EL-GY 6123 Image and Video Processing
Matlab Assignment 2

Name: Amitesh Kumar Sah
NYU ID: N19714360

Question 1

Write a Matlab program for implementing filtering of a gray scale image. Your program should allow you to specify the filter with an arbitrary size (but for simplicity, you can assume the filter size is KxL where both K and L are odd numbers, and the filter origin is at the center. Your program should read in a gray scale image, perform the filtering, display the original and filtered image, and save the filtered image into another file. You should write a separate function for the convolution that can be called by your main program. For simplicity, you can use the simplified boundary treatment. You should properly normalize the filtered image so that the resulting image values can be saved as 8-bit unsigned characters. Apply the filters given in the previous problem to a test image. Observe on the effect of these filters on your image. Note: you cannot use the MATLAB conv2( ) function. In your report, include your MATLAB code, the original test image and the images obtained with the three filters. Write down your observation of the effect of the filters.

Program:

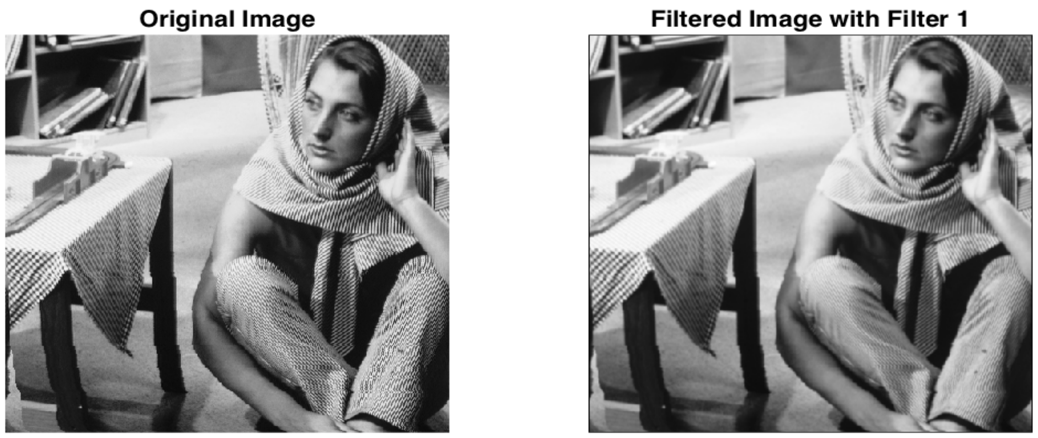
```
clc;clear all; close all;
% i) Your program should read in a gray scale image
inImg = imread('barbara_gray.bmp');

%convert the inImg to int or single or double before proceeding!
grayImg=single(inImg);

% Enter the dimension of filter
k=input('Enter K for KxL filter size(ODD LENGTH)');
l=input('Enter L for KxL filter size(ODD LENGTH)');
my_filter=zeros(k,l);

% Enter the filter value
for i=1:k
    my_filter(i,:)=input('Enter Row in [-1 2 3] form for KxL filter');
end

%perform convolution
tmpImg = my_conv2(grayImg, my_filter);

%% scale tmpImg so that it ranges in 0 to 255 and is stored in uint8
filteredImg=uint8(255*my_mat2gray(tmpImg));

%% iv) display the original and the filtered images. you can use imshow for displaying an image.
%% make sure to conver all images into uint8 before dispalying them
cf=figure(1);
subplot(1,2,1);
imshow(inImg);
title('Original Image');
subplot(1,2,2);
```

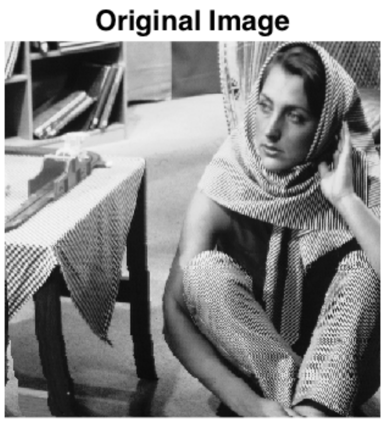
```
imshow(filteredImg);
title('Filtered Image with Filter 1');

% v) save the filtered images into another file.
print(cf, 'HW2_Q1', '-dtiff');
```

