Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Assignment 1

MEDICAL IMAGE ANALYSIS AND PROCESSING
Dr. E. FATEMIZADEH (2023)

---

1 Theoretical Questions

Question 1:

- Histogram equalization: Let p denote the normalized histogram of f with a bin for each possible intensity.

$$p_n = \frac{\text{number of pixels with intensity } n}{\text{total number of pixels}} \qquad n = 0, 1, ..., L-1$$

So:

$$p_r(r_k) = \frac{n_k}{MN}$$

Then we transform the pixel intensities by the function:

$$s = T(r) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

While the discrete version won't result in exactly flat histograms, it will flatten them and in doing so enhance the contrast in the image.

This method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. Through this adjustment, the intensities can be better distributed on the histogram utilizing the full range of intensities evenly. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the highly populated intensity values which are used to degrade image contrast.

- Adaptive histogram equalization: AHE is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and

uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.

Ordinary histogram equalization uses the same transformation derived from the image histogram to transform all pixels. This works well when the distribution of pixel values is similar throughout the image. However, when the image contains regions that are significantly lighter or darker than most of the image, the contrast in those regions will not be sufficiently enhanced.

Adaptive histogram equalization (AHE) improves on this by transforming each pixel with a transformation function derived from a neighbourhood region. It was first developed for use in aircraft cockpit displays. In its simplest form, each pixel is transformed based on the histogram of a square surrounding the pixel. The derivation of the transformation functions from the histograms is exactly the same as for ordinary histogram equalization: The transformation function is proportional to the cumulative distribution function (CDF) of pixel values in the neighbourhood.

Advantages:

It computes the HE of distinct sections of the image.

It preserves the edges in distinct regions of the image.

It enhances the contrast locally.

Disadvantage:

AHE overamplifies the noise in relatively homogeneous regions of an image. To prevent this a variant of adaptive histogram equalization called contrast limited adaptive histogram equalization (CLAHE) is used.

- Contrast limited adaptive histogram equalization: Ordinary AHE tends to overamplify the contrast in near-constant regions of the image, since the histogram in such regions is highly concentrated. As a result, AHE may cause noise to be amplified in near-constant regions. Contrast Limited AHE (CLAHE) is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification.

  In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighbourhood cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighbourhood region. Common values limit the resulting amplification to between 3 and 4.

  It is advantageous not to discard the part of the histogram that exceeds the clip limit but to redistribute it equally among all histogram bins.

  The redistribution will push some bins over the clip limit again (region shaded green in the figure), resulting in an effective clip limit that is larger than the prescribed limit and the exact value of which depends on the image. If this is undesirable, the redistribution procedure can be repeated recursively until the excess is negligible.

According to the descriptions above Image I is filtered with CLAHE and Image II is filtered with AHE.

2 Practical Questions

Question 2.1:

The backward difference is a finite difference defined by

$$\nabla_p \equiv \nabla f_p \equiv f_p - f_{p-1}$$

Higher order differences are obtained by repeated operations of the backward difference operator, so

$$\begin{aligned}\nabla_p^2 &= \nabla(\nabla p) = \nabla\left(f_p - f_{p-1}\right) = \nabla f_p - \nabla f_{p-1} \\ &= \left(f_p - f_{p-1}\right) - \left(f_{p-1} - f_{p-2}\right) \\ &= f_p - 2 f_{p-1} + f_{p-2}.\end{aligned}$$

In general,

$$\nabla_p^k \equiv \nabla^k f_p \equiv \sum_{m=0}^{k} (-1)^m \binom{k}{m} f_{p-m}$$

The forward difference is a finite difference defined by

$$\Delta a_n \equiv a_{n+1} - a_n$$

Higher order differences are obtained by repeated operations of the forward difference operator,

$$\Delta^k a_n = \Delta^{k-1} a_{n+1} - \Delta^{k-1} a_n$$

So

$$\begin{aligned}\Delta^2 a_n &= \Delta_n{}^2 \\ &= \Delta(\Delta_n) \\ &= \Delta(a_{n+1} - a_n) \\ &= \Delta_{n+1} - \Delta_n \\ &= a_{n+2} - 2 a_{n+1} + a_n\end{aligned}$$

In general,

$$\Delta_n^k \equiv \Delta^k\, a_n \equiv \sum_{i=0}^{k} (-1)^i \binom{k}{i} a_{n+k-i}$$

The central difference for a function tabulated at equal intervals f_n is defined by

$$\delta\left(f_n\right) = \delta_n = \delta_n^1 = f_{n+1/2} - f_{n-1/2}$$

First and higher order central differences arranged so as to involve integer indices are then given by

$$\begin{aligned}
\delta_{n+1/2} &= \delta_{n+1/2}^1 \\
&= f_{n+1} - f_n \\
\delta_n^2 &= \delta_{n+1/2}^1 - \delta_{n-1/2}^1 \\
&= f_{n+1} - 2\, f_n + f_{n-1} \\
\delta_{n+1/2}^3 &= \delta_{n+1}^2 - \delta_n^2 \\
&= f_{n+2} - 3\, f_{n+1} + 3\, f_n - f_{n-1}
\end{aligned}$$

Higher order differences may be computed for even and odd powers,

$$\delta_n^{2\,k} = \sum_{j=0}^{2\,k} (-1)^j \binom{2\,k}{j} f_{n+k-j}$$

$$\delta_{n+1/2}^{2\,k+1} = \sum_{j=0}^{2\,k+1} (-1)^j \binom{2\,k+1}{j} f_{n+k+1-j}$$

We use the following MATLAB code:

```
%%
clc
clear all
close all
X = imread('q1.png');% load image
img = double(X);% cast to double
figure;
imshow(img)%show image
plotImages(img , "c")%plot central difference
plotImages(img , "b")%plot backward difference
plotImages(img , "f")%plot forward difference
%% Functions
function plotImages(imgMatrix , string)
sz = size(imgMatrix);% get size of image
derivy = zeros(sz(1), sz(2));% initialize fx
derivx = zeros(sz(1), sz(2));% initialize fy
if strcmp( string , 'b' ) %backward
    for i = 2 : sz(1)
        derivy(i, :) = imgMatrix(i, :) - imgMatrix(i-1, :);%y direction diff
    end
```

```
    for i = 2 : sz(2)
        derivx(:, i) = imgMatrix(:, i) - imgMatrix(:, i-1);%x direction diff
    end
elseif strcmp( string , 'f' ) %forward
    for i = 1 : sz(1) - 1
        derivy(i, :) = imgMatrix(i+1, :) - imgMatrix(i, :);%y direction diff
    end
    for i = 1 : sz(2)-1
        derivx(:, i) = imgMatrix(:, i+1) - imgMatrix(:, i);%x direction diff
    end
elseif strcmp( string , 'c' )
    for i = 2 : sz(1) - 1
        derivy(i, :) = imgMatrix(i+1, :) - imgMatrix(i-1, :);%y direction diff
    end
    for i = 2 : sz(2)-1
        derivx(:, i) = imgMatrix(:, i+1) - imgMatrix(:, i-1);%x direction diff
    end
end
derivxx = sqrt(derivy.*derivy + derivx.*derivx);%xy direction
figure
imshow(cast(abs(derivy), 'uint8'))%show fy
figure
imshow(cast(abs(derivx), 'uint8'))%show fx
figure
imshow(cast(abs(derivxx), 'uint8'))%%show fyx
end
```

The results are as below:

- backward:
  - X direction:

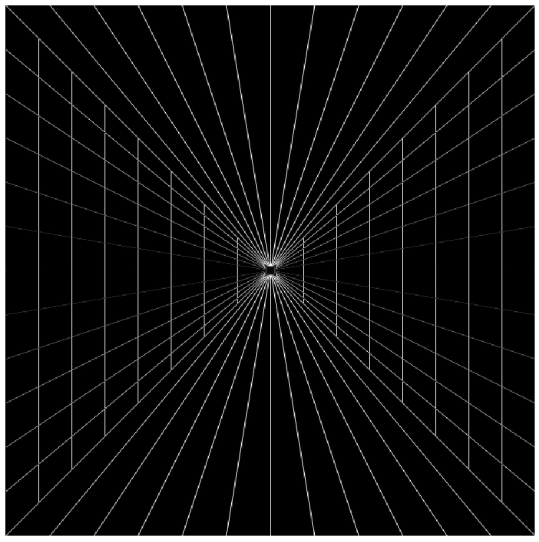


Figure 1: X direction Backward

  - Y direction:

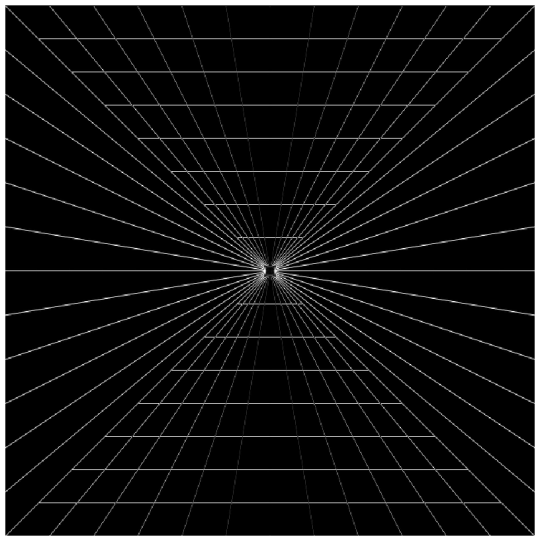


Figure 2: Y direction Backward

- Both direction:

