CSC249 PROJECT1 REPORT

Yiwei Han

yhan32@u.rochester.edu

1. INTRODUCTION

Image filtering processes apply mathematical or kernel to an image to enhance or modify its features. The main purpose of image filtering is to extract or enhance specific image features such as edges, noise, and blur by removing or reducing unwanted information. In practice, image filtering works by applying a filter or kernel to each pixel of the image. A filter is typically a matrix of values that is convolved with the pixel values of an image to create a new image with modified features. Image Filtering can be used for a variety of image-processing tasks, for example, image denoising, edge detection, image segmentation, feature extraction, and image enhancement. The choice of filters and their parameters depends on the specific task and input image characteristics. Overall, image filtering is an essential process in image processing that allows us to extract or enhance certain features of an image and remove unwanted information. It plays an important role in many image processing applications such as computer vision, computer graphics, etc.

In this project, methods of filtering include Sobel, high pass, low pass, median filter, and Fourier transform. Gaussian filter is a low-pass filter that is used to reduce high-frequency noise in an image. A High-pass filter is a sharpening filter that is used to enhance an image's edges and details. The median filter is a non-linear filter that is used to remove impulse noise in an image. A Sobel edge filter is an edge detection filter that is used to highlight edges in an image. Fourier transform is a frequency-domain filter that is used to enhance or remove certain frequencies in an image. I will use an image that I took as an example to perform the above filters and generate results.

2. METHODS

2.1 Methods for Question1

In the first question, I used the MATLAB command *imread()* to read the paired image with different lighting in Homework 1. Then I used the MATLAB command *imshow()* and *images()* to display both images.

2.2 Methods for Question2

2.2.1 Part a

To read and display an image in MATLAB, the method is the same as the method I used in Question 1. I used the *imread()* followed by *imshow()* to display both images.

2.2.2 Part b

First, I converted the image to grayscale. Then I used *fspecial('gaussian',[3 3],10)* to obtain the filter kernel for the gaussian low pass filter. After that, I used *filter2()* to apply the filter to the images. I used *freqz2()* to display the frequency response of the gaussian filter. Then, I used *[isseparable,hcol,hrow] = isfilterseparable(gausFilter)* to determine whether the gaussian filter is separable. Because the inseparable returns 1, it means the filter is separable, and the hcol and hrow determines the horizontal and vertical component of the filter. Then I used *filter2(hcol, filter2(hrow, I))* command to apply the horizontal and vertical component to the image respectively to perform the separated component of gaussian filter image.

2.2.3 Part c

The image is first converted to grayscale. Then the Fourier transform of the image is computed and shifted to the center. The magnitude of the Fourier transform is used to compute the center coordinates. Next, a Gaussian high pass filter is computed for each pixel in the Fourier transform. The distance of each pixel from the center of the Fourier transform is calculated using the Pythagorean theorem. This distance is used to compute the high pass filter coefficients for that pixel using a Gaussian function with a cutoff frequency of d0. Multiply the high pass filter coefficients by the Fourier transform of the original pixel values get the filtered Fourier transform. Pixels at the edge of the image are treated in the same way as pixels in the rest of the image. Finally, the inverse Fourier transform of the filtered Fourier transform is computed and put back in place and get the high pass filtered image.

A Gaussian highpass filter is defined by H(u,v) = 1 - exp(-(D(u,v)^2)/(2*D0^2)) H(u,v) is filter coefficient at position (u,v), D(u, v) is the distance between (u,v) and the center and D0 is the cutoff frequency. Initialize the D0 and n values first, then create a grid of x and y values using a mesh grid. Then iterate through all the elements of the x and y matrices, calculate the distance D between the current position and the center, and use the formula to calculate the filter coefficients and store them in Z. Finally, the frequency response is displayed on the mesh function.

2.2.4 Part d

To process the image using a 3x3 Median Filter, we need to first convert the image to gray scale, then use the *medfilt2()* command in MATLAB, then use the *imfilter()* command to apply the filter.

2.2.5 Part e

To process the image using Sobel edge filter, we need to first convert the image to gray scale, then use the *fspecial('sobel')* command in MATLAB. Because this command only provide the horizontal matrix kernel, we take the transpose of it to make the vertical one. Then I took the sort of the sum of the two matrix square. Then use the *imfilter()* command to apply the filter.

2.2.6 Part f

Adding Gaussian noise by using the *imnoise()* command can be performed by *imnoise(img, 'gaussian')* at the first step.

2.2.7 Part g

Adding impulse noise by using the *imnoise()* command can be performed by *imnoise(img, 'salt & pepper', 0.01)* at the first step. We have set the noise density to 0.1, means that 10% of the pixels in the image will be affected by the noise.

2.3 Methods for Question3

2.3.1 Part a

First, I used rgb2gray to convert the image to grayscale, then I use the fft2() command to do the Fourier transformation.

2.3.2 Part b

I used the imagesc() command following the abs() to display the magnitude of the 2-D FFT. I used imagesc() because it can scale the value to fit the colormap. I used caxis() to adjust the color axis limits of the image. I set the limits to 1e-3 times the original maximum value, reduces the colors displayed in the image.

2.3.3 Part c

I used the fftshift() and the imagesc() again to display the magnitude of the 2-D FFT again.

