histogram_equalization(image1);
[histogram2,histogram_e2,transfer_f2,image_e2] =
histogram_equalization(image2);

plot_data(image1,image_e1,histogram1,histogram_e1,transfer_f1);
plot_data(image2,image_e2,histogram2,histogram_e2,transfer_f2);

%% Functions Part

% get histogram of image
% image: get histogram of it
% histogram: the histogram of image
function histogram = get_histogram(image)
    histogram = zeros(256,1);
    [row,col]=size(image);
    for r = 1:row
        for c = 1:col
            gray = image(r,c);
            histogram(gray+1)=histogram(gray+1)+1;
        end
    end
end

% do the histogram_equalization for image
% image: do histogram_equalization for it
% histogram: original histogram; histogram_e:
% histogram after histogram
% equalizatio; transfer_f: transfer function;
% image_e: image after histogram
% equalizatio
function [histogram,histogram_e,transfer_f,image_e] =
histogram_equalization(image)
    [row,col]=size(image);
    transfer_f = zeros(256,1);
    histogram = get_histogram(image);
    transfer_f(1) = 256*histogram(1)/(row*col);

    for i = 2:256
        transfer_f(i) = transfer_f(i-1)+255*histogram(i)/(row*col);
```

```
    end
    transfer_f = round(transfer_f);

    image_e = image;
    for r = 1:row
        for c = 1:col
            image_e(r,c)=transfer_f(image(r,c)+1);
        end
    end
    histogram_e = get_histogram(image_e);
end

% plot data
% image:original image; image_e:
% image after histogram equalizatio;
% histogram: original histogram;
% histogram_e: histogram after histogram equalizatio;
% transfer_f: transfer function
function plot_data(image,image_e,histogram,histogram_e,transfer_f)
    figure();
    subplot(2,3,1);
    imshow(image);
    title(" Original Image");
    subplot(2,3,2);
    imshow(image_e);
    title(" Image(Histogram Equalization)");
    subplot(2,3,3);
    bar(histogram);
    title(" Histogram");
    subplot(2,3,4);
    bar(histogram_e);
    title(" Histogram(Equalization)");
    subplot(2,3,5);
    plot(transfer_f);
    title(" Transfer Funciton");
end
```

A.2 Problem 2

```
% Problem 2
% by Xue Fanyong
% Student ID:515030910443
% Combining spatial enhancement methods

%% Main Part
image = imread(' Image Path/skeleton_orig.tif');
[row,col] = size(image);
mask = [-1 -1 -1;-1 8 -1;-1 -1 -1];
mask = double(mask);
```

```
b_image = laplace_transformations(image,mask);
c_image = b_image+im2double(image);
d_image = sobel_gradient(image);
e_image = smooth(d_image);
f_image = im2double(e_image).*c_image;
g_image = abs(f_image)+im2double(image);
h_image = sqrt(g_image);
plot_data(image,b_image,c_image,d_image,
          e_image,f_image,g_image,h_image);

%% Function Part

% Laplace Transfromation for image using mask
% Input:
%    image:image you want to perform
%    mask:Laplace mask you want to use
% Output:
%    image_l:image after laplace transformation

function image_l = laplace_transformations(image,mask)
    [row,col] = size(image);
    mask = double(mask);
    %append image
    image_l = im2double(image);
    image = [zeros(row,2) image zeros(row,2)];
    image = [zeros(2,col+4);image;zeros(2,col+4)];
    image_append = im2double(image);

    for r = 1:row
        for c = 1:col
            image_l(r,c) = sum(sum(image_append(r:r+2,c:c+2).*mask));
        end
    end
end
%{
    sobel gradient for image
%}
function image_s = sobel_gradient(image)
    [row,col] = size(image);
    x_mask = [-1 -2 -1;0 0 0;1 2 1];
    y_mask = [-1 0 1;-2 0 2;-1 0 1];
    image_s = image;
    image = double(image);

    for r = 2:row-1
        for c = 2:col-1
            image_s(r,c) =
            abs(sum(sum(image(r-1:r+1,c-1:c+1).*x_mask)))+
            abs(sum(sum(image(r-1:r+1,c-1:c+1).*y_mask)));
        end
    end
```

