Medical Image Analysis
Second hand-in

Mariuta Nicolae (rqt629) September 14, 2015

Exercise 1

For this exercise I have written Matlab code (Figure 1) to calculate the Fourier Transform of the requested image, by using the built-in function for Fast Fourier Transform. After this I used the functions *abs* and *angle* to calculate the magnitude and phase for each resulting complex number ($a + bi$) in the frequency domain:

$$abs = \sqrt{a^2 + b^2}$$

$$angle = arctg(\frac{b}{a})$$

```
%% Magnitude and phase of image
clear;
B = my_load_mgh('orig_1.mgz');

%read the desired slice of the image and calculate the Fourier Transform
I = squeeze(B(100,:,:));
Ifft = fft2(I);

%calculate magnitude and phase for the image in frequency domain
Mag = abs(Ifft);
Phase = angle(Ifft);

%reconstruct images
ImMag = abs(ifft2(Mag));
ImPhase = abs((ifft2(Phase)));

%plot the images
subplot(2,2,1),imagesc(I),colormap('gray'),title('original image');
subplot(2,2,2),imagesc(log(1+ImMag)),colormap('gray'),title('magnitude
reconstruction');
subplot(2,2,3),imagesc(I),colormap('gray'),title('original image');
subplot(2,2,4),imagesc(log(1+ImPhase)),colormap('gray'),title('phase
reconstruction');
```

Figure 2. The code for computing the magnitude and phase only versions of slice 100 in the 3d MRI of the brain and then display the results of images reconstructed.

After obtaining the magnitude and phase I reconstructed the two images: one based on magnitude and the other one based on the phase and taken the absolute values of each complex number in the resulting image matrices obtained by the Inverse Fourier Transform, because otherwise I could not display the results.

I also plotted the 4 images as requested after applying the logarithm to each reconstructed image so the variations in pixels intensities to be more visible, and I obtained Figure 2:

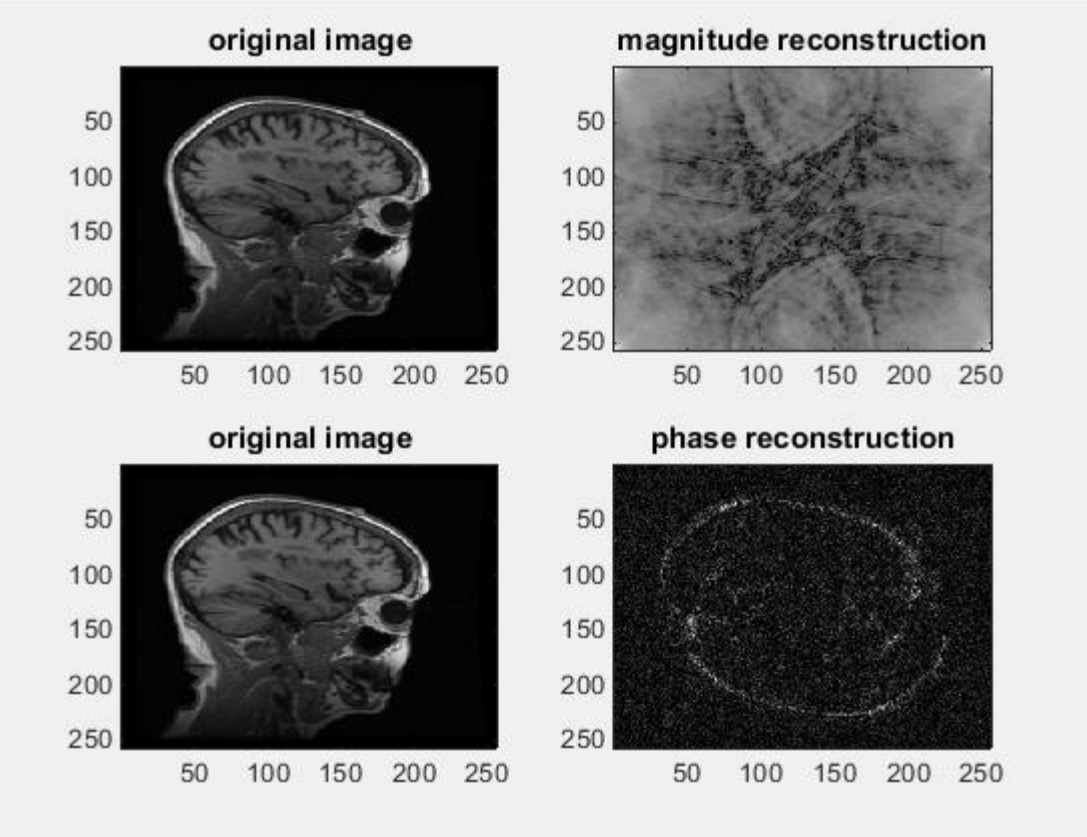


Figure 2. The original image and reconstructions obtained from phase and magnitude

As it can be seen in the resulted pictures, the phase provides more information about the original image: the boundaries of the head and other edges are a bit visible. The magnitude provides information without any meaning. This happens because the phase values determine shift in the sinusoid components of the image and at the location of edges and lines, the sinusoids have the same phase. The phase contains information about the location of the features while the magnitude does not have this information and that is why its information appears to have a strange shape in the reconstruction. I have also seen that in spectral domain the magnitude has most of the high values in the center while in the phase, the sinusoids are visible.

Exercise 2

For this exercise I have created a function because I guess I might need it for future exercises also. It takes as input the image and the deviation of the Gaussian filter. The output is the filtered image.

```
function Ig = my_gaussian_filter(I,d)
%function to apply lowpass filter to an image using a Gaussian kernel
% I -> the original image
% d -> standard deviation of the gaussian

% generate the gaussian filter
g = fspecial('gaussian',size(I,1),d);
g1 = mat2gray(g);

%calculate fourier transform of the image
If = fftshift(fft2(I));
Ifg = If.*g1;

%inverse fourier tranform for obtaining the filtered image
Ig = real(ifft2(fftshift(Ifg)));
```

In the code I have used the matlab built-in function for creating the Gaussian kernel with the desired standard deviation and size of the filter. I hope it is allowed to use imfilter (since it was not specified in the assignment text) but otherwise I could also create it using the Gaussian function to create the matrix (it will certainly have longer computational time if I use the for loops).

For the purpose of filtering, I have used the property that the convolution of an image with a kernel is equal to the multiplication of the two matrices in Fourier domain. Once again, I used the matlab functions for making the FFT in order to multiply with the gaussian and the IFFT function to obtain the filtered image in space domain.