2.3.4 Part d

I simply added the log() function before the original 2-D FFT when I used the imagesc() to display. The small offset I chose to be 1, and it is added with the original 2-D FFT.

imagesc(log((1)+abs(F)))

3. RESULTS and DISCUSSION

3.1 Question 1

Below are the results of display of both the bright and light image that I took for homework 1 using the imshow() and imagesc() command. We can see from the result window from the matlab, there is no essential difference that only the imagesc() has the label for the horizontal and vertical axis. The differences between the two commands inherently are described in the following section.

What is the difference between these two commands?

imshow() is primarily used to display images in full-size aspect ratio. By default, *imshow()* maps pixel values to a grayscale color table when displaying a grayscale image, but you can specify another color table. When displaying a color image, *imshow()* displays the image using the specified color table or the default color table.

imagesc() is primarily used to display images as scaled colormaps. *imagesc()* scales the data in the matrix to fit the area of the colormap and displays the result as an image. Each element specifies the color of a pixel in the image. The resulting image is an m x n pixel grid. where m is the number of rows and n is the number of columns. *imagesc()* can automatically adjust scaling to use the full range of the color table, which is useful for displaying images whose range of pixel values is much larger than used for display.

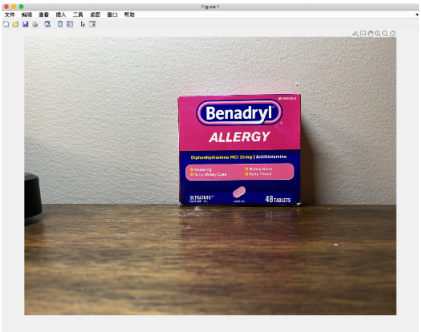


Fig1. Bright Image Display Using imshow()

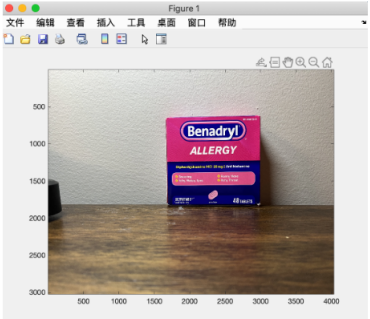


Fig2. Bright Image Display Using imagesc()

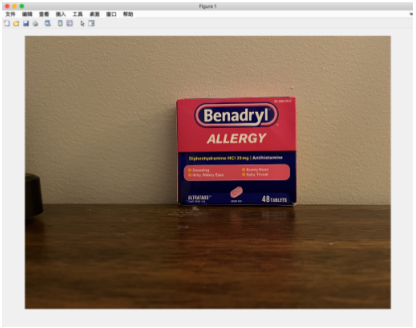


Fig 3. Low Image Display Using imshow()

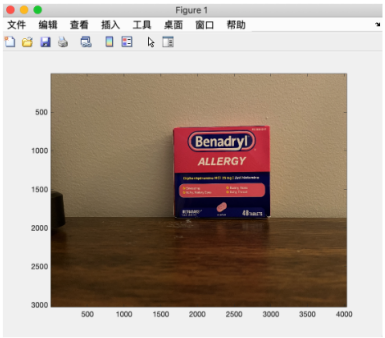


Fig4. Low Image Display Using imagesc()

3.2 Question 2

3.2.1 Part a

The results for this part is the same as the result for Question 1.

3.2.2 Part b

A Gaussian lowpass filter used to attenuate high frequency noise and other unwanted features in the image while preserving low frequencies. The filter convolves the image with a normally distributed Gaussian kernel. Gaussian filter works by assigning a weight to each pixel in the image based on its distance from the center of the kernel. Pixels closer to the center of the kernel have higher weights.

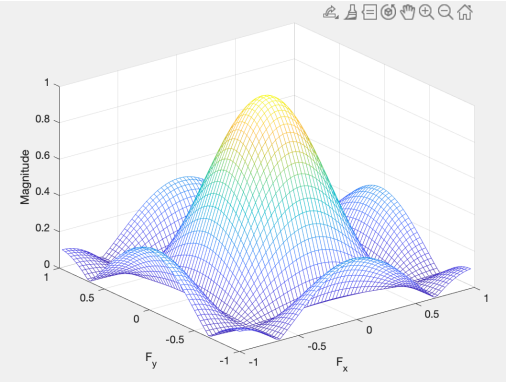


Fig . Frequency Response of Gaussian Filter

Is this filter separable? If yes, determine its 1-D components and apply them along the rows and columns, respectively. How does this result compare with the result of filter2?

Yes, the Gaussian low pass filter is separable. It is composed of two 1-D array, one in the horizontal direction and one in the vertical direction and the whole filter is the product of the two.

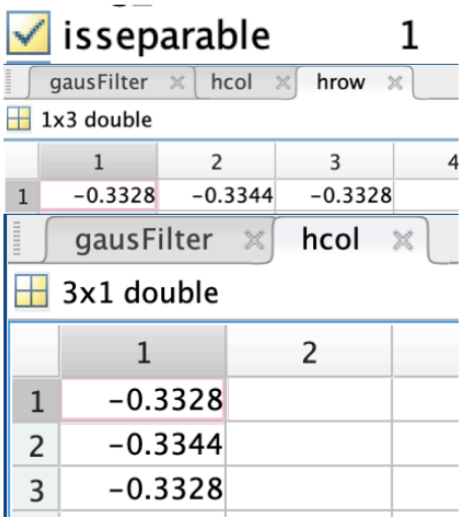


Fig Show whether the Gaussian filter is separable and its horizontal and vertical component

The results between directly using the gaussian filter and apply the 1-D component separately should be the same. The difference is the that if we apply the 1-D component separately, it is more efficient in time and require less operations. So if the processing amount is very large, it is better to apply the 1-D array separately on the image to do the filtering.