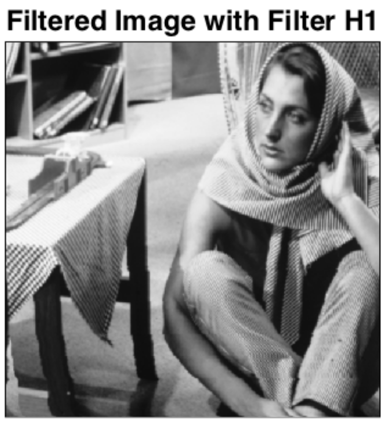
Function for convolution **my_conv2()**

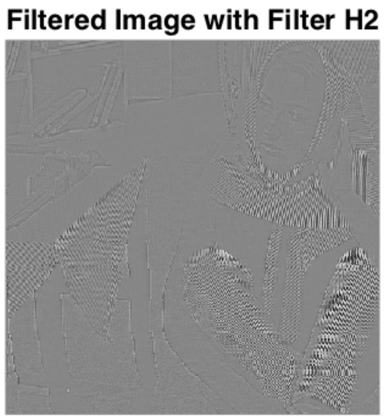
```
function f=my_conv2(grayImg,my_filter)
[xh xw] = size(grayImg);
[hh hw]=size(my_filter);
hhh=(hh-1)/2;
hhw=(hw-1)/2;
z=zeros(xh,xw);

for m=hhh+1:xh-hhh
 for n = hhw+1:xw-hhw
%skip first and last hhw columns to avoid boundary problems
 tmpv = 0;
 for k = -hhh:hhh
 for l = -hhw:hhw
 tmpv = tmpv + grayImg(m-k,n-l)*my_filter(k+hhh+1,l+hhw+1);
 %h(0,0) is stored in h(hhh+1,hhw+1)
 end
 end
 z(m, n) = tmpv;
 end
end
f=z;
end
```

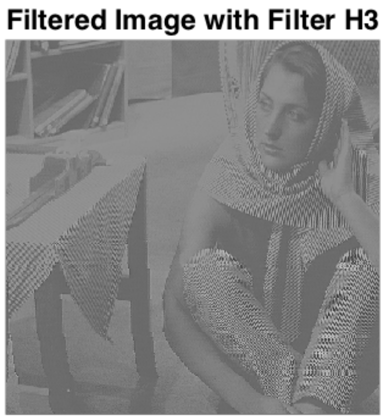
Filtered Image (size of filter =3*3)











Applying the three filter H1, H2, H3 to the original Image from question 5 of written assignment.
When H1 filter is used, image becomes blur. It smoothens the edges, therefore acts as low pass filter.
When H2 is used, we can see both the horizontal and vertical edges and also diagonal edges.
When H3 is used, it converts the image to low contrast dark image.

Question 2

Write a Matlab to simulate noise removal. First create a noisy image, by adding zero mean Gaussian random noise to your image using "imnoise()". You can specify the noise variance in "imnoise( )"). Then apply an averaging filter to the noise added image. For a chosen variance of the added noise, you need to try different window sizes (from 3x3 to 9x9) to see which one gives you the best trade-off between noise removal and blurring. Hand in your program, the original noise-added images at two different noise levels (0.01 and 0.1) and the corresponding filtered images with the best window sizes. Write down your observation. For the filtering operation, if your program in Prob. 1 does not work well, you could use the matlab "conv2()" function. Your program should allow the user to specify the window size as an input parameter.

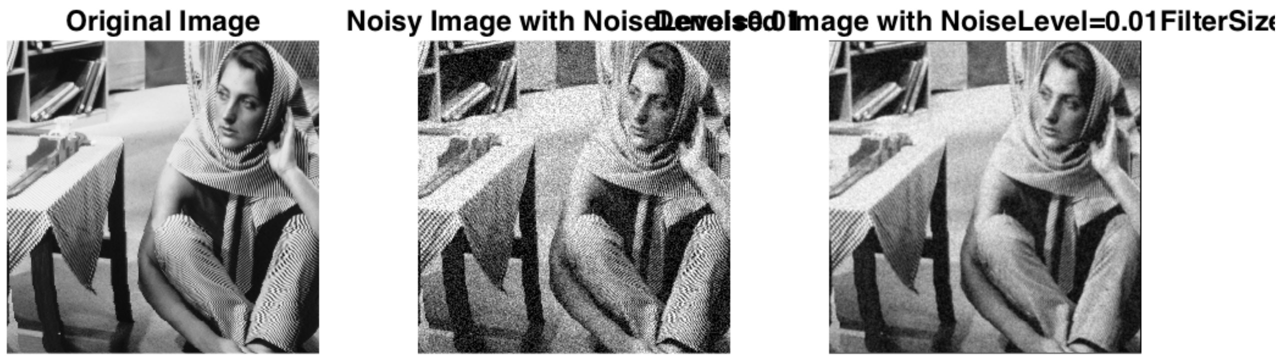
Solution:

Program

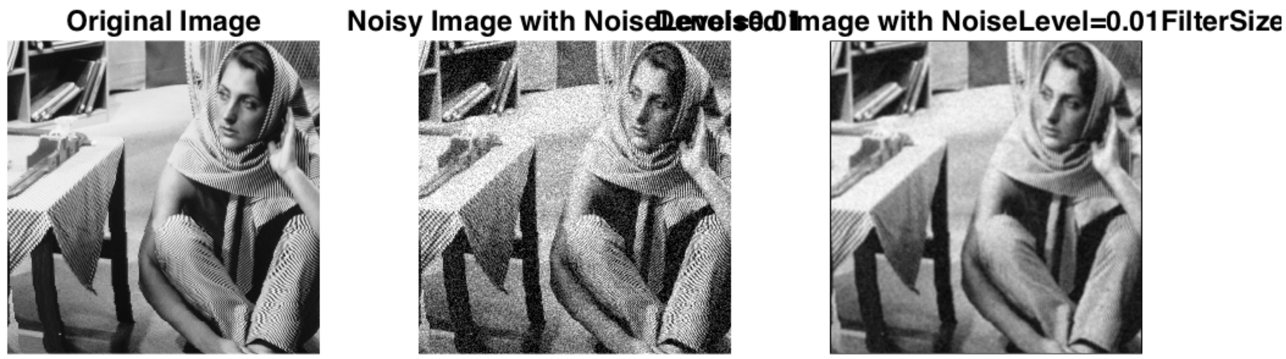
```
clc;clear all;close all;
inImg = imread('barbara_gray.bmp');

%convert the inImg to int or single or double before proceeding!
grayImg=double(inImg);
```

```
noiseLevel=0.01;
noisy_img = imnoise(inImg, 'gaussian', 0, noiseLevel);
%imnoise does not work when inImg is not uint8!
noisy_img=single(noisy_img);

%%
%set filter
filterSize=input('Enter the filter size');

% filterSize=3;
denoising_filter=ones(filterSize,filterSize)/(filterSize*filterSize);
tmpImg = my_conv2(noisy_img, denoising_filter);

% III) Normalize and convert the image to uint8
denoisedImg = 255*my_mat2gray(tmpImg);

%% iv) display the original and the filtered images.
cf=figure(1);
subplot(1,3,1);
imshow(inImg);
title('Original Image');
subplot(1,3,2);
imshow(noisy_img,[]);
title(['Noisy Image with ' num2str(noiseLevel)]);
%note you need to use [] option to see the full range of noisy image
subplot(1,3,3);
imshow(denoisedImg,[]);
title(['Denoised Image with ' num2str(noiseLevel) 'FilterSize=' num2str(filterSize)]);

% v) save the filtered images into another file
print(cf, 'HW2_Q2_fsize3', '-dtiff');
```

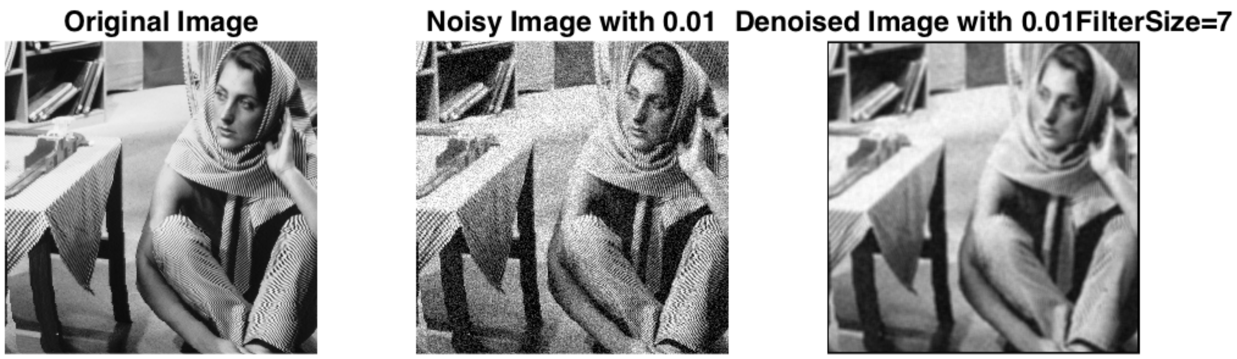
Different window size result:
Window size 3