In the same way as I did at the pervious exercise, I have plotted the results by applying the filter with different standard deviations and I have labeled the plot to be easier to view.

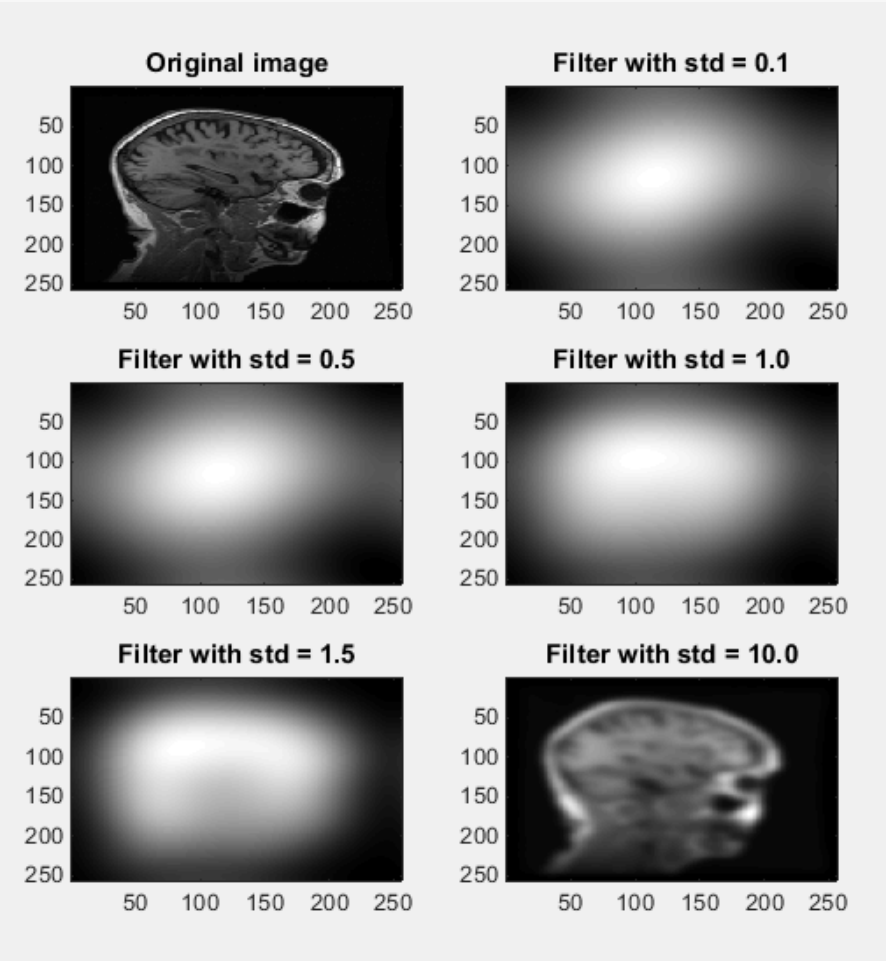


Figure 3 Image filtered with Gaussian kernel of different deviations.

In the results I notice that for the very low standard deviations that are suggested in the assignment text, the image is too blurred to see anything from the original image. This happens because the Gaussian kernel takes only the smallest frequencies from spectrum and it does not return much from the original image. Only at std = 1.5 we start to distinguish a little from the shape of cranium. With a low pass filter with lot larger deviation (10) we can see more features from the original image, but it is also a little blurred. This happens because we remove fewer of the sinusoids with high frequency from the original signal.

Exercise 3

For the histogram equalization, I used the algorithm as it is described on the Wikipedia page, by calculating the histogram, the cumulative histogram and then readjusting the image. I have created separate function for each of these operations:

```
%my function for performing histogram equalization
function Ieq = my_histogram_equalization(I)

%calculate the cumulative histogram
cdf = imChist(I);

%Histogram equalization
Ieq = zeros(size(I));
for R = 1: size(I,1)
    for C = 1: size(I,2)
        Ieq(R,C) = floor(255*cdf(I(R,C)+1));
    end
end

end
```

After performing the histogram equalization, I did as for the previous exercises: I plotted the image before and after equalization and also the histogram and cumulative histogram for each of these:

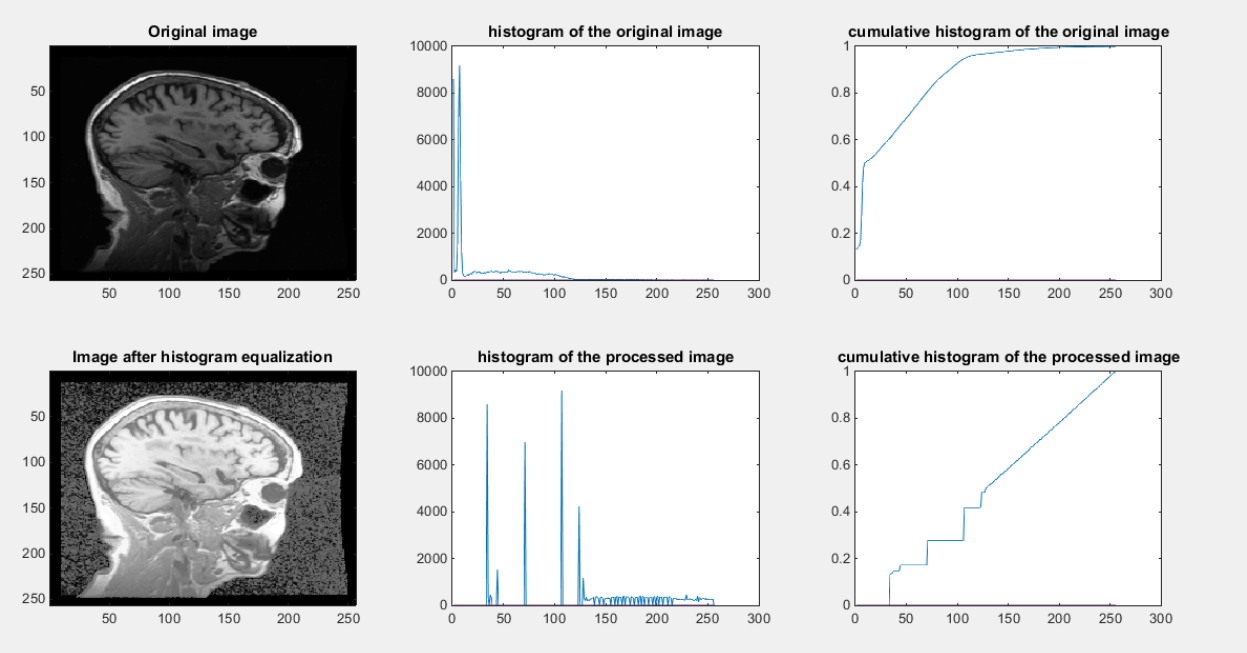


Figure 4. Histogram equalization applied to slice 100, sagittal

For the original image there are some areas with low contrast, mostly in the lower part of it. The histogram shows that there are a large number of pixels with low values and the number of pixels with lower intensities is lower and lower. The cumulative histogram shows a very fast growth in the cumulative distribution function of the image and then in stalls after reaching value 100.