Fig1. Original Bright Image After Gaussian Filter

Fig1. Gaussian Noise Bright Image VS Gaussian Noise

Fig1. Impulse Noise Bright Image VS Gaussian Noise

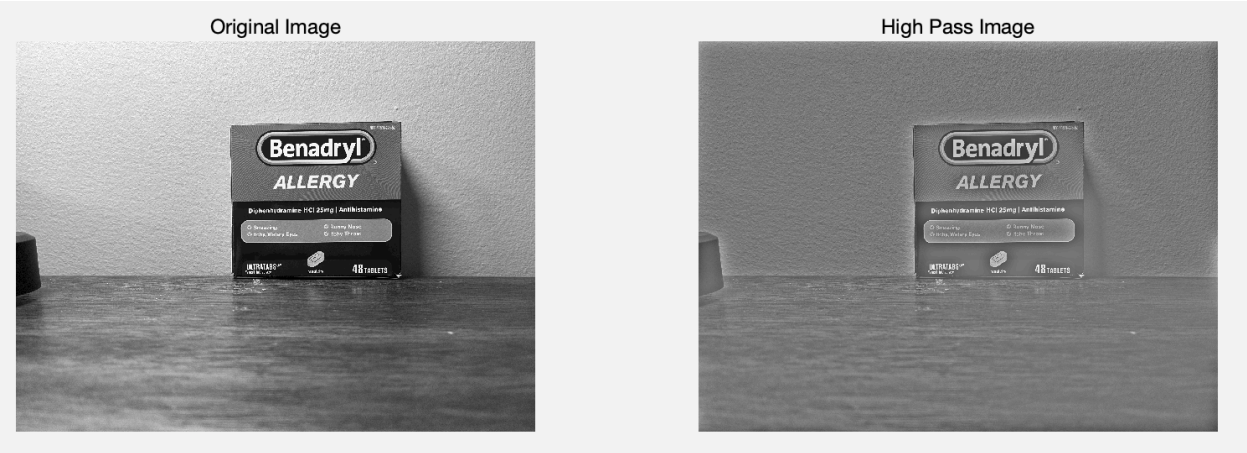
Original low Image VS Gaussian Filter

Gaussian Noise Low Image VS Gaussian Filter

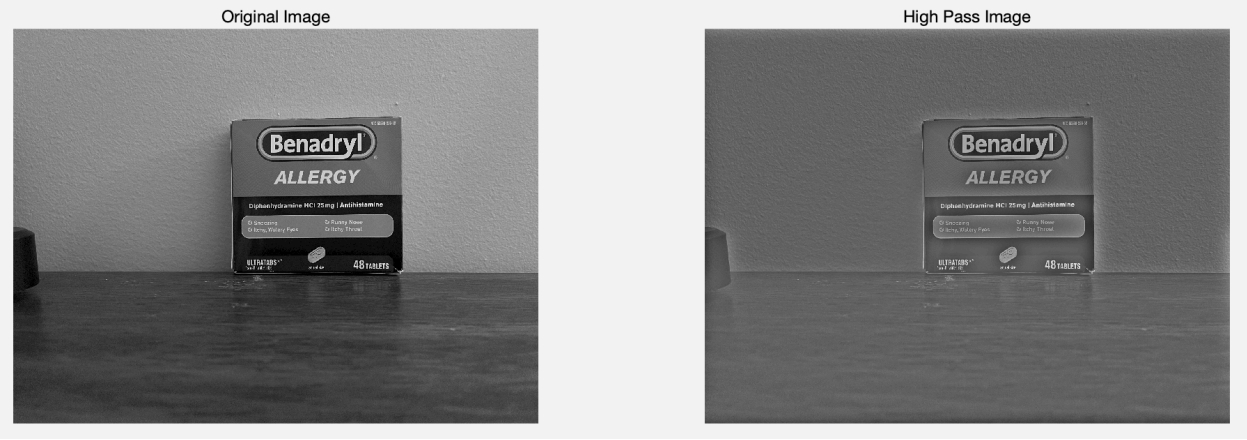
Impulse Noise Low Image VS Gaussian Filter