Original Image
Noisy Image with NoiseLevel=0.01
Denoised Image with NoiseLevel=0.01FilterSize

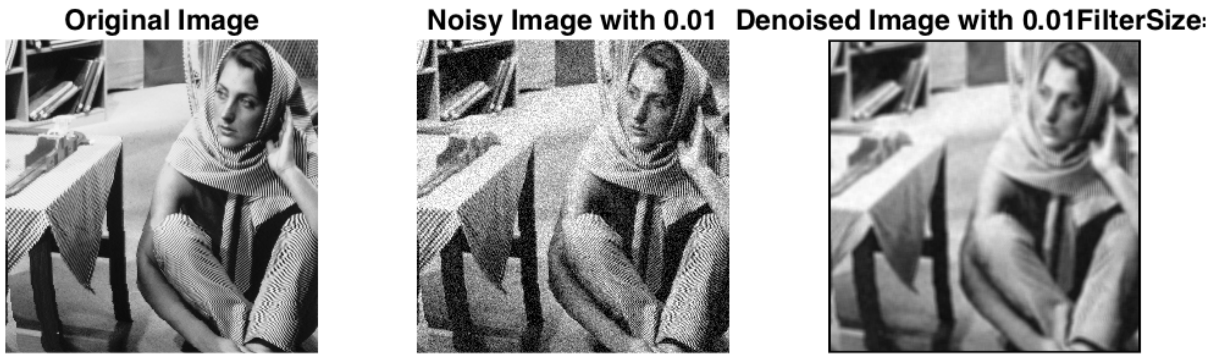
Window size 5


Original Image
Noisy Image with NoiseLevel=0.01
Denoised Image with NoiseLevel=0.01FilterSize