In the image where I applied the histogram equalization, many of the parts of the head became a lot brighter and there it is a lot better contrast in the lower part of the image; features are easier to distinguish. It becomes noisier in the surroundings of the head, that probably because in that area there are many pixels with value different than 0 which were assigned to brighter pixels after equalization. The histogram shows a wider distribution of pixels towards the half of the interval 0-255 and the cumulative distribution function grows a lot slower, with a slope of about 45 degrees with stairs in the lower part where there are high peaks in the histogram.

I can say that the histogram equalization performed well in this case making brain features more visible and the noise in the background does not matter since we are not that interested to process that area.

Exercise 4

For this exercise I used the Matlab built-in functions for applying the median filter and average filter. I have done this because the algorithms are quite easy but a bit hard to implement since I have to take care of edges and it will take too much computational time by using 4 for loops (for each pixel in the image I have to take values for all pixels in the window size).

For the median filter I used the function *medfilt2* that takes as input the image and the size of the window. For the average filter I created the convolution window filled with values 1/n where n is the size of the matrix and then used it for the function *imfilter*. In order to avoid the drawbacks of implementing this in space domain, this can be also done by calculating the FFT of both matrices (original image and kernel) and then multiply them in frequency domain.

I have used for both filters window of size 3x3, 5x5, 7x7, 11x11 and 19x19 and displayed the results to it will be easy to compare.

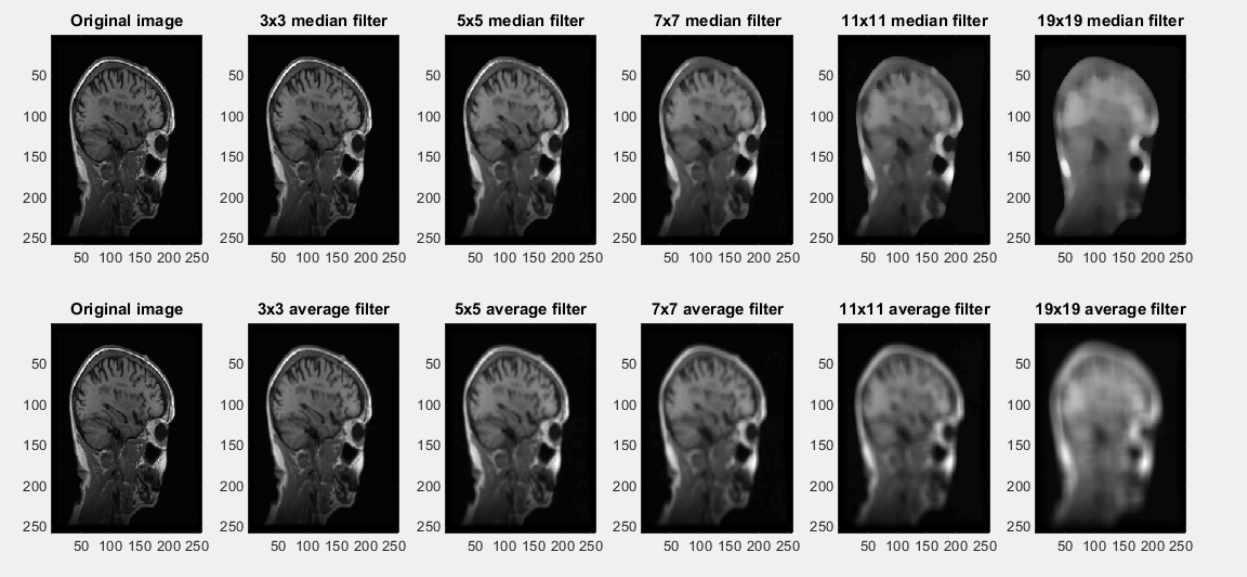


Figure 5. Median and average filter applied to slice 100, sagittal

For using each filter I see significant change in the image for window of size 5x5 or greater. For very large filter 19x19 both images are very blurred.

I can notice that the median filter preserves the sharp edges better than the median filter. This happens because the median filter is more robust and very unrepresentative pixels in the matrix are ignored while these make a significant change at the average filter. Because it actually takes the value of one pixel in the neighborhood, the median filter does not create new unrealistic pixels and that is why the filter preserves better the edges while these become very blurred in the average filtering.

I can say that the median filter deals better with details that have high spatial frequency while being very effective at removing the noise in the image. The disadvantage of the median filter is the larger computational time. For each step we have to take all pixels in the kernel, sort them and then take the median one. The filtering is done lot faster using the average filter if we use the multiplication in frequency domain.