3.2.3 Part C

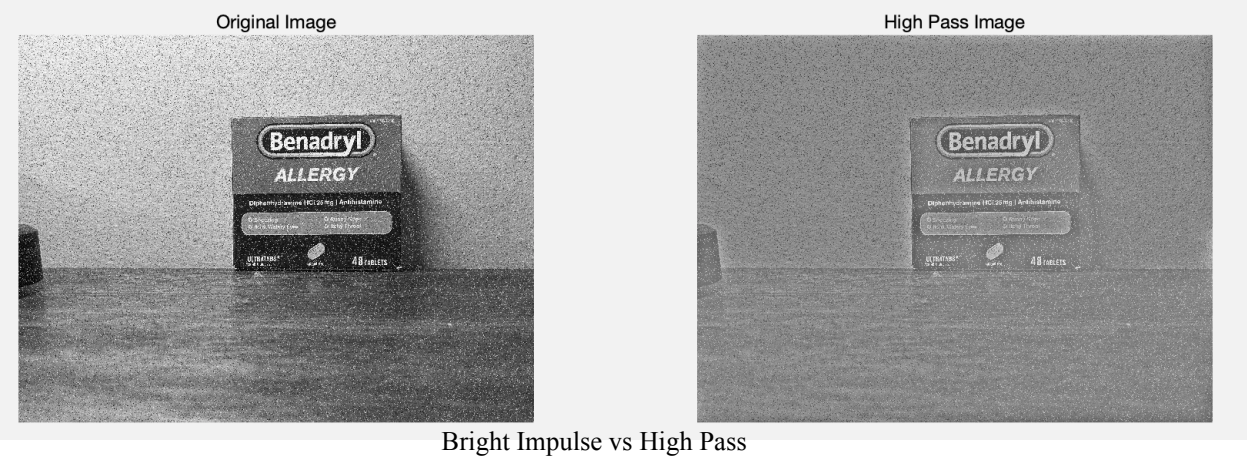
High pass filter passes high frequency components of an image and attenuates low frequency components. High pass filters are commonly used for tasks such as: High pass filters are often applied to images using convolution. Convolution slides the filter kernel over the image and computes the filter response at each pixel location. The resulting filtered image has low frequency content reduced and edges and details enhanced.