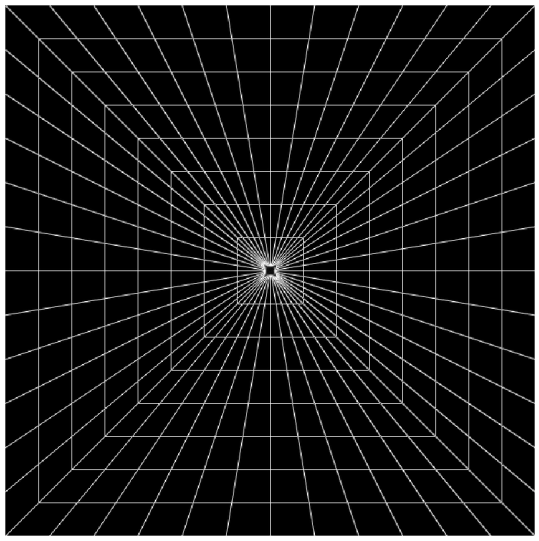


Figure 3: Both direction Backward

- forward:
  - X direction:

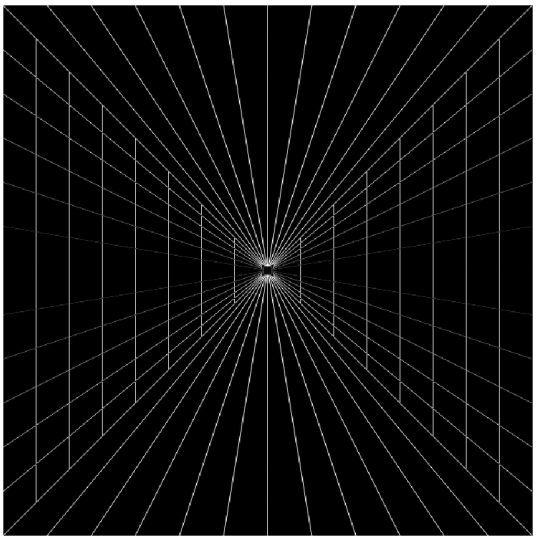


Figure 4: X direction Backward

– Y direction:

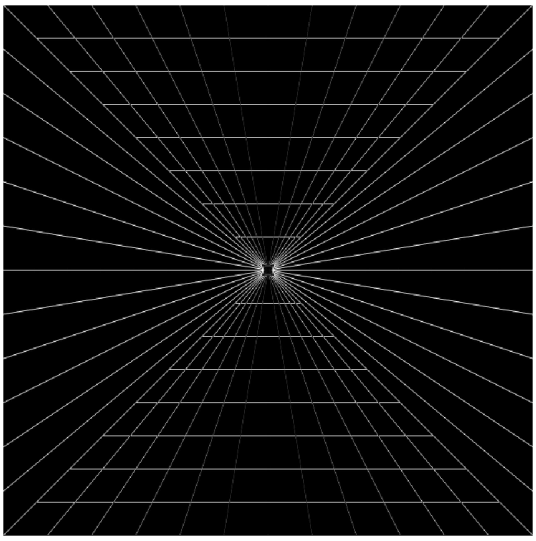


Figure 5: Y direction Backward

– Both direction:

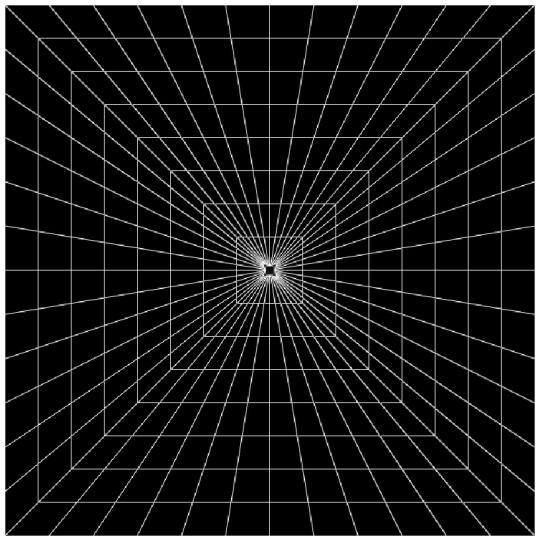


Figure 6: Both direction Backward

- central:
  - X direction:

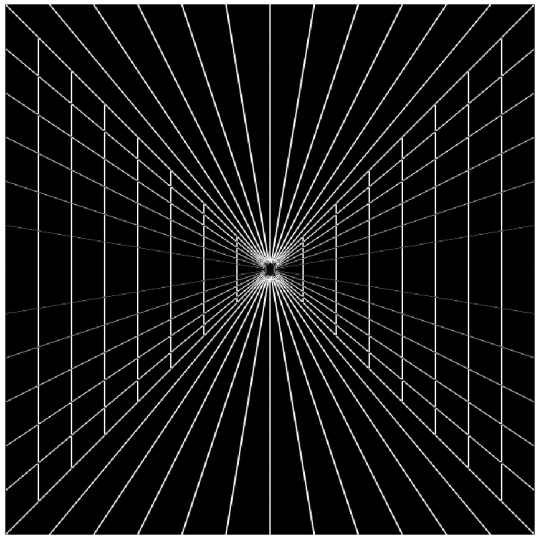


Figure 7: X direction Backward

  - Y direction:

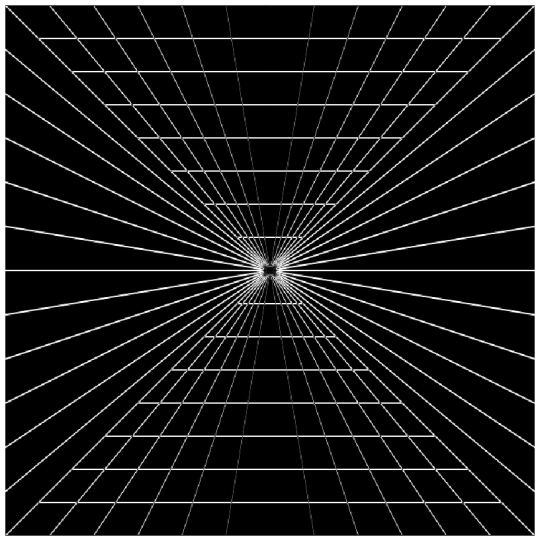


Figure 8: Y direction Backward

– Both direction:

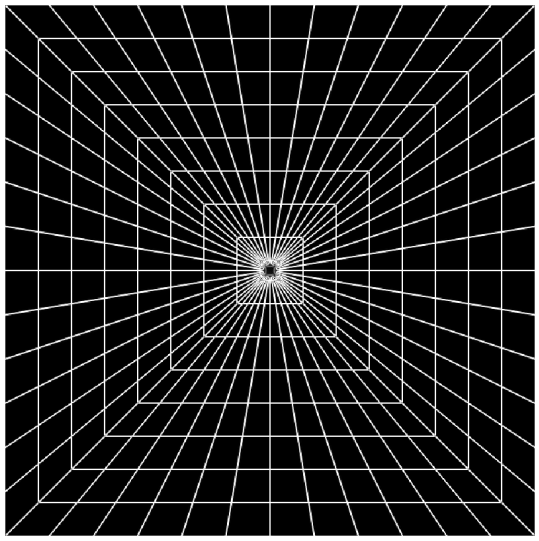


Figure 9: Both direction Backward

Question 2.2:

First we calculate SNR in each part of the image:

	1	2	3	4
1	5.1457	2.2204	2.2514	1.1817e+03
2	top-left	bottom-left	bottom-right	top-right

Figure 10: SNR for each part of image

The above SNRs have been saved in SNR1 matrix.

Now we filter noisy image in 7 ways:

1. 3 by 3 median filtering
2. 5 by 5 median filtering
3. 3 by 3 mean filtering
4. 5 by 5 mean filtering
5. 9 by 9 mean filtering
6. gaussian filter sigma = 0.5
7. gaussian filter sigma = 1

SNRs are as below:

	1	2	3	4	5	6	7
1	4.0084	4.0084	3.3397	3.3678	3.3325	4.3396	3.4422

Figure 11: SNR for 7 filtering methods

As we see in the table, Gaussian filter's performance and Median filters' performance are better than others. Now by the following figures we can assign a number between 1 and 20 to compare the performances visually.

	1	2	3	4	5	6	7
1	4.0084	4.0084	3.3397	3.3678	3.3325	4.3396	3.4422
2	16	16	13	12	10	15	14

Figure 12: SNRs and 1-20 numbers to visually compare

We can see that median filter is the best one for pepper and salt noise. Also Gaussian filter is much better than other filters for Gaussian noise. And we can see that mean filter is not an appropriate filter to delete these kinds of noise.