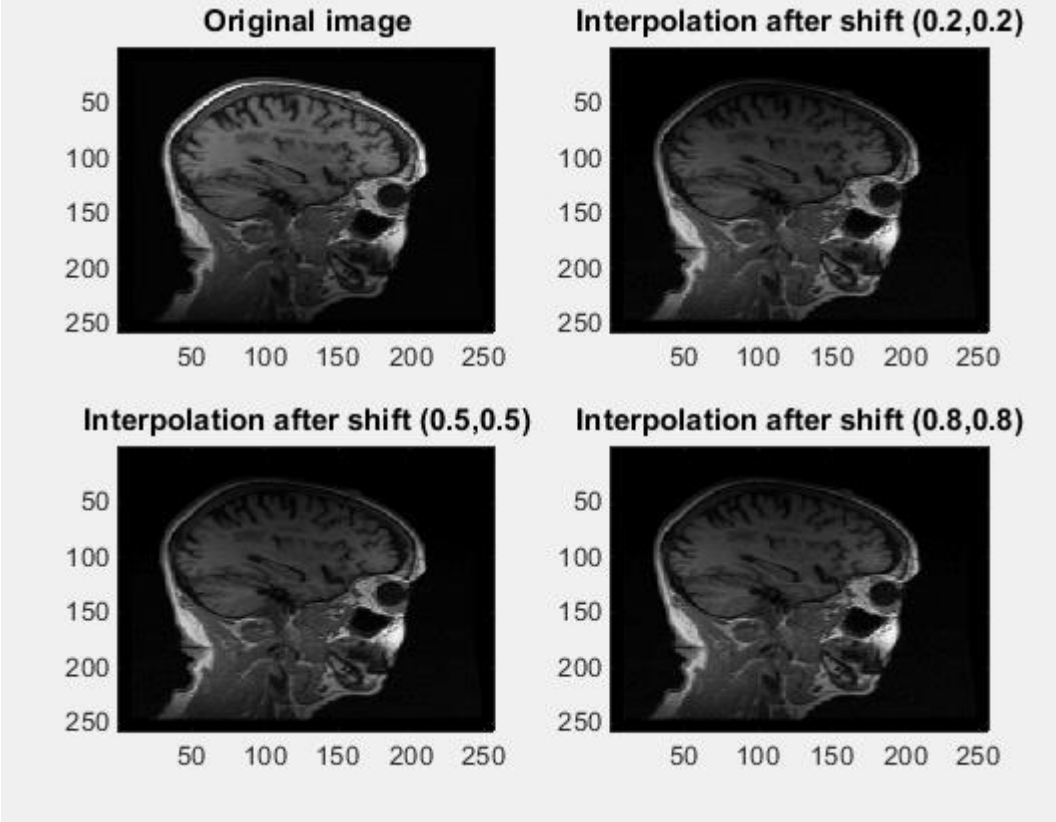
Exercise 5

For the piecewise linear interpolation, I created the function *my_interpolation* that takes as input the original image with the list of points in the grid and the new points where we need to interpolate.

```
%my function for interpolation. Input parameters: original image, original
%grid and the new points
function Interpolate = my_interpolation(I, pts, ptsShift)
% the interpolate image
Interpolate = zeros(size(I));

for i = 1:size(pts,1)
   xi = pts(i,1);yi = pts(i,2);
   x = ptsShift(i,1);y = ptsShift(i,2);

   %calculate weights for pixels in original image:
    w11 = ((xi+1) - x)*((yi+1) - y);
    w21 = (x - xi)*((yi+1) - y);
    w12 = ((xi+1) - x)*(y - yi);
    w22 = (x - xi)*(i - yi);

    %calculate interpolation at point (xi,yi) in the new grid
    Interpolate(xi,yi) = I(xi,yi)*w11 + I(xi+1,yi)*w21 +  I(xi,yi+1)*w12 +
I(xi+1,yi+1)*w22/(((xi+1)-xi)*((yi+1)-yi));
end

end
```

I have mostly used the first algorithm shown on Wikipedia page about Billinear Interpolation (https://en.wikipedia.org/wiki/Bilinear_interpolation) and I have basically implemented the equation shown there, but I have split it to calculate the weights of the old points and then I calculated the actual interpolation.

For being able to visualize how my function work, I translated the old points with different amount on x and y coordinate by adding to each values: (0.2,0.2) , (0.5,0.5), (0.8,0.8) and then I plotted the results.



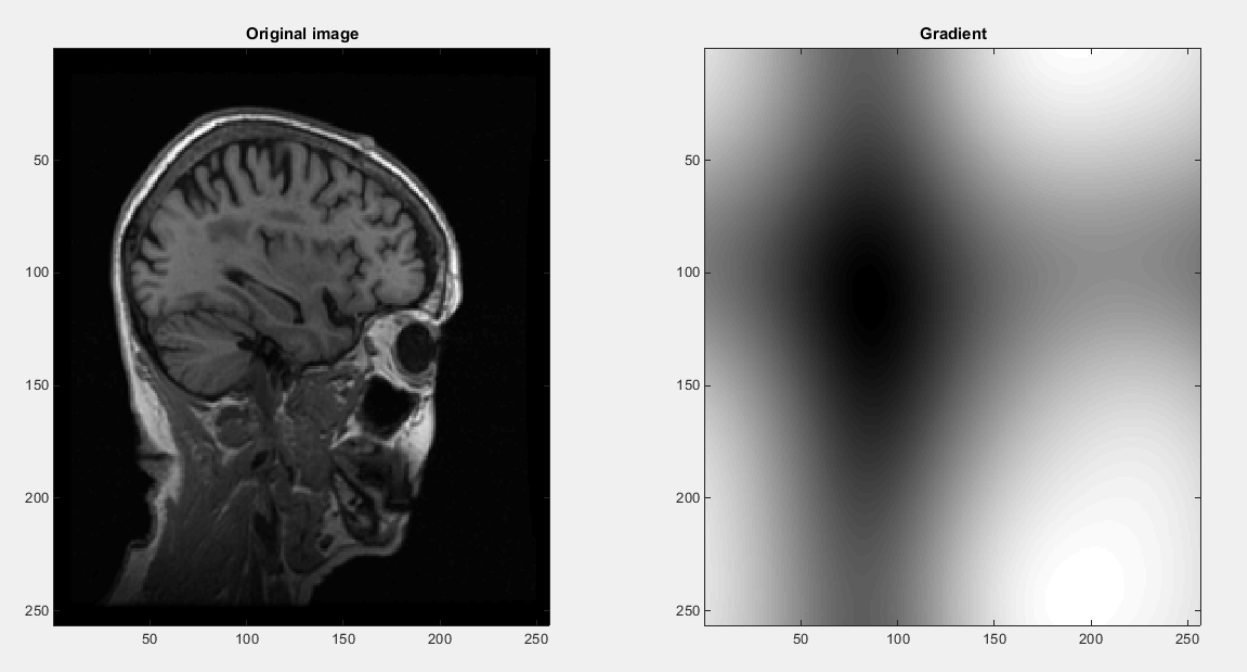
Since we calculate a weighted average of the neighboring pixels, the interpolation shows an image with lower contrast, slightly different from the original image, so it might be better to also apply some filtering after using the interpolation.

Used alone, the interpolation does not have a good effect on the original image but it is certainly necessary for transformations where resampling is needed like rotation, translation or scaling when we no longer have integral coordinates for the points and we want to calculate the pixel values of the new points but as noticed, perhaps a histogram equalization should be performed to restore the original contrast of the image.