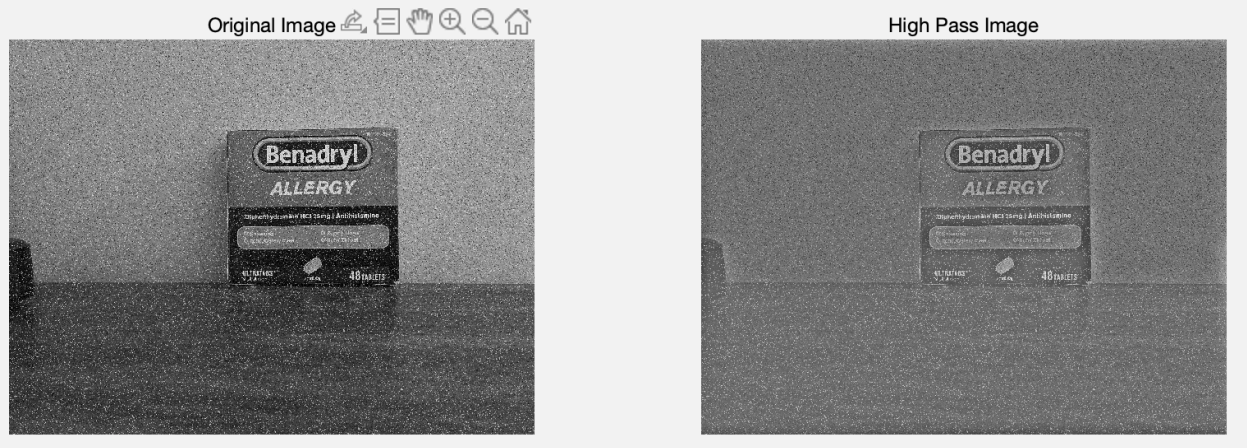
Original Bright vs High Pass



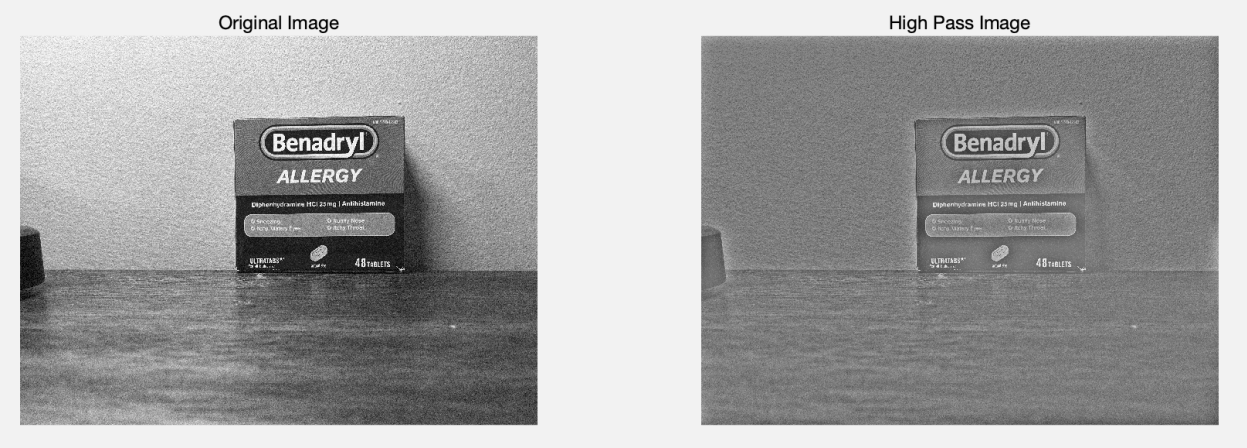
Original Low vs High Pass



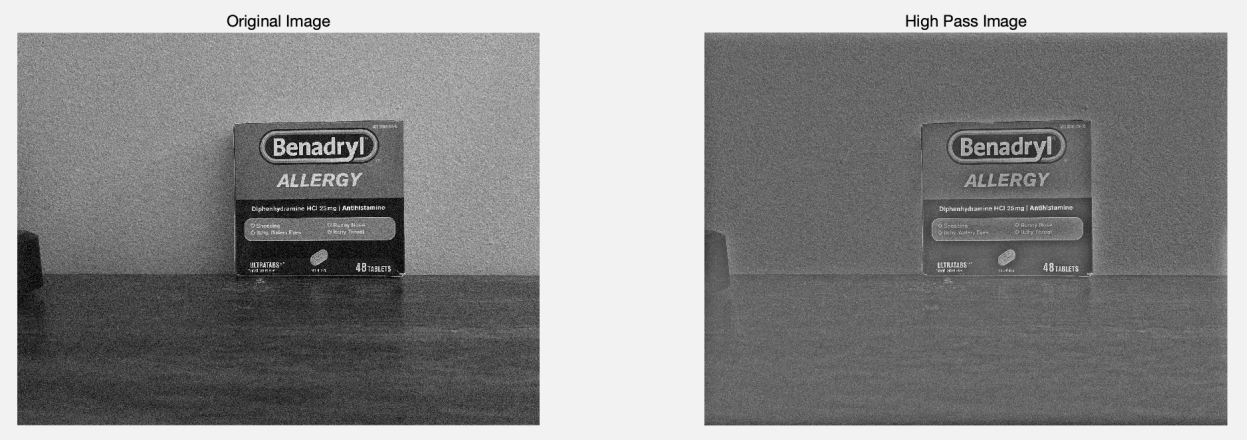
Bright Impulse vs High Pass



Low Impulse vs High Pass

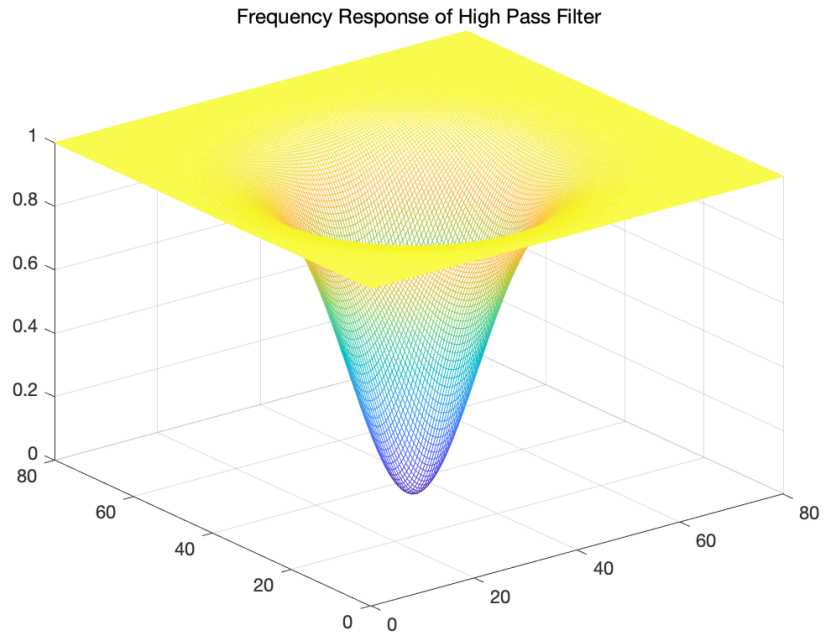


Bright Gauss vs High Pass



Low Gauss vs High Pass

High pass filter has a bell-shaped curve with a peak amplitude corresponding to the highest frequency passed by the filter.



3.2.4 Part d

A median filter is a filter used to reduce noise in image processing. This is a nonlinear filter that replaces each pixel value in an image with the median value of its neighboring pixels. Median filtering effectively removes image noise without significantly affecting image edge sharpness and other features. Belows are before and after apply the median filter to original and noise pictures.