```
end
%{
    smooth image using 5*5 mean filter
%}
function image_s = smooth(image)
    [row,col] = size(image);
    image_s = image;
    for r = 3:row-2
        for c = 3:col-2
            image_s(r,c) = mean(mean(image(r-2:r+2,c-2:c+2)));
        end
    end
end
%{
    plot data
%}
function plot_data(a,b,c,d,e,f,g,h)
    figure();

    subplot(241);
    imshow(a);
    title('(a) Oringinal Image');

    subplot(242);
    imshow(b,[]);
    title('(b) Laplacian of (a)');

    subplot(245);
    imshow(c,[]);
    title('(c) Sharpened image');

    subplot(246);
    imshow(d);
    title('(d) Sobel gradient');

    subplot(243);
    imshow(e);
    title('(e) Smoothed sobel image');

    subplot(244);
    imshow(f,[]);
    title('(f) Product of (c) and (e)');

    subplot(247);
    imshow(g,[]);
    title('(g) Sharpened image');

    subplot(248);
    imshow(h,[]);
    title('(h) Final result');
end
```

A.3 Problem 3

```
%{
    Problem 3
    by Fanyong Xue
    Student ID:515030910443
    Filtering in frequency domain

%}

%% Main Part
image = imread('Image Path/characters_test_pattern.tif');

ideal_low_plot_data(image);
ideal_high_plot_data(image);
butterworth_low_plot_data(image);
butterworth_high_plot_data(image);
gaussian_low_plot_data(image);
gaussian_high_plot_data(image);


%% Function Part

% Ideal
function ideal_high_plot_data(image)

    figure('name','Ideal High Pass');

    subplot(321);
    imshow(image);
    title('Original Image');

    subplot(322);
    imshow(IHPF(image,10),[]);
    title('Radius = 10');

    subplot(323);
    imshow(IHPF(image,30),[]);
    title('Radius = 30');

    subplot(324);
    imshow(IHPF(image,60),[]);
    title('Radius = 60');

    subplot(325);
    imshow(IHPF(image,160),[]);
    title('Radius = 160');

    subplot(326);
    imshow(IHPF(image,460),[]);
    title('Radius = 460');
```

```
end
function ideal_low_plot_data(image)

    figure('name','Ideal Low Pass');

    subplot(321);
    imshow(image);
    title('Original Image');

    subplot(322);
    imshow(ILPF(image,10),[]);
    title('Radius = 10');

    subplot(323);
    imshow(ILPF(image,30),[]);
    title('Radius = 30');

    subplot(324);
    imshow(ILPF(image,60),[]);
    title('Radius = 60');

    subplot(325);
    imshow(ILPF(image,160),[]);
    title('Radius = 160');

    subplot(326);
    imshow(ILPF(image,460),[]);
    title('Radius = 460');

end
function image_i = ILPF(image,radius)

    fliter = ILPF_fliter(image,radius);
    image_i = transfer(image,fliter);
    %{
    image_f = fft2(image,2*row,2*col);
    image_f = fftshift(image_f);
    %image_f = log(1+abs(image_f));

    %%%%%%
    image_i = image_f.*fliter;

    image_i = ifftshift(image_i);
    image_i = ifft2(image_i);

    %image_i = abs(image_i);
    image_i = image_i(1:row,1:col);
    image_i = real(image_i);
    image_f = log(1+abs(image_f));
    %}
```

```
end
function image_i = IHPF(image,radius)
    fliter = 1-ILPF_fliter(image,radius);
    image_i = transfer(image,fliter);
end
function fliter = ILPF_fliter(image,radius)
    [row,col]=size(image);
    fliter = zeros(2*row,2*col);

    for r = 1:2*row
        for c = 1:2*col
            if sqrt((r-row)^2+(c-col)^2) <= radius
                fliter(r,c) = 1;
            end
        end
    end
end


% Butterworth
function butterworth_low_plot_data(image)
    figure('name','Butterworth Low Pass');

    subplot(321);
    imshow(image);
    title('Original Image');

    subplot(322);
    imshow(BLFP(image,2,10),[]);
    title('n=2,Radius = 10');

    subplot(323);
    imshow(BLFP(image,2,30),[]);
    title('n=2,Radius = 30');

    subplot(324);
    imshow(BLFP(image,2,60),[]);
    title('n=2,Radius = 60');

    subplot(325);
    imshow(BLFP(image,2,160),[]);
    title('n=2,Radius = 160');

    subplot(326);
    imshow(BLFP(image,2,460),[]);
    title('n=2,Radius = 460');

end
function butterworth_high_plot_data(image)
    figure('name','Butterworth High Pass');

```