Exercise 6

For the gradient of the image, I used the method with Sobel operator that is show on Wikipedia (https://en.wikipedia.org/wiki/Sobel_operator) by using the 2 kernels for obtaining the gradient on both: x and y coordinate. In order to reduce the computational time and also make it easier to implement, I calculated the FFT of the original image and the 2 gradient kernels. After this, I have done the multiplication in frequency domain and them I calculated the Inverse Fourier Transforms.

I have used the equation $G = \sqrt{G_x^2 + G_y^2}$ and plotted the result.



Unfortunately, the results for my implementation do not show the edges as shown in the documentation I read and I am pretty sure I have done a mistake somewhere.

I think that my implementation is not very wrong and there is a small mistake in building the matrices and unfortunately, I did not have enough time to fix it and make it work properly.

Exercise 7

The purpose of this paper is to show a way to deal with the differences in intensity homogeneity between serial scans by performing the bias field calculation and then make correction. The technique does not make assumptions about the signal distribution, bias field or signal homogeneity. It is used the fact that the difference image of registered serial scans has small-scale structure. The method show has applications in measuring the atrophy for patients with dementia.

The detection in rates of atrophy is very important in order to measure of atrophy progression and it is very important that the detection to be very accurate.

Because of different factors like inhomogeneity in magnetic field there appear artifacts in the MR images that cause intensity inhomogeneity. Also called bias, it represents the slowly changing and spatial variation that appears within the scan.

This effect can have the strength up to 20% and this causes problems in the techniques for the MR image analysis, like applying segmentation, registration performed on serial scans or intensity-quantification based techniques.

There are two types of techniques for intensity correction homogeneity: techniques that use modified image acquisition protocols to handle bias correction at the source or post processing methods.

The first category of techniques does not manage to eliminate distortions caused by the object to be scanned and these methods require increased scan acquisition time. Post processing techniques are based on the assumption of spatial homogeneity.

But the technique presented in this paper does not make any assumptions about tissue type, form of the bias field or regional homogeneity. Instead, the technique use the difference image between longitudinal scans of a patient, which can be considered to consist only of noise and atrophy and the differential bias field to identify the differential bias field by the application of simple filters.

For the algorithm, it is assumed that the image is given by:

$$v(x) = u(x)b(x) + n(x)$$

where v(x) is the measured intensity, u(x) is the true signal intensity, b(x) is the bias field and n(x) is the noise.

In the next stage, log transform is applied to the registered images with intensities v1(x) and v2(x) and the difference is calculated between the resulting matrices. The result of this operation includes the original additive noise of the system, the difference between the true signals in the images and the differential bias field.

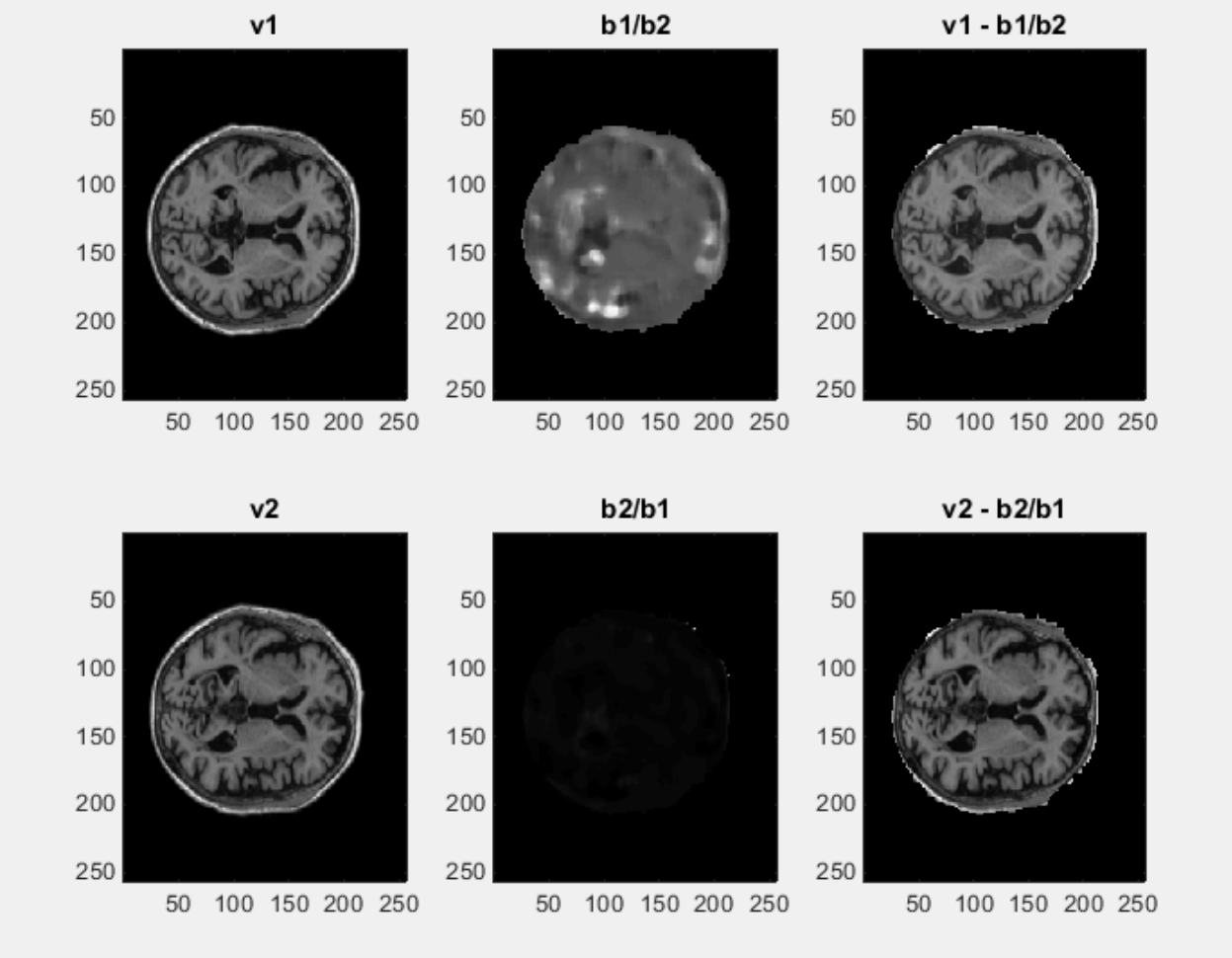
To obtain the differential bias field, the median filter is applied to the difference image that removes the Gaussian noise. A kernel with size 11x11x11 is used to erase any anatomical structure difference but it is still enough to obtain the differential bias accurately.