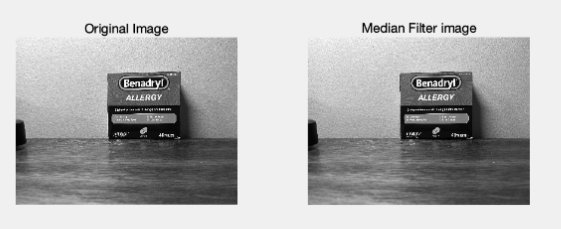


Fig1. Original Bright Image VS Median Filter

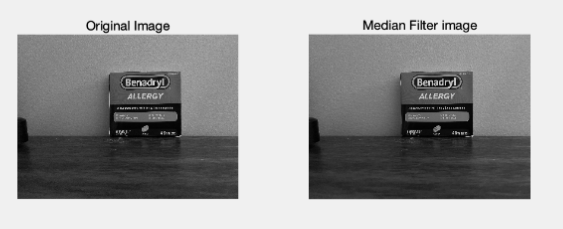


Fig1. Original Low Image VS Median Filter

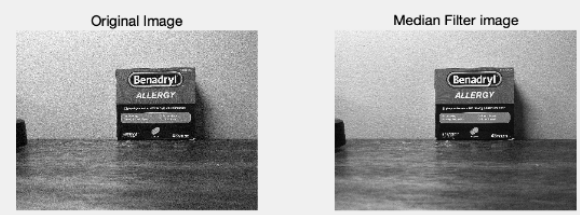


Fig1. Gaussian Bright Image VS Median Filter

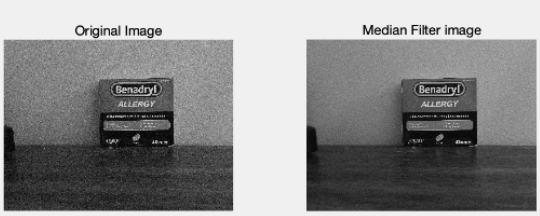


Fig1. Gaussian low Image VS Median Filter

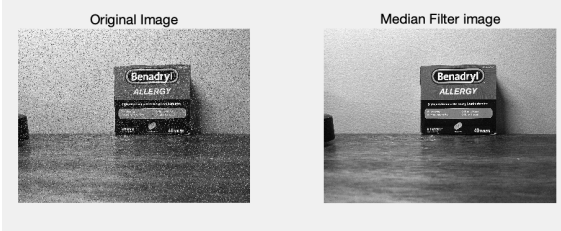


Fig 3. Impulse Bright Image VS median filter

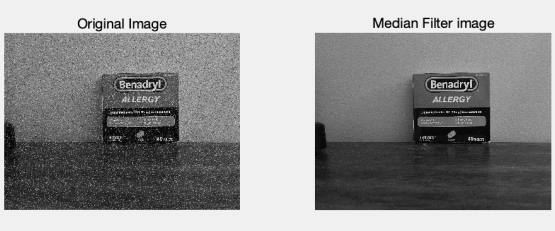


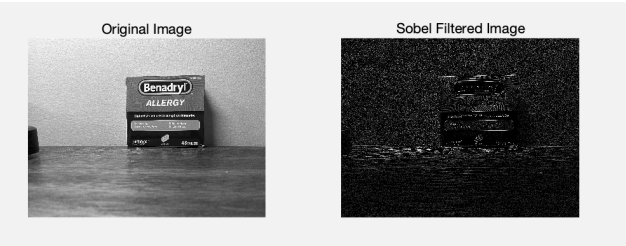
Fig 3. Impulse Low Image VS median filter

Comment on the frequency response of the median filter.

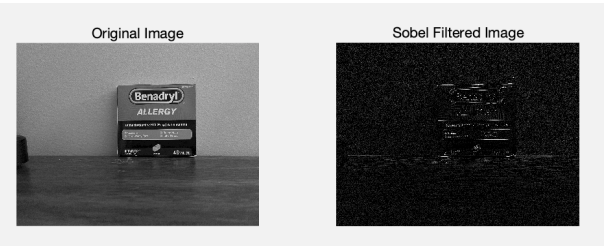
The frequency response of the median filter is a nonlinear filter and cannot be characterized by frequency response. The median filter is often used as a smoothing filter to remove high frequency noise. Specifically, the median filter replaces each pixel value with the median value of the pixel's neighborhood. This tends to make the transitions between adjacent pixels smoother and suppress high frequency features. As a result, the median filter can be thought of as a lowpass filter that rolls off high frequencies. The exact frequency response of the median filter depends on the size of the filter window and certain properties of the input image. In general, the larger the filter window, the stronger the lowpass effect and the more effective attenuating high frequency content.

3.2.5 Part e

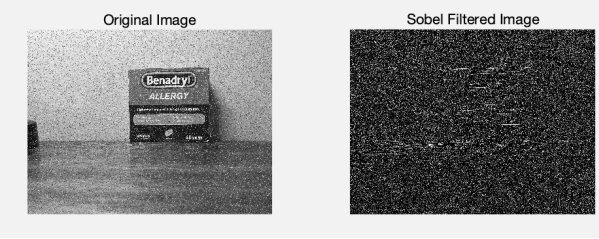
The fspecial('sobel') function creates a filter that approximates the image intensity gradient at each pixel. Sobel is a filter that finds edges by calculating gradients of image intensities. Because fspecial(‘sobel’) only has the matrix for horizontal operation, we take the transpose of it to form the vertical one. The Sobel filter now has two matrix, one for computing the horizontal gradient and one for computing the vertical gradient. The horizontal core detects pixel intensity changes from left to right, and the vertical core detects pixel intensity changes from top to bottom. The Sobel filter combines the results of both kernels to produce an approximation of the image gradient. The results of the images after sobel filter is displayed below.



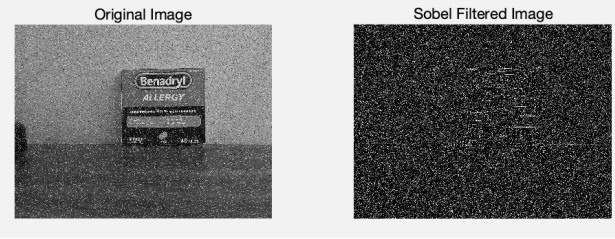
Original Bright Image vs Sobel Filter



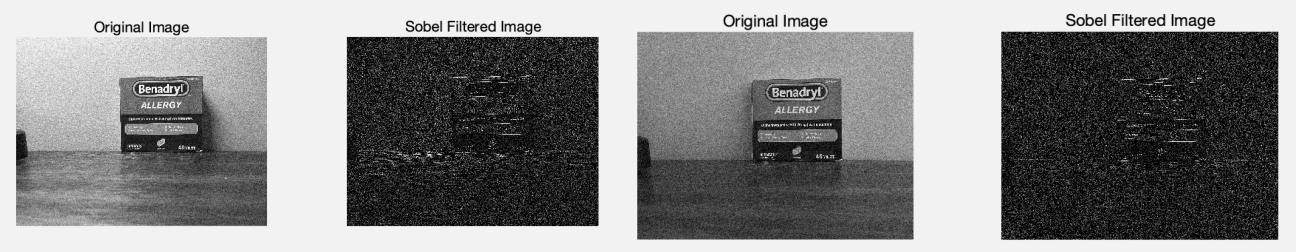
Original Low Image vs Sobel Filter



Impulse Bright Image vs Sobel Filter



Impulse low Image vs Sobel Filter



Gaussian Bright Image vs Sobel Filter

Gaussian Low Image vs Sobel Filter

3.2.6 Part f

Below is the comparison of the original image versus the image adding the gaussian noise. In an image, Gaussian noise appears as random variations in pixel values that are distributed according to a Gaussian probability distribution. Gaussian noise can cause blurring, loss of detail, and reduction in contrast in the image.