```
    subplot(321);
    imshow(image);
    title(' Original Image' );

    subplot(322);
    imshow(BHFP(image,2,10),[]);
    title(' n=2,Radius = 10' );

    subplot(323);
    imshow(BHFP(image,2,30),[]);
    title(' n=2,Radius = 30' );

    subplot(324);
    imshow(BHFP(image,2,60),[]);
    title(' n=2,Radius = 60' );

    subplot(325);
    imshow(BHFP(image,2,160),[]);
    title(' n=2,Radius = 160' );

    subplot(326);
    imshow(BHFP(image,2,460),[]);
    title(' n=2,Radius = 460' );

end
function image_b = BLFP(image, n,radius)

    fliter = BLFP_fliter(image, n,radius);
    image_b = transfer(image,fliter);

end
function image_b = BHFP(image, n,radius)

    fliter = 1-BLFP_fliter(image, n,radius);
    image_b = transfer(image,fliter);

end
function fliter = BLFP_fliter(image, n,radius)
    [row,col]=size(image);
    fliter = zeros(2*row,2*col);

    for r = 1:2*row
        for c = 1:2*col
            fliter(r,c) = 1/(1+(sqrt((r-row)^2+
                         (c-col)^2)/radius)^(2*n));
        end
    end
end

% Gaussian
function gaussian_low_plot_data(image)
```

```
    figure('name','Gaussian Low Pass');

    subplot(321);
    imshow(image);
    title('Original Image');

    subplot(322);
    imshow(GLFP(image,10),[]);
    title('Radius = 10');

    subplot(323);
    imshow(GLFP(image,30),[]);
    title('Radius = 30');

    subplot(324);
    imshow(GLFP(image,60),[]);
    title('Radius = 60');

    subplot(325);
    imshow(GLFP(image,160),[]);
    title('Radius = 160');

    subplot(326);
    imshow(GLFP(image,460),[]);
    title('Radius = 460');
end
function gaussian_high_plot_data(image)
    figure('name','Gaussian High Pass');

    subplot(321);
    imshow(image);
    title('Original Image');

    subplot(322);
    imshow(GHFP(image,10),[]);
    title('Radius = 10');

    subplot(323);
    imshow(GHFP(image,30),[]);
    title('Radius = 30');

    subplot(324);
    imshow(GHFP(image,60),[]);
    title('Radius = 60');

    subplot(325);
    imshow(GHFP(image,160),[]);
    title('Radius = 160');

    subplot(326);
    imshow(GHFP(image,460),[]);
```

```
    title('Radius = 460');
end
function image_g = GLFP(image,radius)
    fliter = GLFP_fliter(image,radius);
    image_g = transfer(image,fliter);
end
function image_g = GHFP(image,radius)
    fliter = 1-GLFP_fliter(image,radius);
    image_g = transfer(image,fliter);
end
function fliter = GLFP_fliter(image,radius)
    [row,col]=size(image);
    fliter = double(zeros(2*row,2*col));

    for r = 1:2*row
        for c = 2:2*col
            fliter(r,c) = exp(-1*(((r-row)^2+
                    (c-col)^2)/(2*radius^2)));
        end
    end
end

function image_t=transfer(image,fliter)
    [row,col]=size(image);
    image_f = fft2(image,2*row,2*col);
    image_f = fftshift(image_f);
    image_t = image_f.*fliter;
    image_t = ifftshift(image_t);
    image_t = ifft2(image_t);
    image_t = image_t(1:row,1:col);
    image_t = abs(image_t);
end
%{

function [ mfft2 ] = JCGuoFFT2( data )
    h = size(data, 1);
    w = size(data, 2);
    mfft2 = data;

    if power(2, log2(h)) ~= h || power(2, log2(w)) ~= w
        disp('JCGuoFFT2 exit: h and w must be the power of 2!')
    else
        for i = 1 : h
            mfft2(i, :) = IterativeFFT(mfft2(i, :));
        end

        for j = 1 : w
            mfft2(:, j) = IterativeFFT(mfft2(:, j));
        end
    end
end
```

```
function image_s = shift_image(image)