For my implementation in Matlab I have applied the algorithm as it is described in the paper and by using some basic function from the Maltab and I have taken slice 100 from coronal plane.

```
%% Exercise 7: DBC implementation
clear;
%load the image
v1 = my_load_mgh('images/nu_1.mgz');
Iv1 = squeeze(v1(:,100,:));

v2 = my_load_mgh('images/nu_2.mgz');
Iv2 = squeeze(v2(:,100,:));

%log transform of the images
lv1 = log(Iv1);
lv2 = log(Iv2);
```

```
%calculate the differential bias field
diff1 = lv1-lv2;
diff2 = lv2-lv1;
med_diff1 = medfilt2(diff1, [11 11]);
med_diff2 = medfilt2(diff2, [11 11]);
ratio1 = exp(med_diff1);
ratio2 = exp(med_diff2);

%plot the results
subplot(2,3,1),imagesc(Iv1),colormap('gray'),title('v1');
subplot(2,3,4),imagesc(Iv2),colormap('gray'),title('v2');
subplot(2,3,2),imagesc(ratio1),colormap('gray'),title('b1/b2');
subplot(2,3,5),imagesc(ratio2),colormap('gray'),title('b2/b1');
subplot(2,3,3),imagesc(Iv1-ratio2),colormap('gray'),title('v1 - b1/b2');
subplot(2,3,6),imagesc(Iv2-ratio1),colormap('gray'),title('v2 - b2/b1');
```

By running this code and displaying the results, I obtain:



The technique removes much of the differential homogeneity field and now we can actually compare the two sequential images of the patient. In the article it is told that with their experiments the obtained good results even if registration of the images had large errors. In our case there is not so much inhomogeneity between the two images so the differences are not so visible.

Exercise 10

9.2.

The main difference between these 2 types of methods comes from that the spatial filtering methods manipulate pixel values in spatial domain according to how they are located in the whole image or local regions. In contrast, the frequency domain methods manipulate the information provided by the frequency domain.

Usually, the spatial domain methods run faster when it is about simple operations like adding, subtracting, histogram transformation because these do not require a Fourier Transform that takes some computational time and maybe also the inverse of the transform to view the results of the filtering. However, some local spatial methods that need kernels in spatial domain, like median filter may run slower because there is an iterative operation that has to be done for each pixel by reading the values of neighboring pixels and that sometimes take longer time than some frequency domain filtering methods.

The spatial domain filtering methods may prove to be more effective in noise removal when we know some information about the frequencies of sinusoids in the image. The band-pass filters (usually in high or low ranges) provide very good noise removal if we know the frequency of a noise component. And applying convolution in frequency domain is a lot faster than doing the same operation in spatial domain.

Each category of filtering provides better results in specific cases but the computational times are also strongly considered in many applications.

9.3.

I already did this for the Exercise 4 and I have shown that the median filter provides better denoising at the expense of computational time.

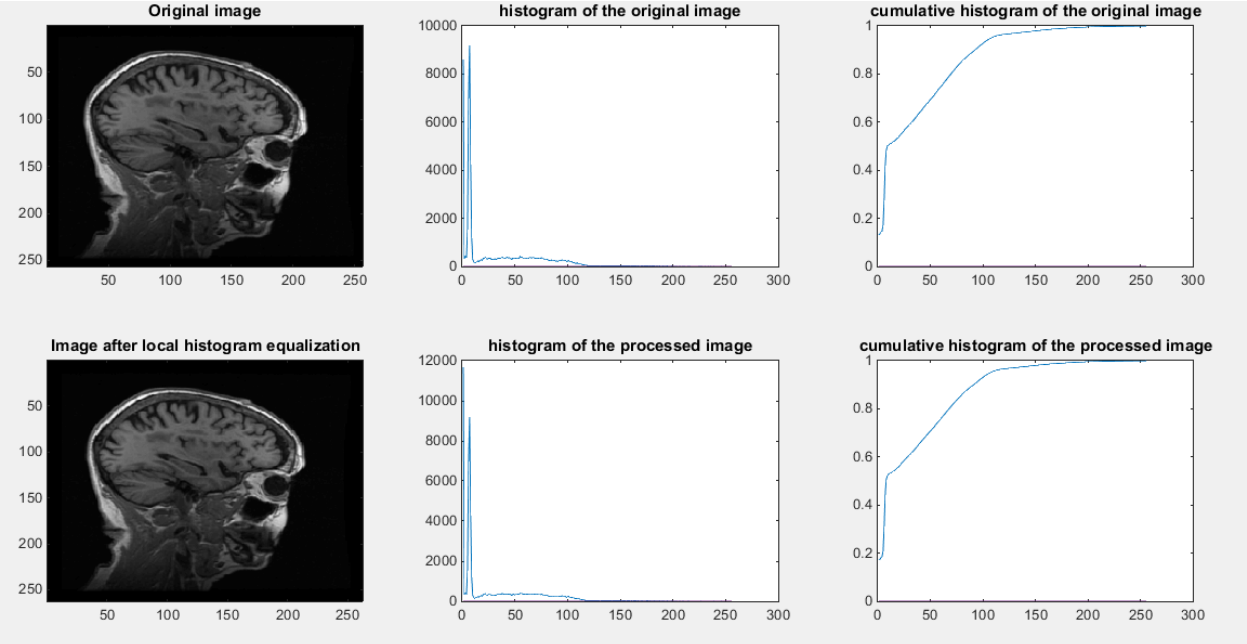
9.4.a.

I have already done this for Exercise 3 and I have shown that histogram equalization obtains an image with better contrast

9.4. d.

For this question, I take all pixels in a window 7x7 around each pixel and then calculate the cumulative histogram for the pixels in that window and then adjust the value of the pixel in the middle of the window in the same way I did for global histogram equalization.

Again, I plotted the original image, the equalized image , histogram and cumulative histogram for each image.



In the histograms for each image there is not much difference, but looking at the images themselves I notice that the contrast is improved, especially in the lower part of the image, but there is no longer that noise on the background that I had for the normal histogram equalization. This happens as described in the course book, so it provides better results than the global histogram equalization.