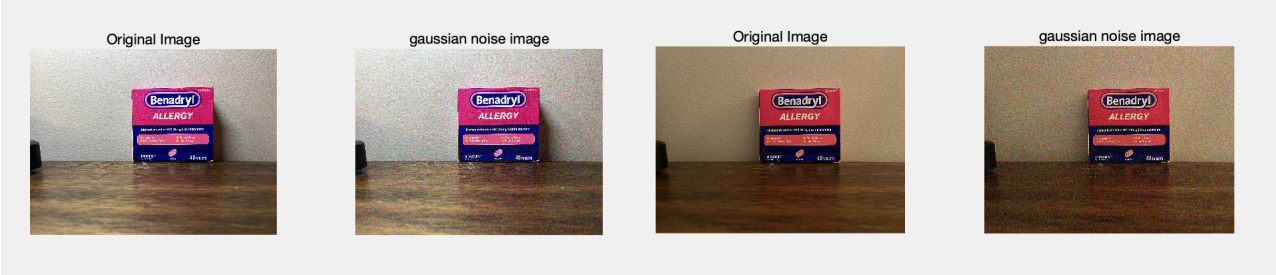


Fig1. Original Bright Image VS Gaussian Noise

Fig1. Original Low Image VS Gaussian Noise

3.2.7 Part g

Below is the comparison of the original image versus the image adding the impulse noise. Impulse noise appears as randomly distributed white and black pixels in an image.

Fig1. Original Bright Image VS Impulse Noise

Fig1. Original Low Image VS Impulse Noise

3.2.8 Part h

Comment on the results obtained in steps b) through g).

From our output display above, we can see that all filter has its own strength and weakness. Firstly, low pass filter is good at smooth images, reduce noise, and blur. We can see that they can remove noises after the gaussian noise and impulse noise is added to the image.

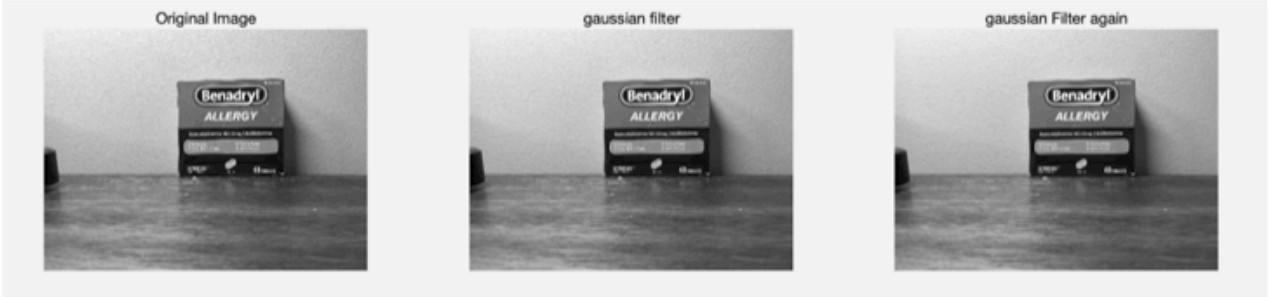
Secondly, high pass filter is good at sharpen images, enhance fine details, and detect edges, as opposed to gaussian low pass filter.

Thirdly, from the output of the median filter to original image, gaussian noise image and impulse noise image, we can see that the median filter is especially good for removing impulse noise images.

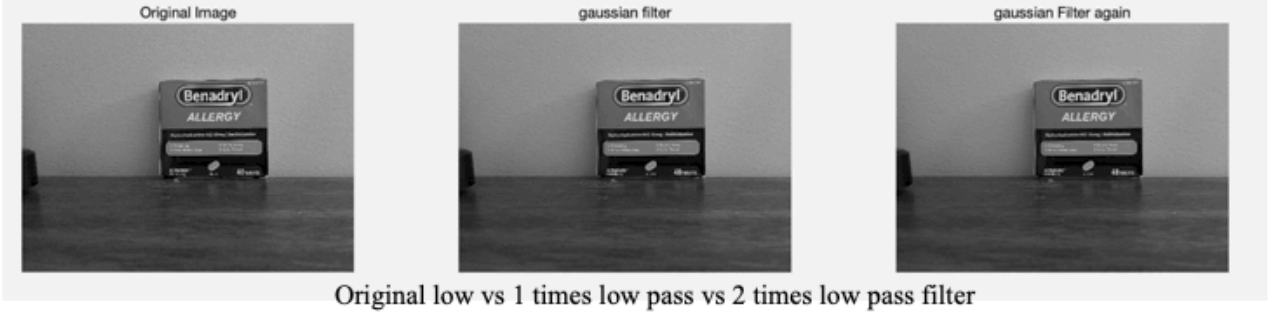
Lastly, Sobel filters are effective at detecting edges, and it is pretty sensitive to noise, and comparing impulse noise and gaussian noise, we can see that sobel filter is more sensitive to impulse noise because gaussian noise tend to be more uniform that does less effect on the output of sobel filter than impulse noise.