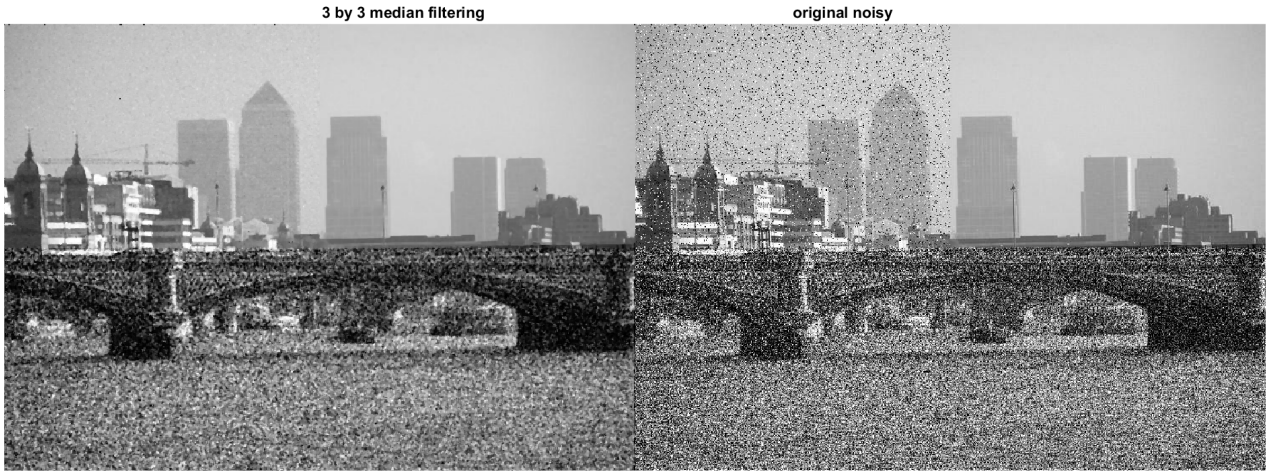


Figure 13:

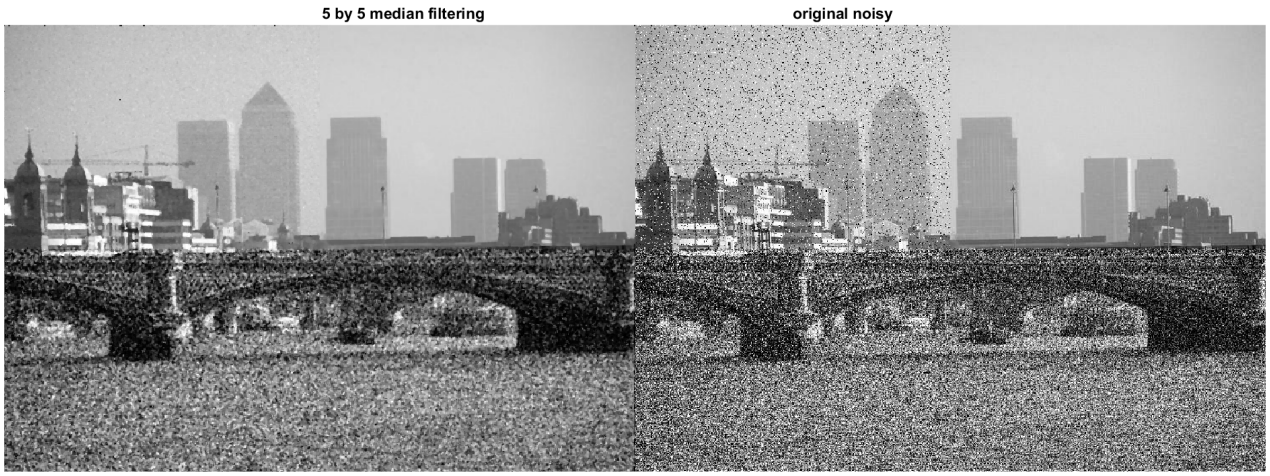


Figure 14:

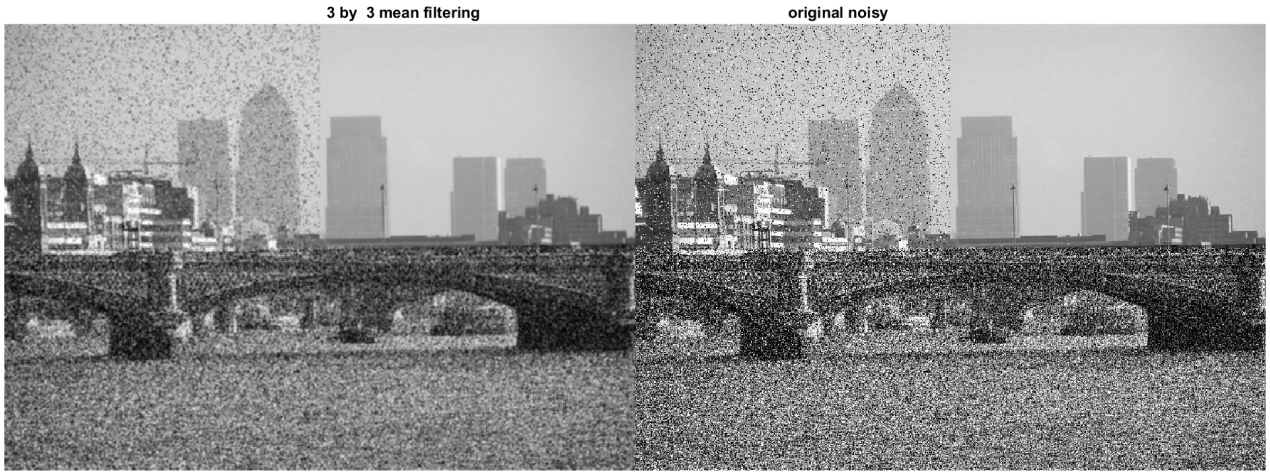


Figure 15:

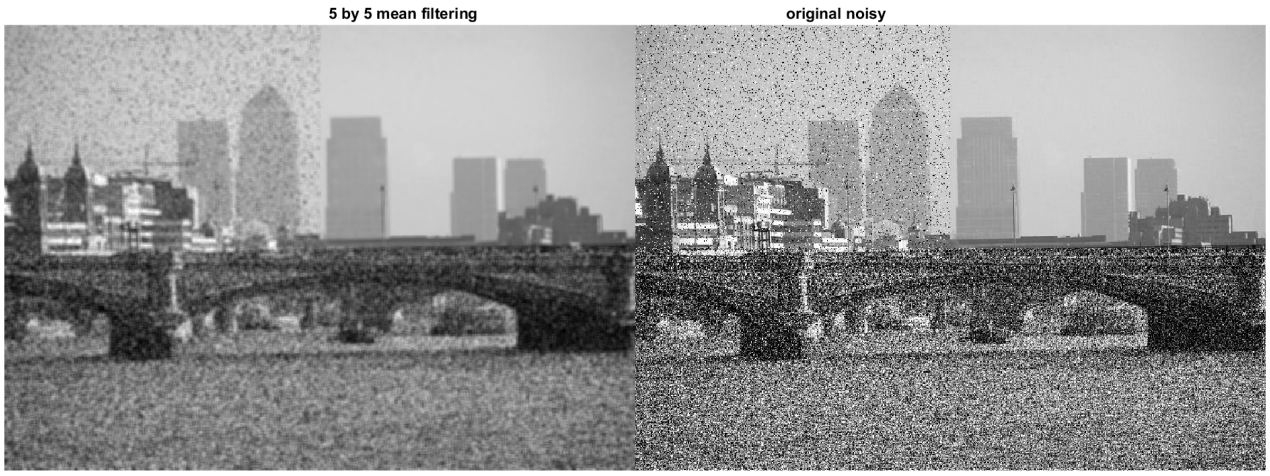


Figure 16:

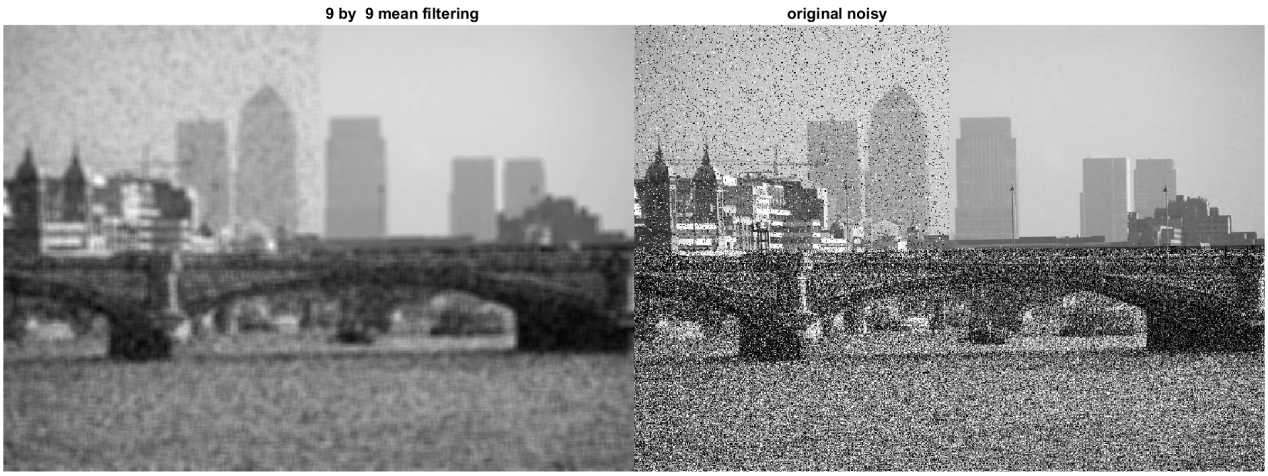


Figure 17:

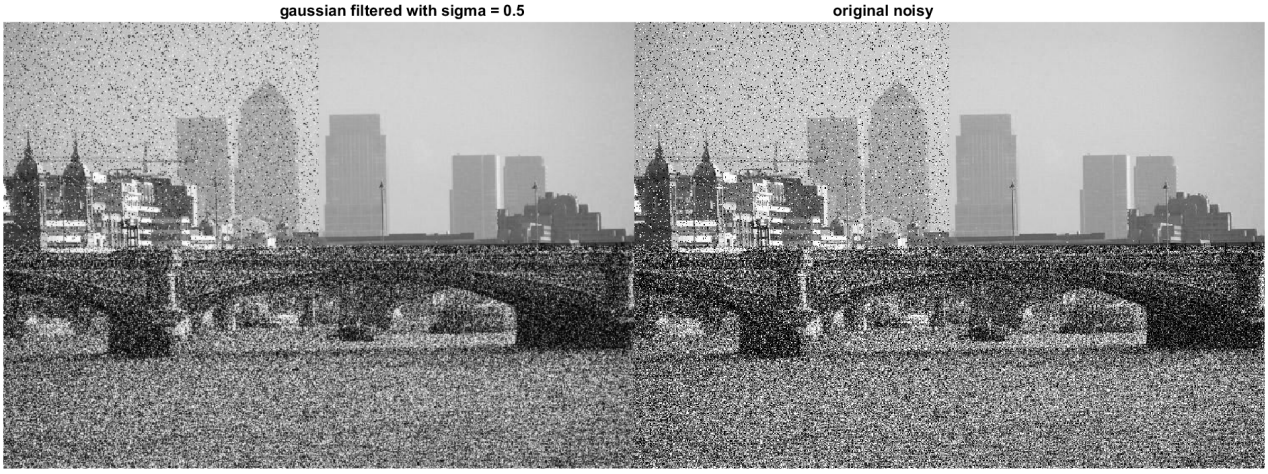


Figure 18:

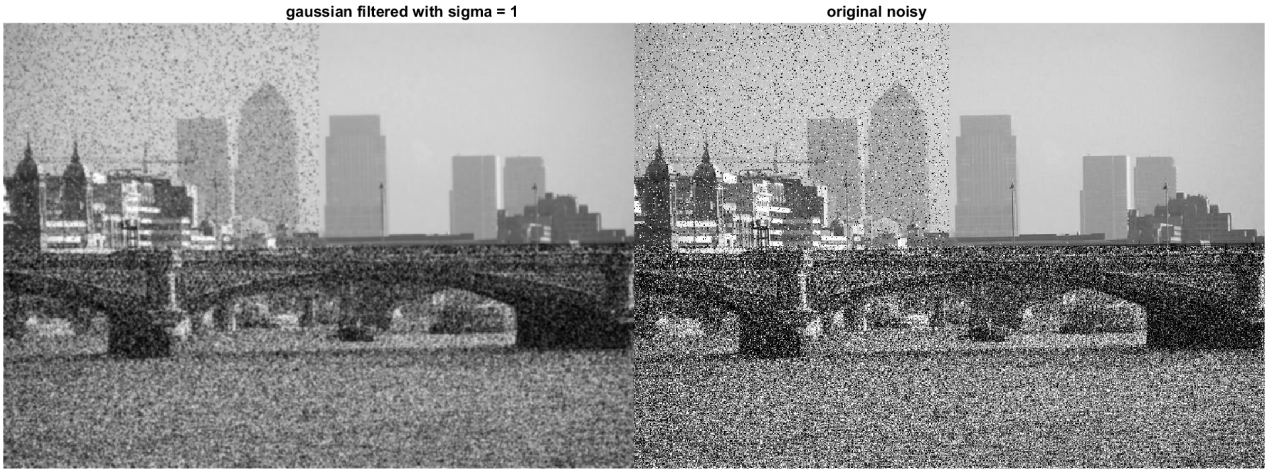


Figure 19:

We used the following MATLAB code for this part:

```
clc
clear all
close all
NoiseImgMatrix = imread('city_noise.jpg');
ImgMatrix = imread('city_orig.jpg');
sz = size(NoiseImgMatrix);
%% SNR only salt & pepper
noise1 = NoiseImgMatrix(1:sz(1)/2 , 1:sz(2)/2);
org1 = ImgMatrix(1:sz(1)/2 , 1:sz(2)/2);
SNR1(1) = sum(sum(org1.*org1))/sum(sum((org1 - noise1).*(org1 - noise1)));
%% SNR salt & pepper $ gaussian
noise2 = NoiseImgMatrix(sz(1)/2+1 : end , 1:sz(2)/2);
org2 = ImgMatrix(sz(1)/2+1 : end , 1:sz(2)/2);
SNR1(2) = sum(sum(org2.*org2))/sum(sum((org2 - noise2).*(org2 - noise2)));
%% SNR only gaussian
noise3 = NoiseImgMatrix(sz(1)/2+1 : end , sz(2)/2+1 : end);
org3 = ImgMatrix(sz(1)/2+1 : end , sz(2)/2+1 : end);
SNR1(3) = sum(sum(org3.*org3))/sum(sum((org3 - noise3).*(org3 - noise3)));
%% SNR No noise
noise4 = NoiseImgMatrix(1:sz(1)/2 , sz(2)/2+1 : end);
org4 = ImgMatrix(1:sz(1)/2 , sz(2)/2+1 : end);
SNR1(4) = sum(sum(org4.*org4))/sum(sum((org4 - noise4).*(org4 - noise4)));
%% 2-D 3 by 3 median filtering
filtered = medfilt2(NoiseImgMatrix);
figure;
imshowpair(filtered , NoiseImgMatrix,'montage')
title('3 by 3 median filtering
              original noisy             ');
SNR2(1) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
%% 2-D 5 by 5 median filtering
filtered = medfilt2(NoiseImgMatrix);
figure;
imshowpair(filtered , NoiseImgMatrix,'montage')
title('5 by 5 median filtering
              original noisy             ');
SNR2(2) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
%% 3 by 3 mean filtering
filtered = imfilter(NoiseImgMatrix, ones(3)/9, 'symmetric');
figure;
imshowpair(filtered , NoiseImgMatrix,'montage')
title('3 by  3 mean filtering
             original noisy             ');
SNR2(3) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
%% 5 by 5 mean filtering
filtered = imfilter(NoiseImgMatrix, ones(5)/25, 'symmetric');
figure;
imshowpair(filtered , NoiseImgMatrix,'montage')
title('5 by 5 mean filtering
            original noisy             ');
SNR2(4) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
%% 9 by  9 mean filtering
filtered = imfilter(NoiseImgMatrix, ones(9)/81, 'symmetric');
figure;
imshowpair(filtered , NoiseImgMatrix,'montage')
title('9 by  9 mean filtering
              original noisy             ');
SNR2(5) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
```

```
    NoiseImgMatrix)));
%% gaussian filter sigma = 0.5
filtered = imgaussfilt(NoiseImgMatrix,0.5);
figure;
imshowpair(filtered, NoiseImgMatrix,'montage')
title('gaussian filtered with sigma = 0.5
                        original noisy                ');
SNR2(6) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
%% gaussian filter sigma = 1
filtered = imgaussfilt(NoiseImgMatrix,1);
figure;
imshowpair(filtered, NoiseImgMatrix,'montage')
title('gaussian filtered with sigma = 1
                      original noisy                ');
SNR2(7) = sum(sum(filtered.*filtered))/sum(sum((filtered - NoiseImgMatrix).*(filtered -
    NoiseImgMatrix)));
```

Question 2.3:

First part:

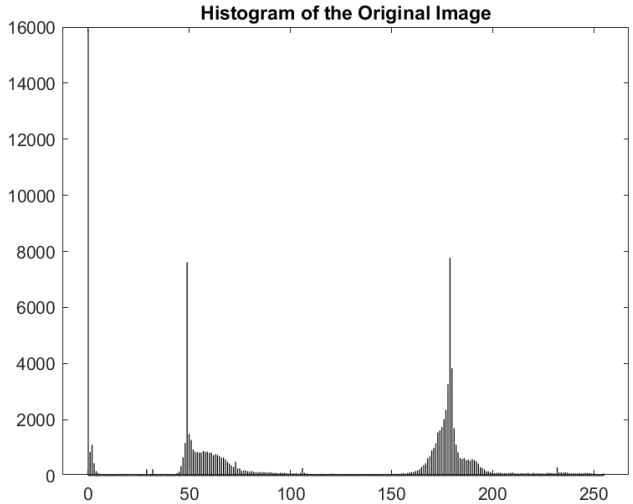
Normalize an histogram is a technique consisting into transforming the discrete distribution of intensities into a discrete distribution of probabilities. To do so, we need to divide each value of the histogram by the number of pixel. Because a digital image is a discrete set of values that could be seen as a matrix and it's equivalent to divide each nk by the dimension of the array which is the product of the width by the length of the image.