Anexes with the Matlab code

Exercise 1

```
%% Exercise 1: Magnitude and phase of image
clear;
B = my_load_mgh('orig_1.mgz');

%read the desired slice of the image and calculate the Fourier Transform
I = squeeze(B(100,:,:));
Ifft = fft2(I);

%calculate magnitude and phase for the image in frequency domain
Mag = abs(Ifft);
Phase = angle(Ifft);

%reconstruct images
ImMag = abs(ifft2(Mag));
ImPhase = abs((ifft2(Phase)));

%plot the images
subplot(2,2,1),imagesc(I),colormap('gray'),title('original image');
subplot(2,2,2),imagesc(log(1+ImMag)),colormap('gray'),title('magnitude
reconstruction');
subplot(2,2,3),imagesc(I),colormap('gray'),title('original image');
subplot(2,2,4),imagesc(log(1+ImPhase)),colormap('gray'),title('phase
reconstruction');
```

Exercise 2

```
%% Exercise2: Low pass filter using Gaussian
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));

%apply low pass filter using different fitlers
Ig1 = my_gaussian_filter(I,0.1);
Ig2 = my_gaussian_filter(I,0.5);
```

```
Ig3 = my_gaussian_filter(I,1.0);
Ig4 = my_gaussian_filter(I,1.5);
Ig5 = my_gaussian_filter(I,10.0);

%plot the results
subplot(3,2,1);imagesc(I),colormap('gray'),title('Original image');
subplot(3,2,2);imagesc(Ig1),colormap('gray'),title('Filter with std = 0.1');
subplot(3,2,3);imagesc(Ig2),colormap('gray'),title('Filter with std = 0.5');
subplot(3,2,4);imagesc(Ig3),colormap('gray'),title('Filter with std = 1.0');
subplot(3,2,5);imagesc(Ig4),colormap('gray'),title('Filter with std = 1.5');
subplot(3,2,6);imagesc(Ig5),colormap('gray'),title('Filter with std = 10.0');
```

Exercise 3

```
%% Exercise 3: Histogram Equalization
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));

%calculate histogram
px = my_histogram(I);
%calculate cumulative histogram
cdf = imChist(I);
%perform histogram equalization
Ieq = my_histogram_equalization(I);

%calculate histogram of the equalized image
pxeq = my_histogram(Ieq);
cdfeq = imChist(Ieq);


%displaying results
subplot(2,3,1),imagesc(I),colormap('gray'),title('Original image');
subplot(2,3,2),plot(1:256,px),title('histogram of the original image');
subplot(2,3,3),plot(1:256, cdf),title('cumulative histogram of the original
image');
subplot(2,3,4),imagesc(Ieq),colormap('gray'),title('Image after histogram
equalization');
subplot(2,3,5),plot(1:256,pxeq),title('histogram of the processed image');
subplot(2,3,6),plot(1:256, cdfeq),title('cumulative histogram of the
processed image');
```

Exercise 4

```
%% Exercise 4: Median and average filter
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));

%median filtering
Im3 = medfilt2(I, [3 3]);
Im5 = medfilt2(I, [5 5]);
Im7 = medfilt2(I, [7 7]);
Im11 = medfilt2(I, [11 11]);
Im19 = medfilt2(I, [19 19]);

%average filtering
h3 = ones(3,3)/9;Ia3 = imfilter(I,h3);
h5 = ones(5,5)/25;Ia5 = imfilter(I,h5);
h7 = ones(7,7)/49;Ia7 = imfilter(I,h7);
h11 = ones(11,11)/121;Ia11 = imfilter(I,h11);
h19 = ones(19,19)/361;Ia19 = imfilter(I,h19);

%displaying results
subplot(2,6,1),imagesc(I),colormap('gray'),title('Original image');
subplot(2,6,2),imagesc(Im3),colormap('gray'),title('3x3 median filter');
subplot(2,6,3),imagesc(Im5),colormap('gray'),title('5x5 median filter');
subplot(2,6,4),imagesc(Im7),colormap('gray'),title('7x7 median filter');
subplot(2,6,5),imagesc(Im11),colormap('gray'),title('11x11 median filter');
subplot(2,6,6),imagesc(Im19),colormap('gray'),title('19x19 median filter');

subplot(2,6,7),imagesc(I),colormap('gray'),title('Original image');
subplot(2,6,8),imagesc(Ia3),colormap('gray'),title('3x3 average filter');
subplot(2,6,9),imagesc(Ia5),colormap('gray'),title('5x5 average filter');
subplot(2,6,10),imagesc(Ia7),colormap('gray'),title('7x7 average filter');
subplot(2,6,11),imagesc(Ia11),colormap('gray'),title('11x11 average filter');
subplot(2,6,12),imagesc(Ia19),colormap('gray'),title('19x19 average filter');
```

Exercise 5

```
%% Exercise 5: Piecewise Linear Interpolation
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));
```

```
%create the original grid
[Xgrid, Ygrid] = meshgrid(1:(size(I,1)-2),1:(size(I,2)-2));
Xgrid = Xgrid+1;
Ygrid = Ygrid+1;
pts = [Xgrid(:) Ygrid(:)];

%create the new grid for interpolate
XgridShift1 = Xgrid +0.2;
YgridShift1 = Ygrid +0.2;
ptsShift1 = [XgridShift1(:) YgridShift1(:)];

XgridShift2 = Xgrid +0.5;
YgridShift2 = Ygrid +0.5;
ptsShift2 = [XgridShift2(:) YgridShift2(:)];

XgridShift3 = Xgrid +0.8;
YgridShift3 = Ygrid +0.8;
ptsShift3 = [XgridShift3(:) YgridShift3(:)];


%apply interpolation
Interpolate1 = my_interpolation(I,pts,ptsShift1);
Interpolate2 = my_interpolation(I,pts,ptsShift2);
Interpolate3 = my_interpolation(I,pts,ptsShift3);

subplot(2,2,1),imagesc(I),colormap('gray'),title('Original image');
subplot(2,2,2),imagesc(Interpolate1),colormap('gray'),title('Interpolation
after shift (0.2,0.2)');
subplot(2,2,3),imagesc(Interpolate1),colormap('gray'),title('Interpolation
after shift (0.5,0.5)');
subplot(2,2,4),imagesc(Interpolate1),colormap('gray'),title('Interpolation
after shift (0.8,0.8)');
```