    [row,col]=size(image);
    image_s = image;

    for r = 1:row
        for c = 1:col
            image_s(r,c) = image(r,c)*(-1)^(r+c);
        end
    end
end


function image_f = DFT(image,rows,cols)
    [row,col]=size(image);

    %pad image to rows*cols
    image = [image zeros(row,cols-col)];
    image = [image;zeros(rows-row,cols)];
    image = double(image);
    for i = 1:rows
        k = cols/2;
        M = round(log2(cols));
        for j = 1:cols-2
            if j<k
                t = image(i,k);
                image(i,k) = image(i,j);
                image(i,j) = t;
            end
            l = cols/2;
            while l<=k
                k = k-1;
                l = l/2;
            end
            k = k+1;
        end
        for m = 1:M
            la = 2^m;
            lb = la/2;
            for l = 1:lb
                r = (l-1)*2^(M-m);
                n = l-1;
                while n<rows-1
                    lc = n+lb;
                    t = image(lc,j)*exp(2*pi*r/rows);
                    image(i,lc) = image(i,n) - t;
                    image(i,n) = image(i,n) + t;
                    n = n+la;
                end
            end
```

```
            end
        end
        image_f = image;
end
function image_f=DFT(image,rows,cols)

end

function v = DFT_1(V)
        n = length(V);
        fft_m = BitReverseCopy(V);

        for r = 1:log2(n)
            m = power(2,r);
            wm = exp(- 2 * pi * i / m);

            for k = 0 : m : n - 1
                w = 1;
                for j = 0 : m / 2 - 1
                    t = w * fft_m(k + j + m / 2 + 1);
                    u = fft_m(k + j + 1);
                    fft_m(k + j + 1) = u + t;
                    fft_m(k + j + m / 2 + 1) = u - t;
                    w = w * wm;
                end
            end
        end
end
%}
```

A.4 Problem 4

```
%{
        Problem 4
        by Fanyong Xue
    Student ID:515030910443
        Generating different types of noise and comparing different noise reduction
%}

%% Main Part
image = imread('Image Path/Fig0503.tif');
plot_data_noises(image);

circuit = imread('Image Path/Circuit.tif');
plot_data_noises(circuit);
plot_mean_filter(circuit);
plot_order_statistic_filter(circuit);

%% Function Part
% plot data by mean filters
```

```
function plot_mean_filter(image)
    figure('name','Mean Filters1');

    [row,col] = size(image);
    image = im2double(image);
    subplot(221);
    imshow(image);
    title('(a) X-ray image');

    n = uniform_noise(row,col,0,0.3);
    g = gaussian_noise(n,0,0.08);
    image_b = image+g;
    subplot(222);
    imshow(image_b);
    title('(b) Image corrupted by additive Gaussian noise');

    subplot(223);
    imshow(arithmetic_mean_filter(image_b,3,3));
    title('(c) Result of filtering with an arithmetic mean filter');

    subplot(224);
    imshow(geometric_mean_filter(image_b,3,3));
    title('(d) Result of filtering with a geometric mean filter');

    figure('name','Mean Filters2');

    subplot(221);
    image_a_ = impulse_noise(image,0.1,0,-1,0);
    imshow(image_a_);
    title('(a) Image corrupted by pepper noise');

    subplot(222);
    image_b_ = impulse_noise(image,0.1,0,1,0);
    imshow(image_b_);
    title('(b) Image corrupted by salt noise');


    subplot(223);
    imshow(contraharmonic_mean_filter(image_a_,1.5,3,3));
    title('(c) Result of filtering (a) with a contra-harmonic filter');

    subplot(224);
    imshow(contraharmonic_mean_filter(image_b_,-1.5,3,3));
    title('(c) Result of filtering (a) with a contra-harmonic filter');

    figure('name','Mean Filters3');

    subplot(121);
    imshow(contraharmonic_mean_filter(image_a_,-1.5,3,3));
    title('(c) Result of filtering (a) with a
            contra-harmonic filter(Q=-1.5)');
```