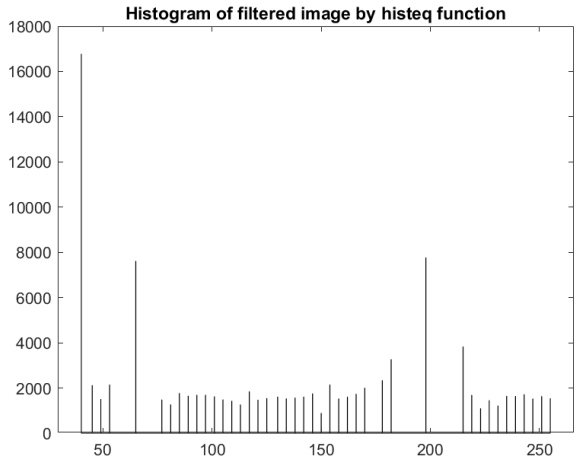
$$n_{kn} = \frac{n_k}{length \times width} = p_r(r_k)$$

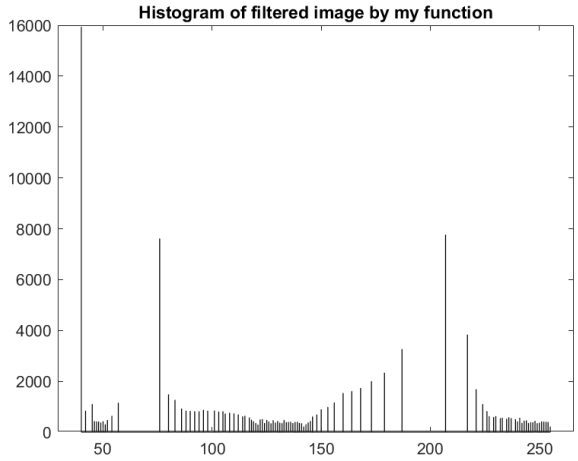
Which could be written in terms of mathematical transformation:

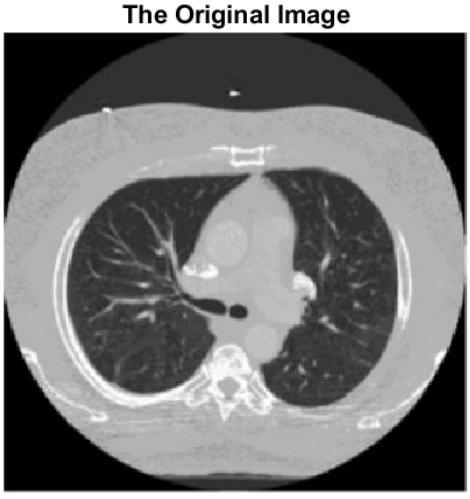
$$\begin{cases} [0, L-1] \to \mathbb{N} \\ x \to Card(x) \end{cases} \quad \text{becomes} \quad \begin{cases} [0, L-1] \to [0, 1] \\ x \to pdf(x) = \frac{Card(x)}{\sum_{i=0}^{L-1} Card(x_i)} \end{cases}$$

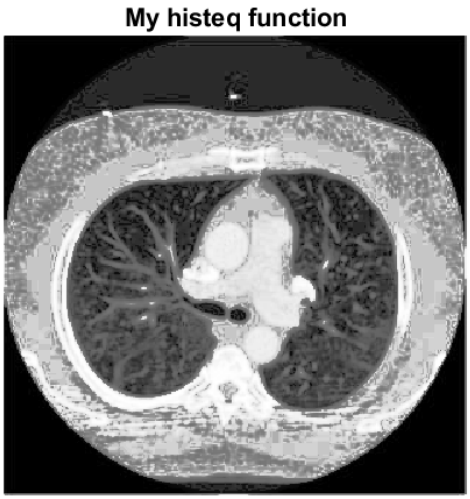
Where Card means the cardinality of the set so in our case the number of pixel.




Histogram of filtered image by histeq function
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250


Histogram of filtered image by my function
16000
14000
12000
10000
8000
6000
4000
2000
0
50
100
150
200
250


The Original Image

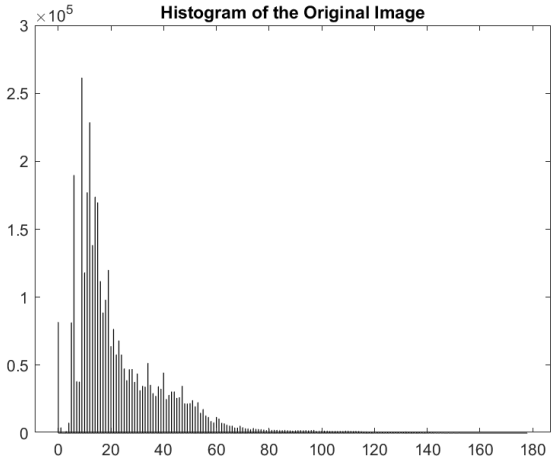


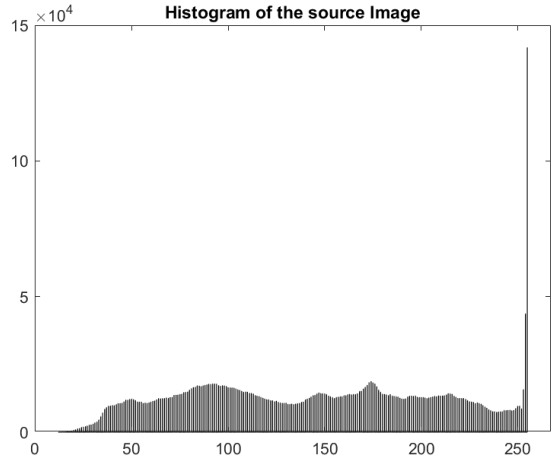
Histogram equalization enhances the contrast of an image by redistributing its intensity values through its histogram. The technique works by spreading out the pixels' intensity values in the image's dynamic range, which in turn stretches the image's contrast over a wider range.

This process involves mapping the original intensity values of the image to a new set of values that are more evenly distributed across the available intensity range. Areas of the image with low contrast, which previously may have appeared washed out or overly dark, will also now have an increased contrast.

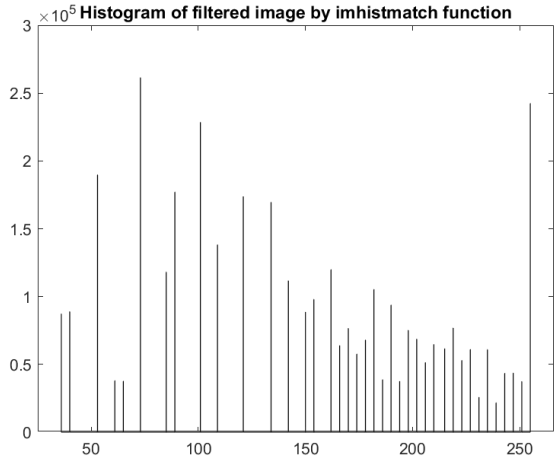
By redistributing the original image's intensity values, histogram equalization can increase the overall image contrast, making it appear sharper and more defined. But it's important to note that histogram equalization amplifies noise and can cause color distortion in some cases. Therefore, it's often necessary to carry out additional image processing to remove any unwanted effects of histogram equalization.

Second part:


Histogram of the Original Image
×10^5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
140
160
180


Histogram of the source Image
×10^4
15
10
5
0
0
50
100
150
200
250


Histogram of filtered image by my function
×10^5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250


Histogram of filtered image by imhistmatch function
×10^5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250


The Original Image


The Original source Image

My codes:

```
%% Implementation part a
clear all;  clc; close all;
M = imread('q3_1.jpg');%load image
I = M(:,:,1);
[row, column] = size(I);% get size of image
freq = zeros(256, 1);%nk matrix
for i = 1:row
    for j = 1:column
        freq(I(i, j) + 1) = freq(I(i, j) + 1) + 1;% calculate intensity
    end
end
pdf = freq / (row * column);%generate pdf of intensity
cdf  = cumsum(pdf);%generate cdf of intensity
outMapper = round((255) * cdf);
output = outMapper(I + 1);% Generate the output image
figure
subplot(3,2,1), imshow(I), title('The Original Image');
subplot(3,2,2), histogram(I, 2550), title('Histogram of the Original Image');
subplot(3,2,3), imshow(output, []), title('My histeq function');
subplot(3,2,4), histogram(uint8(output), 2550), title('Histogram of filtered image by my
    function');
subplot(3,2,5), imshow(histeq(I), []), title('The built-in histeq');
subplot(3,2,6), histogram(histeq(I), 2550), title('Histogram of filtered image by histeq
    function');
figure, imshow(I), title('The Original Image');
```