Exercise 6

```
%% Exercise 6: Gradient of an image by using Sobel operator
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));

%define the Sobel operator kernels
Gx = [-1 0 1 ; -2 0 2 ; -1 0 1];
Gy = [-1 -2 -1 ; 0 0 0 ; 1 2 1];

%fourier transform of the image and the kernels
Ifft = fft2(I);

Gxfft = fft2(Gx);
Gxfft2 = zeros(size(I));
xpos = 1; ypos = 1;
```

```
Gxfft2(xpos:xpos+3-1,ypos:ypos+3-1) = Gxfft;

Gyfft = fft2(Gy);
Gyfft2 = zeros(size(I));
xpos = 1; ypos = 1;
Gyfft2(xpos:xpos+3-1,ypos:ypos+3-1) = Gyfft;

%apply convolution in fourier domain
Gradxfft = Gxfft2.*Ifft;
Gradyfft = Gyfft2.*Ifft;

%ifft for each gradient
Gradx = abs(ifft2(Gradxfft));
Grady = abs(ifft2(Gradyfft));

Igrad = sqrt(Gradx.^2 + Grady.^2);

subplot(1,2,1),imagesc(I),colormap('gray'),title('Original image');
subplot(1,2,2),imagesc(Igrad),colormap('gray'),title('Gradient');
```

Exercise 7

```
%% Exercise 7: DBC implementation
clear;
%load the image
v1 = my_load_mgh('images/nu_1.mgz');
Iv1 = squeeze(v1(:,100,:));

v2 = my_load_mgh('images/nu_2.mgz');
Iv2 = squeeze(v2(:,100,:));

%log transform of the images
lv1 = log(Iv1);
lv2 = log(Iv2);

%calculate the differential bias field
diff1 = lv1-lv2;
diff2 = lv2-lv1;
med_diff1 = medfilt2(diff1, [11 11]);
med_diff2 = medfilt2(diff2, [11 11]);
ratio1 = exp(med_diff1);
ratio2 = exp(med_diff2);

%plot the results
subplot(2,3,1),imagesc(Iv1),colormap('gray'),title('v1');
subplot(2,3,4),imagesc(Iv2),colormap('gray'),title('v2');
subplot(2,3,2),imagesc(ratio1),colormap('gray'),title('b1/b2');
subplot(2,3,5),imagesc(ratio2),colormap('gray'),title('b2/b1');
subplot(2,3,3),imagesc(Iv1-ratio2),colormap('gray'),title('v1 - b1/b2');
subplot(2,3,6),imagesc(Iv2-ratio1),colormap('gray'),title('v2 - b2/b1');
```

Exercise 10 9.4.d.

```
%% Question 9.4.d.
clear;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(100,:,:));

%window size
M = 7;
N = 7;
mid_val=round((M*N)/2);

%find the number of rows and columns to be padded with zero
in=0;
for i=1:M
    for j=1:N
        in=in+1;
        if(in==mid_val)
            PadM=i-1;
            PadN=j-1;
            break;
        end
    end
end

%padding the image with zero on all sides
B=padarray(I,[PadM,PadN]);


for i= 1:size(B,1)-((PadM*2)+1)

    for j=1:size(B,2)-((PadN*2)+1)
        cdf=zeros(256,1);
        inc=1;
        for x=1:M
            for y=1:N
  %find the middle element in the window
                if(inc==mid_val)
                    ele=B(i+x-1,j+y-1)+1;
                end
                    pos=B(i+x-1,j+y-1)+1;
                    cdf(pos)=cdf(pos)+1;
                   inc=inc+1;
            end
        end

        %compute the cdf for the values in the window
        for l=2:256
            cdf(l)=cdf(l)+cdf(l-1);
        end
            Img(i,j)=round(cdf(ele)/(M*N)*255);
     end
end
```

```
%calculate histogram
px = my_histogram(I);
%calculate cumulative histogram
cdf = imChist(I);

%calculate histogram of the equalized image
pxeq = my_histogram(B);
cdfeq = imChist(B);


%displaying results
subplot(2,3,1),imagesc(I),colormap('gray'),title('Original image');
subplot(2,3,2),plot(1:256,px),title('histogram of the original image');
subplot(2,3,3),plot(1:256, cdf),title('cumulative histogram of the original
image');
subplot(2,3,4),imagesc(B),colormap('gray'),title('Image after local histogram
equalization');
subplot(2,3,5),plot(1:256,pxeq),title('histogram of the processed image');
subplot(2,3,6),plot(1:256, cdfeq),title('cumulative histogram of the
processed image');
```

Image Cumulative Histogram

```
%my function to calculate cumulative histogram of an image
function C = imChist(I)

%calculate histogram
H = my_histogram(I);
C = cumsum(H);
count = size(I,1)*size(I,2);

for i = 1 : size(C)
    C(i) = C(i)/count;
end

end
```

Gaussian filter

```
function Ig = my_gaussian_filter(I,d)
%function to apply lowpass filter to an image using a Gaussian kernel
% I -> the original image
% d -> standard deviation of the gaussian

% generate the gaussian filter
```

```
g = fspecial('gaussian',size(I,1),d);
g1 = mat2gray(g);

%calculate fourier transform of the image
If = fftshift(fft2(I));
Ifg = If.*g1;

%inverse fourier tranform for obtaining the filtered image
Ig = real(ifft2(fftshift(Ifg)));
```

Histogram calculation

```
%my function to calculate histogram of an image with pixels intensities
%from 0 to 255
function H = my_histogram(I)

%first element in H is count for pixel intensities 0;
%the last one is for pixel intensity 255
H = zeros(256);
for R = 1: size(I,1)
    for C = 1: size(I,2)
        H(I(R,C)+1) = H(I(R,C)+1) +1;
    end
end

end
```

Histogram equalization

```
%my function for performing histogram equalization
function Ieq = my_histogram_equalization(I)

%calculate the cumulative histogram
cdf = imChist(I);

%Histogram equalization
Ieq = zeros(size(I));
for R = 1: size(I,1)
    for C = 1: size(I,2)
        Ieq(R,C) = floor(255*cdf(I(R,C)+1));
    end
end

end
```

Interpolation

```
%my function for interpolation. Input parameters: original image, original
%grid and the new points
function Interpolate = my_interpolation(I, pts, ptsShift)
% the interpolate image
Interpolate = zeros(size(I));

for i = 1:size(pts,1)
   xi = pts(i,1);yi = pts(i,2);
   x = ptsShift(i,1);y = ptsShift(i,2);

   %calculate weights for pixels in original image:
    w11 = ((xi+1) - x)*((yi+1) - y);
    w21 = (x - xi)*((yi+1) - y);
    w12 = ((xi+1) - x)*(y - yi);
    w22 = (x - xi)*(i - yi);

    %calculate interpolation at point (xi,yi) in the new grid
    Interpolate(xi,yi) = I(xi,yi)*w11 + I(xi+1,yi)*w21 +  I(xi,yi+1)*w12 +
I(xi+1,yi+1)*w22/(((xi+1)-xi)*((yi+1)-yi));
end

end
```