```
    subplot(122);
    imshow(contraharmonic_mean_filter(image_b_,1.5,3,3));
    title(' (c) Result of filtering (a) with a
              contra-harmonic filter(Q=1.5)');
end
% plot data by order statistic filters
function plot_order_statistic_filter(image)
    figure('name','Order-Statistic Filters1');
    image = im2double(image);
    image_a = impulse_noise(image,0.1,0.1,-1,1);

    subplot(221);
    imshow(image_a);
    title('Image corrupted by salt-and-pepper noise');

    subplot(222);
    image_b = median_filter(image_a,3,3);
    imshow(image_b);
    title(' (b) Result of one pass with a median filter');

    subplot(223);
    image_c = median_filter(image_b,3,3);
    imshow(image_c);
    title(' (c) Result of processing (b) with the same filter');

    subplot(224);
    imshow(median_filter(image_c,3,3));
    title(' (d) Result of processing (c) with the same filter');

    figure('name','Order-Statistic Filters2');

    image_a__ = impulse_noise(image,0.1,0,-1,0);
    subplot(121);
    imshow(max_filter(image_a__,3,3));
    title(' (a) Result of filtering pepper noise with a max filter');

    image_b__ = impulse_noise(image,0.1,0,1,0);
    subplot(122);
    imshow(min_filter(image_b__,3,3));
    title(' (a) Result of filtering salt noise with a min filter');

    figure('name','Order-Statistic Filters3');
    [row,col] = size(image);
    n = uniform_noise(row,col,0,0.3);
    image_a_ = image + n;
    subplot(321);
    imshow(image_a_);
    title(' (a) Image corrupted by additive uniform noise');

    image_b_ = impulse_noise(image_a_,0.1,0.1,-1,1);
```

```
    subplot(322);
    imshow(image_b_);
    title(' (b) Image corrupted by additive salt-and-pepper noise');

    subplot(323);
    imshow(arithmetic_mean_filter(image_b_,5,5));
    title(' (c) Image (b) filtered with an arithmetic mean filter');

    subplot(324);
    imshow(geometric_mean_filter(image_b_,5,5));
    title(' (c) Image (b) filtered with a geometric mean filter');

    subplot(325);
    imshow(median_filter(image_b_,5,5));
    title(' (c) Image (b) filtered with a median filter');

    subplot(326);
    imshow(alpha_trimmed_mean_filter(image_b_,5,5,5));
    title(' (c) Image (b) filtered with an alpha-trimmed mean filter');
end
% arithmetic mean filter with m*n mask
function image_ = arithmetic_mean_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    %m and n should be odd numbers
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            image_(r,c) = mean(mean(image(r-m_:r+m_,c-n_:c+n_)));
        end
    end
end
% geometric mean filter with m*n mask
function image_ = geometric_mean_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            image_(r,c) = nthroot(prod(prod(
            image(r-m_:r+m_,c-n_:c+n_))),m*n);
        end
    end
end
% harmonic mean filter with m*n mask
function image_ = harmonic_mean_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
```

```
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            image_(r,c) = (m*n)/
            (sum(sum(1./image(
            r-m_:r+m_,c-n_:c+n_))));
        end
    end
end
% contraharmonic mean filter with m*n mask and its oder is q
function image_ = contraharmonic_mean_filter(image,q,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            image_(r,c) = (sum(sum(image(
            r-m_:r+m_,c-n_:c+n_).^(q+1))))/
            (sum(sum(image(
            r-m_:r+m_,c-n_:c+n_).^q)));
        end
    end
    image_ = real(image_);
end
% median filter with m*n mask
function image_ = median_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            image_(r,c) = median(median(image(r-m_:r+m_,c-n_:c+n_)));
        end
    end
end
% max filter with m*n mask
function image_ = max_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            temp = image(r-m_:r+m_,c-n_:c+n_);
            image_(r,c) = max(temp(:));
        end
    end
end
% min filter with m*n mask
```

```
function image_ = min_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            temp = image(r-m_:r+m_,c-n_:c+n_);
            image_(r,c) = min(temp(:));
        end
    end
end
% midpoint filter with m*n mask
function image_ = midpoint_filter(image,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            temp = image(r-m_:r+m_,c-n_:c+n_);
            image_(r,c) = (max(temp(:))+min(temp(:)))/2;
        end
    end
end
% alpha trimmed mean filter with m*n mask(deleta d pixels)
function image_ = alpha_trimmed_mean_filter(image,d,m,n)
    [row,col] = size(image);
    image_ = image;
    m_ = floor(m/2);
    n_ = floor(n/2);
    for r = ceil(m/2):row-m_
        for c = ceil(n/2):col-n_
            temp = image(r-m_:r+m_,c-n_:c+n_);
            temp_ = sort(temp(:));
            image_(r,c) = mean(temp_(floor(d/2):m*n-floor(d/2)));
        end
    end
end