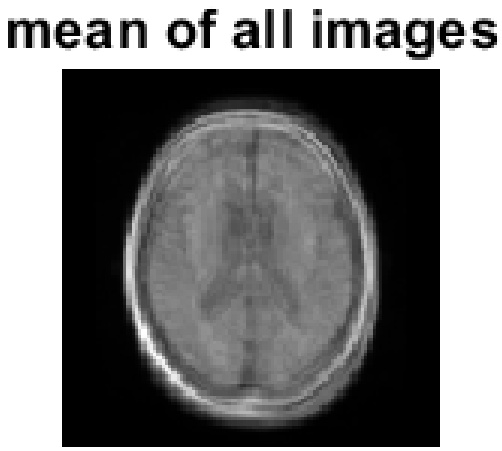
```
saveas(gcf,'The Original Image','png')
figure, histogram(I, 2550), title('Histogram of the Original Image');
saveas(gcf,'Histogram of the Original Image','png')
figure, imshow(output, []), title('My histeq function');
saveas(gcf,'My histeq function','png')
figure, histogram(uint8(output), 2550), title('Histogram of filtered image by my function');
saveas(gcf,'Histogram of filtered image by my function','png')
figure, imshow(histeq(I), []), title('The built-in histeq');
saveas(gcf,'The built-in histeq','png')
figure, histogram(histeq(I), 2550), title('Histogram of filtered image by histeq function');
saveas(gcf,'Histogram of filtered image by histeq function','png')
%% Implementation part b
clc; clear all; close all;
M = imread('q3_2.jpg');%load image
I = M(:,:,1);
M1 = imread('q3_3.jpg');
I1 = M1(:,:,1);
[row, column] = size(I);% get size of image
freq = zeros(256, 1);%nk matrix
for i = 1:row
    for j = 1:column
        freq(I(i, j) + 1) = freq(I(i, j) + 1) + 1;% calculate intensity
    end
end
pdf1 = freq / (row * column);%generate pdf of intensity
cdf1  = cumsum(pdf1);%generate cdf of intensity

[row, column] = size(I1);% get size of image
freq = zeros(256, 1);%nk matrix
for i = 1:row
    for j = 1:column
        freq(I1(i, j) + 1) = freq(I1(i, j) + 1) + 1;% calculate intensity
    end
end
pdf2 = freq / (row * column);%generate pdf of intensity
cdf2  = cumsum(pdf2);%generate cdf of intensity


for i = 1 : 256
    [~, mm] = min(abs(cdf1(i) - cdf2));
    Mapper(i) = mm-1;
end

output = Mapper(double(I) + 1);% Generate the output image


figure
subplot(4,2,1), imshow(I), title('The Original Image');
subplot(4,2,2), histogram(I, 2550), title('Histogram of the Original Image');
subplot(4,2,3), imshow(output,[]), title('My histmatch function');
subplot(4,2,4), histogram(uint8(output), 2550), title('Histogram of filtered image by my
    function');
subplot(4,2,5), imshow(imhistmatch(I,I1), []), title('The built-in imhistmatch');
subplot(4,2,6), histogram(imhistmatch(I,I1), 2550), title('Histogram of filtered image by
    imhistmatch function');
subplot(4,2,7), imshow(I1), title('The Original Image');
subplot(4,2,8), histogram(I1, 2550), title('Histogram of the source Image');
figure

figure
imshow(I), title('The Original Image');
```

```
saveas(gcf,'The Original Image','png')
figure
histogram(I, 2550), title('Histogram of the Original Image');
saveas(gcf,'Histogram of the Original Image','png')
figure
imshow(output,[]), title('My histmatch function');
saveas(gcf,'My histmatch function','png')
figure
histogram(uint8(output), 2550), title('Histogram of filtered image by my function');
saveas(gcf,'Histogram of filtered image by my function','png')
figure
imshow(imhistmatch(I,I1), []), title('The built-in imhistmatch');
saveas(gcf,'The built-in imhistmatch','png')
figure
histogram(imhistmatch(I,I1), 2550), title('Histogram of filtered image by imhistmatch
    function');
saveas(gcf,'Histogram of filtered image by imhistmatch function','png')
figure
imshow(I1), title('The Original source Image');
saveas(gcf,'The Original source Image','png')
figure
histogram(I1, 2550), title('Histogram of the source Image');
saveas(gcf,'Histogram of the source Image','png')
```

Question 2.4:

First we resize the images to 100*100 pixels.

- Mean of all images:



- Top five eigenvalues' corresponding eigenvectors are shown in the figures below:

Image for 1st eigen vector

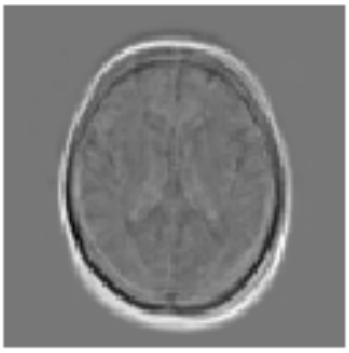


Image for 2nd eigen vector

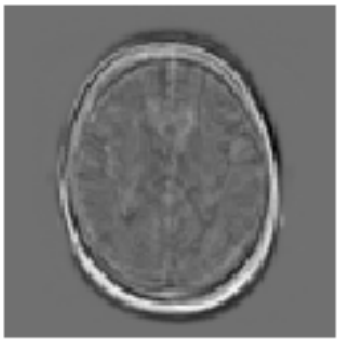


Image for 3rd eigen vector

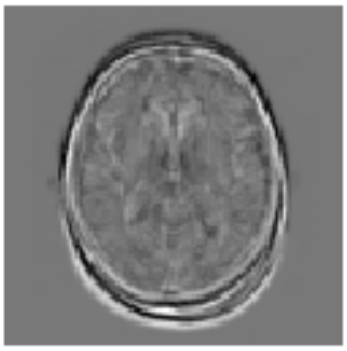
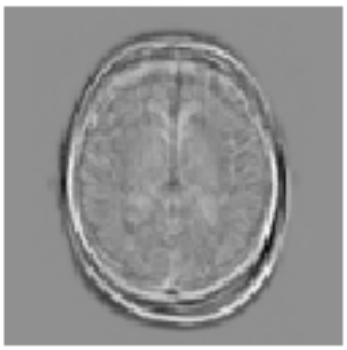
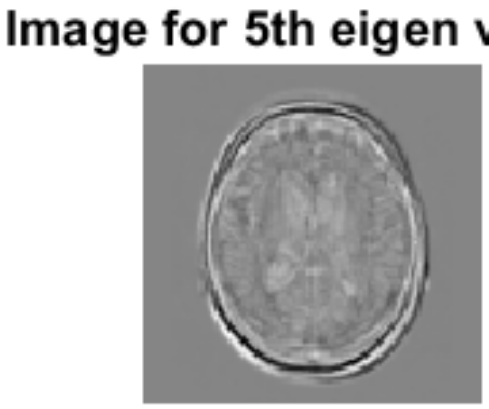


Image for 4th eigen vector



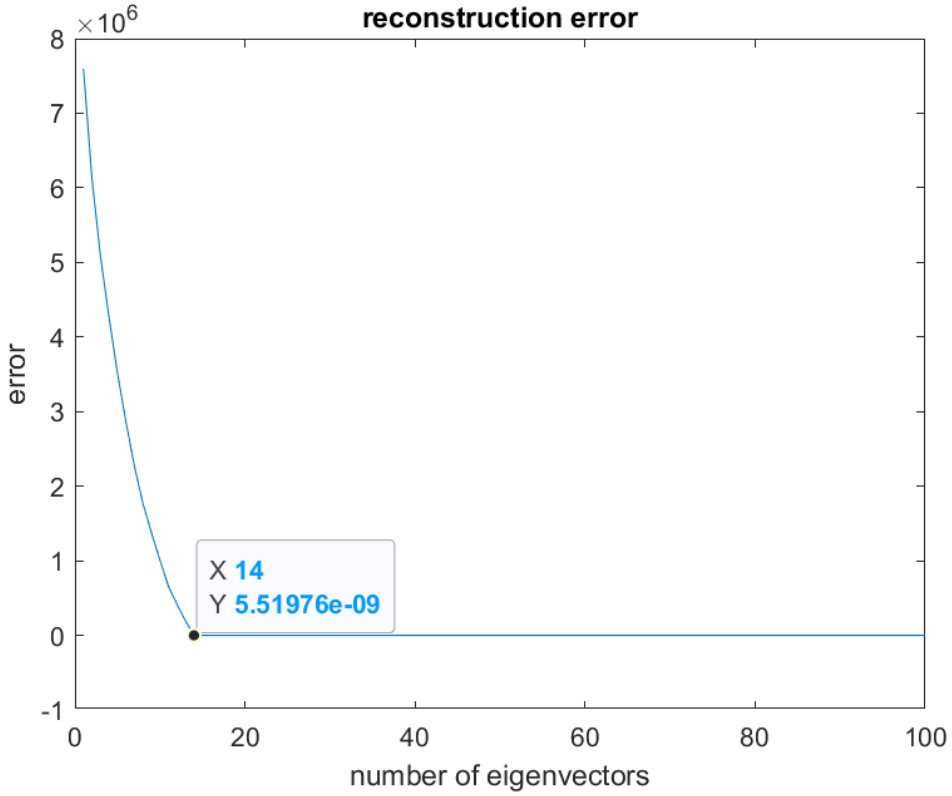


- We can calculate the reconstruction error from the following formula:

$$E\{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}_{\boldsymbol{m}}\|^2\} = \sum_{i=m+1}^{n} \lambda_i$$