Window size 7



Window size 9

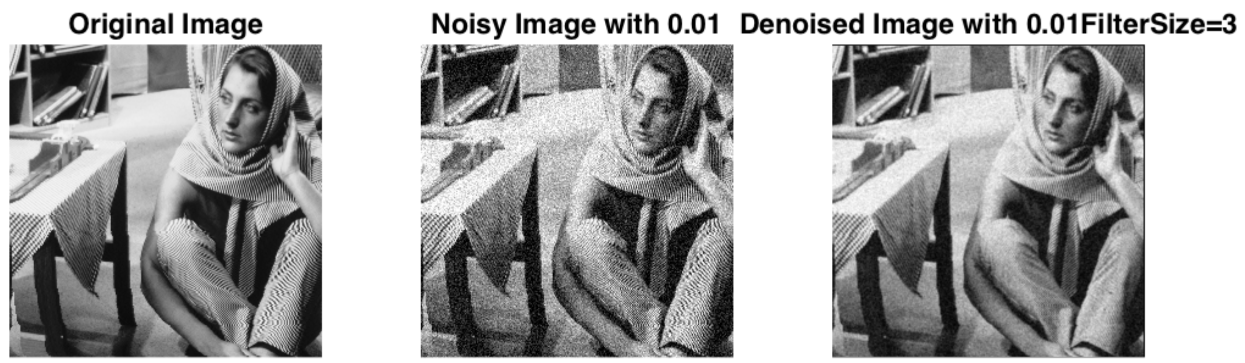


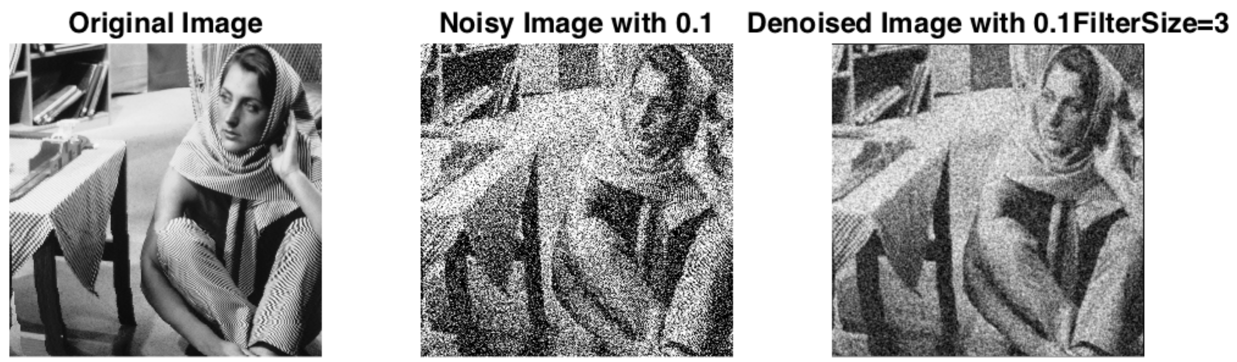
The best trade-off between noise removal and blurring is obtained from 3*3 window size filter for a chosen variance of 0.01 noise added. When 9*9 is used, the image becomes too blur although the noise is removed properly.

For different noise level,

```
clc;clear all;close all;
inImg = imread('barbara_gray.bmp');

%convert the inImg to int or single or double before proceeding!
grayImg=double(inImg);

noiseLevel1=0.01;
noisy_img1 = imnoise(inImg, 'gaussian', 0, noiseLevel1);
noiseLevel2=0.1;
noisy_img2 = imnoise(inImg, 'gaussian', 0, noiseLevel2);
```

```
%imnoise does not work when inImg is not uint8!
noisy_img1=single(noisy_img1);
noisy_img2=single(noisy_img2);

%%
%set filter
filterSize=input('Enter the filter size');

% filterSize=3;
denoising_filter=ones(filterSize,filterSize)/(filterSize*filterSize);

tmpImg1 = my_conv2(noisy_img1, denoising_filter);
tmpImg2 = my_conv2(noisy_img2, denoising_filter);
% III) Normalize and convert the image to uint8
denoisedImg1 = 255*my_mat2gray(tmpImg1);
denoisedImg2 = 255*my_mat2gray(tmpImg2);
%% iv) display the original and the filtered images.

cf=figure(1);
subplot(2,3,1);
imshow(inImg);
title('Original Image');
subplot(2,3,2);
imshow(noisy_img1,[]);
title(['Noisy Image with ' num2str(noiseLevel1)]);
%note you need to use [] option to see the full range of noisy image
subplot(2,3,3);
imshow(denoisedImg1,[]);
title(['Denoised Image with ' num2str(noiseLevel1) 'FilterSize=' num2str(filterSize)]);
subplot(2,3,4);
imshow(inImg);
title('Original Image');
subplot(2,3,5);
imshow(noisy_img2,[]);
title(['Noisy Image with ' num2str(noiseLevel2)]);
%note you need to use [] option to see the full range of noisy image
subplot(2,3,6);
imshow(denoisedImg2,[]);
title(['Denoised Image with ' num2str(noiseLevel2) 'FilterSize=' num2str(filterSize)]);

% v) save the filtered images into another file.
print(cf, 'HW2_Q2_diffnoise', '-dtiff');
```