% adding noises to image and plot them
function plot_data_noises(image)

    [row,col] = size(image);
    image = im2double(image);
    figure('name','Uniform Noise');
    n = uniform_noise(row,col,0,0.3);
    image_=image+n;
    subplot(121);
    imshow(image_,[]);
    subplot(122);
```

```
    bar(get_histogram(image_));

    figure('name','Gaussian Noise');
    g = gaussian_noise(n,0,0.08);
    image_ = image+g;
    subplot(121);
    imshow(image_,[]);
    subplot(122);
    bar(get_histogram(image_));

    figure('name','Rayleigh Noise');
    r = rayleigh_noise(n,-0.2,0.03);
    image_ = image+r;
    subplot(121);
    imshow(image_,[]);
    subplot(122);
    bar(get_histogram(image_));


    figure('name','Exponential Noise');
    e = exponential_noise(n,25);
    image_ = image+e;
    subplot(121);
    imshow(image_,[]);
    subplot(122);
    bar(get_histogram(image_));

    figure('name','Gamma Noise');
    ga = gamma_noise(n,25,3);
    image_ = image+ga;
    subplot(121);
    imshow(image_,[]);
    subplot(122);
    bar(get_histogram(image_));

    figure('name','Impulse Noise');
    image_ = impulse_noise(image,0.1,0.1,0.2,-0.2);
    subplot(121);
    imshow(image_,[]);
    subplot(122);
    bar(get_histogram(image_));
end
% row*col uniform noise from low~high
function n = uniform_noise(row,col,low,high)
    n = low + (high-low)*rand([row col]);
end
% normalizing uniform noise to 0~1
function n = normalized(uniform_noise)
    max_ = max(uniform_noise(:));
    min_ = min(uniform_noise(:));
    n = double(uniform_noise-min_);
```

```
    n = n/double(max_-min_);
end
% rayleigh noise
function n = rayleigh_noise(uniform_noise,a,b)
    uniform_noise_ = normalized(uniform_noise);
    n = a + sqrt(-b*log(1-uniform_noise_));
end
% exponential noise
function n = exponential_noise(uniform_noise,a)
    uniform_noise_ = normalized(uniform_noise);
    n = -1/a*log(1-uniform_noise_);
end
% impulse noise
function image_ = impulse_noise(image,pa,pb,a,b)
    [row,col]=size(image);
    uniform_noise_ = rand([row col]);
    image_ = image;
    for r = 1:row
        for c = 1:col
            if uniform_noise_(r,c)<pa
                image_(r,c) = image(r,c)+a;
            elseif uniform_noise_(r,c)>(1-pb)
                image_(r,c) = image(r,c)+b;
            end
        end
    end
end
% gamma noise
function n = gamma_noise(uniform_noise_,a,b)
    uniform_noise_ = normalized(uniform_noise_);
    n = -1/a*log(1-uniform_noise_);
    [row,col] = size(uniform_noise_);
    for i = 2:b
        uniform_noise__=uniform_noise(row,col,0,1);
        n = n-1/a*log(1-uniform_noise__);
    end

end
% gaussian noise
function n = gaussian_noise(uniform_noise,ex,sigma)
    [row,col] = size(uniform_noise);
    n_ = normalized(uniform_noise);
    n = n_;
    for r = 1:row
        if mod(col,2)==0
            for c = 1:2:col
                n(r,c) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,c+1));
                n(r,c+1) = sqrt(-2*log(n_(r,c)))*sin(2*pi*n_(r,c+1));
            end
        else
            for c = 1:2:col-1
```

```
                    n(r,c) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,c+1));
                    n(r,c+1) = sqrt(-2*log(n_(r,c)))*sin(2*pi*n_(r,c+1));
                end
                n(r,col) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,1));
            end
        end
        n = n*double(sigma);
        n = n+ex;
end
% plot image' histogram (but add other 150 to display the negative value)
function histogram = get_histogram(image)
        [row,col]=size(image);
        histogram = zeros(500,1);%-150~350
        for r = 1:row
            for c = 1:col
                gray = int16(image(r,c) /0.004);
                if gray >349
                    gray = 349;
                end
                if gray <-150
                    gray = -150;
                end
                histogram(gray+1+150)=histogram(gray+1+150)+1;
            end
        end
end
```