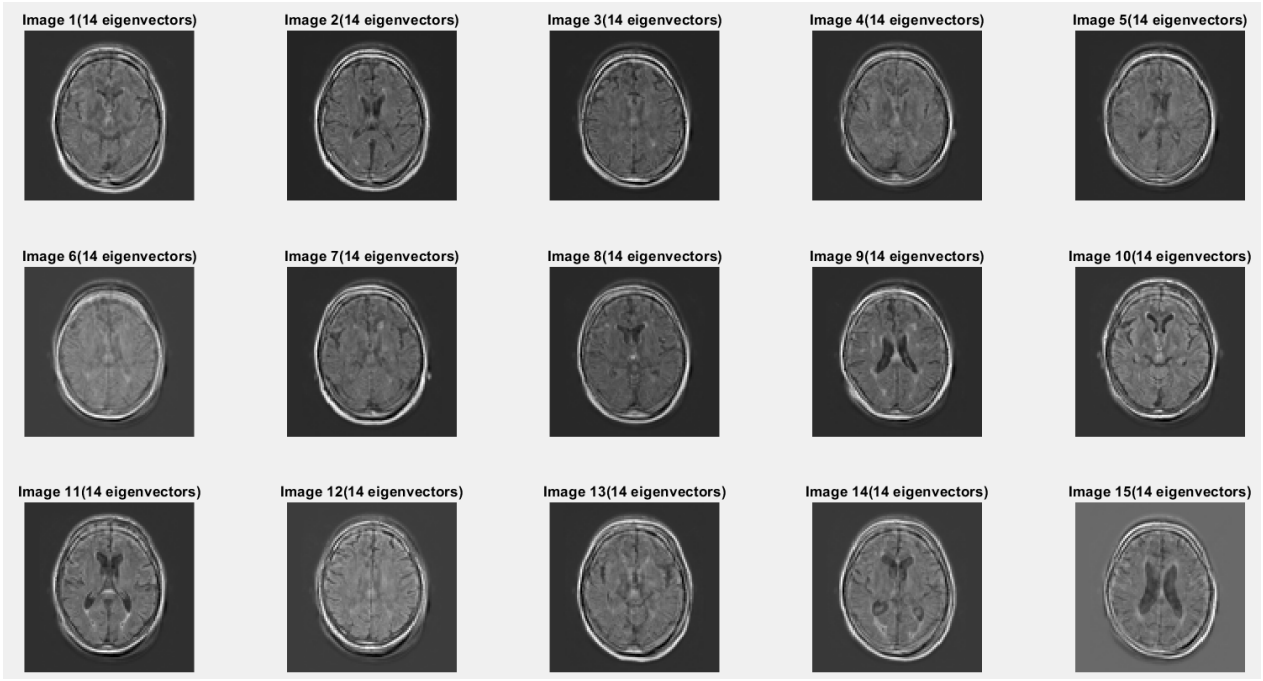
Which m is number of best eigenvectors we have used in reconstruction.

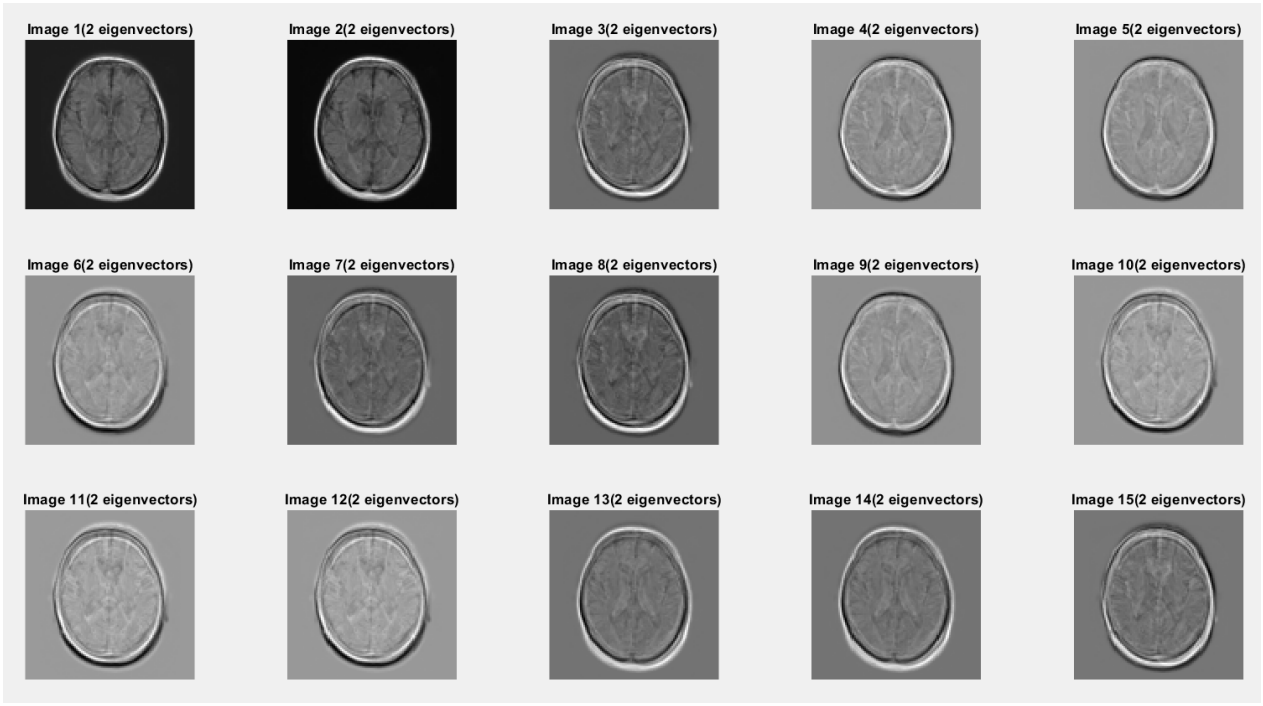
Now we plot error per m:



As you see in the figure above, the best number of eigenvectors to reconstruct the image is 14.

So we reconstruct the original image with 14 eigenvectors:





The code below has been used to do this part:

```
%% load and resize images
clc;clear all;close all;
L = 100;
I1 = imread('brain_data/1.jpg');
I(1:L, 1:L, 1) = imresize(I1,[100 100]);
I1 = imread('brain_data/2.jpg');
I(1:L, 1:L, 2) = imresize(I1,[100 100]);
I1 = imread('brain_data/3.jpg');
I(1:L, 1:L, 3) = imresize(I1,[100 100]);
I1 = imread('brain_data/4.jpg');
```

```
I(1:L, 1:L, 4) = imresize(I1,[100 100]);
I1 = imread('brain_data/5.jpg');
I(1:L, 1:L, 5) = imresize(I1,[100 100]);
I1 = imread('brain_data/6.jpg');
I(1:L, 1:L, 6) = imresize(I1,[100 100]);
I1 = imread('brain_data/7.jpg');
I(1:L, 1:L, 7) = imresize(I1,[100 100]);
I1 = imread('brain_data/8.jpg');
I(1:L, 1:L, 8) = imresize(I1,[100 100]);
I1 = imread('brain_data/9.jpg');
I(1:L, 1:L, 9) = imresize(I1,[100 100]);
I1 = imread('brain_data/10.jpg');
I(1:L, 1:L, 10) = imresize(I1,[100 100]);
I1 = imread('brain_data/11.jpg');
I(1:L, 1:L, 11) = imresize(I1,[100 100]);
I1 = imread('brain_data/12.jpg');
I(1:L, 1:L, 12) = imresize(I1,[100 100]);
I1 = imread('brain_data/13.jpg');
I(1:L, 1:L, 13) = imresize(I1,[100 100]);
I1 = imread('brain_data/14.jpg');
I(1:L, 1:L, 14) = imresize(I1,[100 100]);
I1 = imread('brain_data/15.jpg');
I(1:L, 1:L, 15) = imresize(I1,[100 100]);
%% mean of images
f = figure;
imshow(mean(double(I),3), [])
title('mean of all images')
%% feature matrix
ff = [];
for i = 1 : 15
    temp = I(:,:, i);
   ff = [ff reshape(temp, [L*L 1])];
end
features = double(ff');
 %% mean vector
meanVector = reshape(mean(I,3), [1 L*L]);
%% covariance matrix
M = zeros(L*L, L*L);
for i = 1 :15
    M = M + ((features(i, :)- meanVector)')*(features(i, :)-meanVector);
end
M = M/14;
%% eigen values and eigen vectors of covariance matrix
D = eig(M);
[V,~] = eig(M);
%% sort eigen values
[B,S] = sort(D, 'descend');
%% bigger eigen values
figure
temp1 = V(:, S(1));
imshow(reshape(temp1, [L L]) ,[])
title("Image for 1st eigen vector")
figure
temp2 = V(:, S(2));
imshow(reshape(temp2, [L L]) ,[])
title("Image for 2nd eigen vector")
figure
temp3 = V(:, S(3));
imshow(reshape(temp3, [L L]) ,[])
title("Image for 3rd eigen vector")
figure
```

```
temp4 = V(:, S(4));
imshow(reshape(temp4, [L L]) ,[])
title("Image for 4th eigen vector")
figure
temp5 = V(:, S(5));
imshow(reshape(temp5, [L L]) ,[])
title("Image for 5th eigen vector")
%% plot error
y = zeros(1, 100);
error = 0;
for j = 10000 : -1 : 101
    error = error + B(j);
end
y(100) = error;
for i = 99 : -1: 1
    y(i) = y(i+1) + B(i+1);
end
figure
plot(y)
ylabel("error")
xlabel("number of eigenvectors")
title("reconstruction error")

%% reconstruct the image with 14 eigenvectors
newP = V(:, S(1:14));
newM = features*newP;
reconstructed = newP*newM';
for i = 1 : 14
    reconstructed(:,i) = reconstructed(:,i) + meanVector';
end
%%
figure
for i = 1:15
    subplot(3, 5, i)
    imshow(reshape(reconstructed(:,i), [L L]) ,[])
    title(sprintf('Image %d(14 eigenvectors)', i))
end
%%
figure
imshow(reshape(reconstructed(:,1), [L L]) ,[])

%% reconstruct the image with 2 eigenvectors
newP1 = V(:, S(1:2));
newM1 = features*newP1;
reconstructed1 = newP1*newM1';
for i = 1 : 2
    reconstructed1(:,i) = reconstructed1(:,i) + meanVector';
end
%%
figure
for i = 1:15
    subplot(3, 5, i)
    imshow(reshape(reconstructed1(:,i), [L L]) ,[])
    title(sprintf('Image %d(2 eigenvectors)', i))
end
%%
figure
imshow(reshape(reconstructed1(:,1), [L L]) ,[])
```

- PCA is a popular technique for image reconstruction that involves reducing the dimensionality of image data by creating a set of principal components that can be used to reconstruct the original

image. Here are some pros and cons of using PCA for image reconstruction:

PROS: 1. PCA is a powerful technique for compressing high-dimensional data, such as images. By creating a smaller set of principal components, we can represent the image with fewer variables, making it easier to store and analyze.