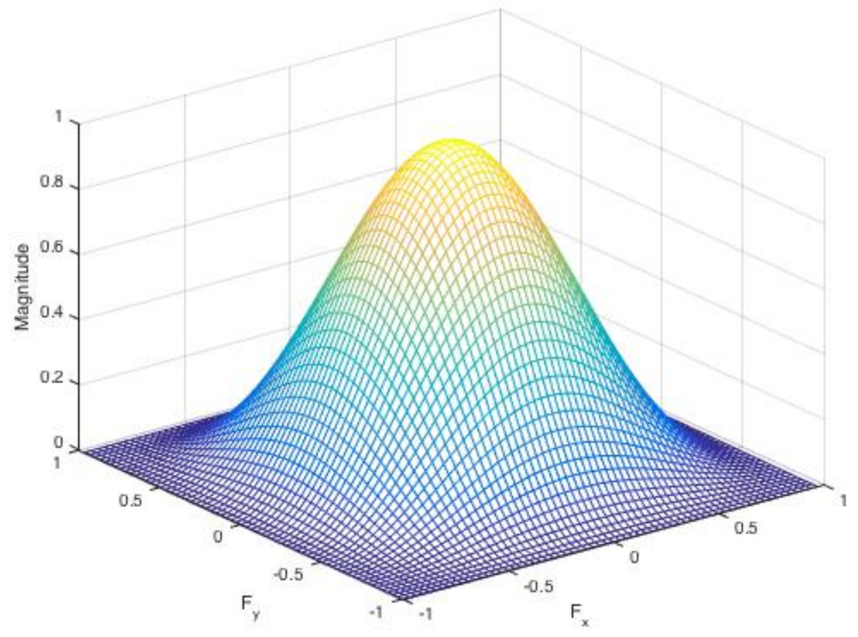




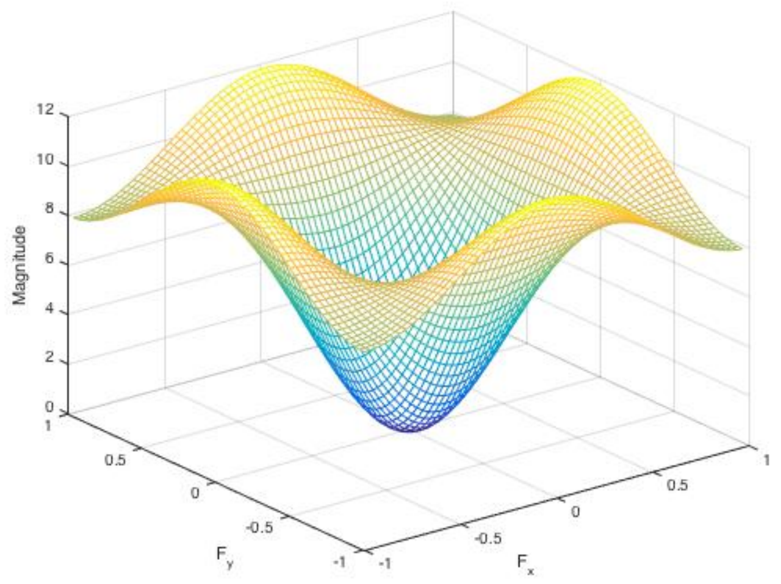
When variance of Gaussian noise is more, we lose more information. Even after denoising, we cannot completely regain the information.

Question 5 (written assignment)

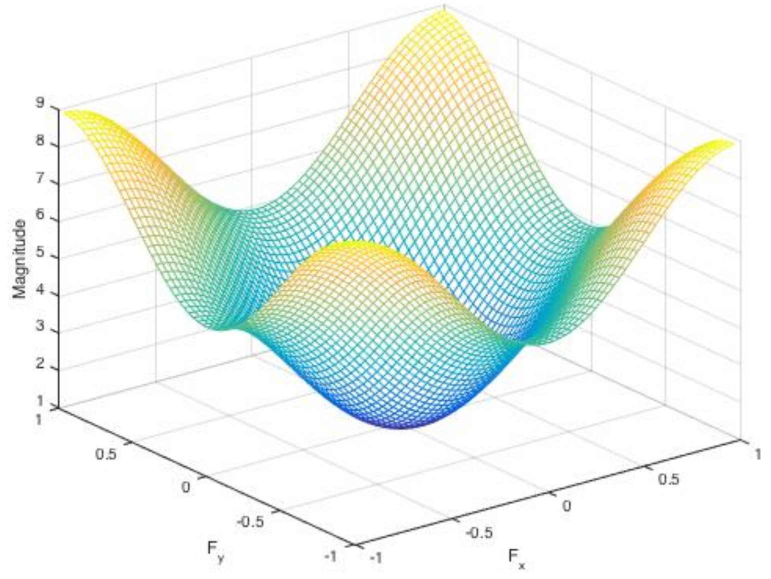
Frequency response of H1 filter



Frequency Response of H2 filter



Frequency Response of H3 filter



H1 filter acts as low pass filter.
H2 filter is used to detect edges.
H3 acts as low contrast filter