2. PCA retains much of the original information in the image, even after dimensionality reduction. By selecting the most important principal components, we are able to reconstruct a high-fidelity image that is very similar to the original.

3. PCA is relatively easy to implement and can be applied to a wide range of image data without requiring extensive domain-specific knowledge or complex algorithms.

CONS: 1. PCA is a linear technique and may not be able to capture the complex nonlinear relationships that exist within some image data. This can lead to errors and inaccuracies in the reconstructed image.

2. PCA requires a large amount of memory and computational resources in order to process large datasets. This can make it difficult to apply to large-scale image reconstruction tasks.

3. PCA relies on the assumption that the original image data is normally distributed. If the distribution of the data is significantly different, then the performance of PCA may be significantly reduced.

In summary, while PCA is a powerful tool for image reconstruction that provides many benefits, it also has some limitations that need to be taken into consideration when applying it to real-world image processing tasks.

Question 2.5:

Steps:

1. First we extract green areas of the image.
2. Then we erode the result image of step 1 by a disk with length 1 pixel to make holes bigger.
3. In this step we should calculate difference between images of step 1 and step 2 to get boundaries of green area.
4. We will fill the extracted boundaries.
5. make corresponding pixels of non black pixels in image of step 4 red.

The steps are implemented in the following MATLAB code:

```
clc; close all;clear all;
%% extract green area
rgb = imread('q5.png');%load image
HSV = rgb2hsv(rgb);%convert RGB values to HSV
masErodedImage = HSV(:,:,1) < 0.37 & HSV(:,:,1)>0.28;
masboundary = HSV(:,:,2) > 0.9;
mask3 = HSV(:,:,3) > 0.9;
mask = masErodedImage & masboundary & mask3;%green areas
BW = bwareafilt(mask,[0 inf]);% white(green area) and black(other areas)
greenArea = rgb .* uint8(BW);%convert white to green area
%% code for INNER Boundary
mainImage=rgb2gray(greenArea);% Convert RGB into grayscale.
SE=strel('disk',1,0);%structuring element.
ErodedImage=imerode(mainImage,SE);% Erode the image.
boundary= mainImage-ErodedImage;%difference of original and eroded image.
filledHoles = imfill(boundary,'holes');% fill the holes
%% make non black pixels of filledHoles red in main image
final = rgb;
for i = 1:428
    for j = 1:642
```

```
21        if(filledHoles(i, j)~=0)
22            final(i,j, 1) = 255;
23            final(i,j, 2) = 0;
24            final(i,j, 3) = 0;
25        end
26    end
27 end
28 %% figures
29 figure
30 imshow(mainImage), title('original image.');
31 figure
32 imshow(greenArea), title('just green area of main image');
33 figure
34 imshow(ErodedImage), title('eroded image');
35 figure
36 imshow(boundary), title('boundary image');
37 figure
38 imshow(filledHoles), title('filled holes');
39 figure
40 imshow(final), title('result');
```

Step by step results:

Figure 20: Step 1

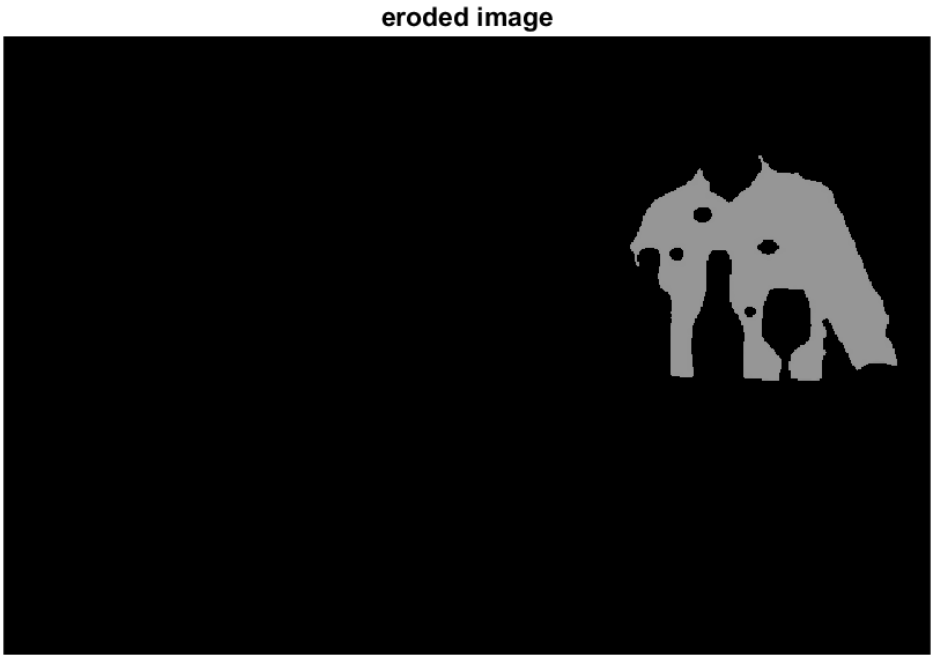


Figure 21: Step 2

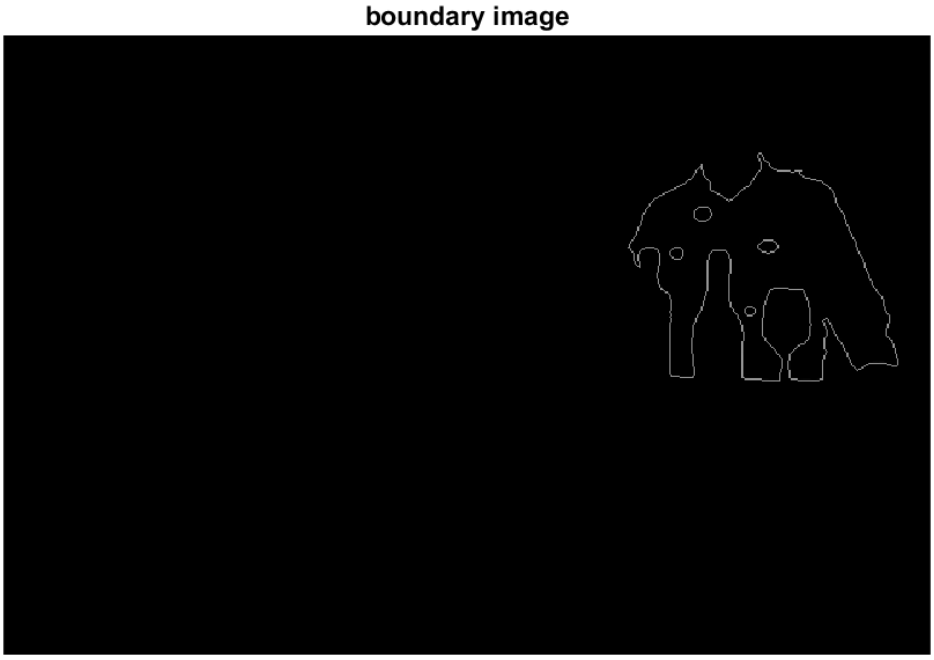


Figure 22: Step 3

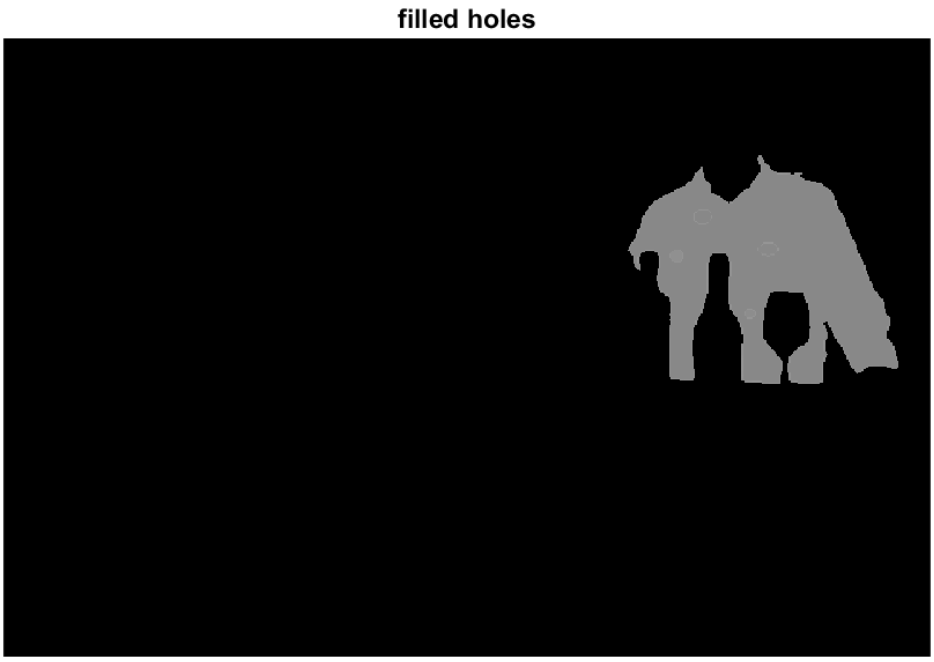


Figure 23: Step 4

Figure 24: Step 5