How does a repeated application of the Low Pass filter in b) affect the results?

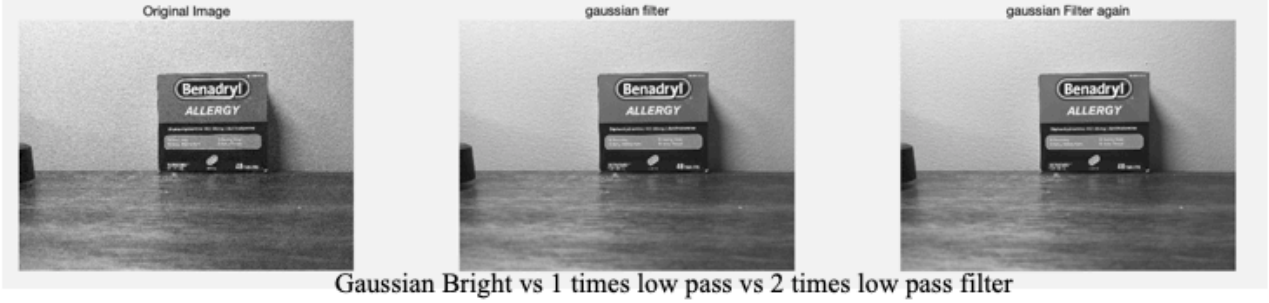
Repeatedly applying a Gaussian lowpass filter to an image produces a repeated blurring effect on the image. This is analogous to reapplying a lowpass filter to an already slightly blurred image. The more times you apply the filter, the blurrier the image will be as the filter removes high frequency components. You can also simulate the effect of a larger smoothing kernel by repeating the Gaussian lowpass filter again. However, depending on the image used, repeated application of the Gaussian lowpass filter may result in loss of image detail. Below are the results of reaped application of the low pass filter in (b).



Original Bright vs 1 times low pass vs 2 times low pass filter

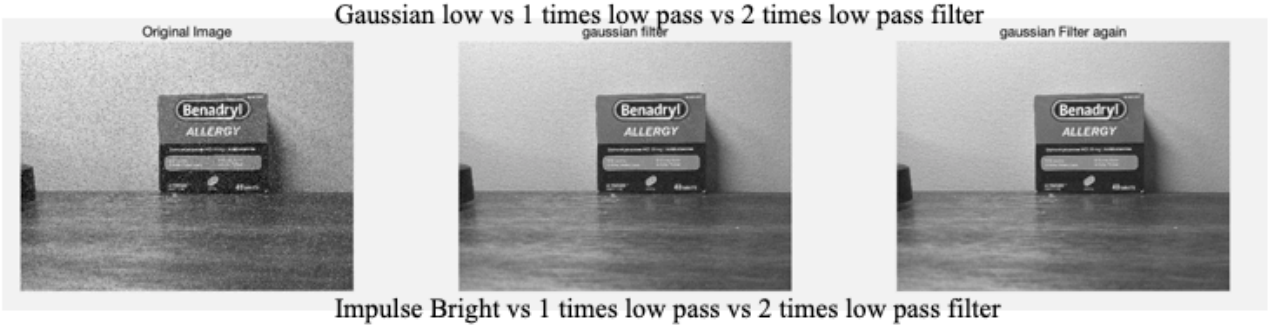


Original low vs 1 times low pass vs 2 times low pass filter

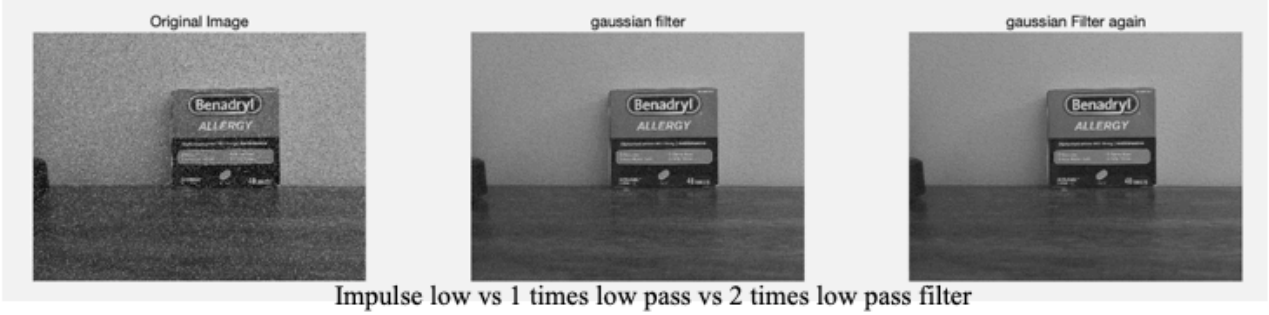


Gaussian Bright vs 1 times low pass vs 2 times low pass filter

Gaussian low vs 1 times low pass vs 2 times low pass filter



Impulse Bright vs 1 times low pass vs 2 times low pass filter



Impulse low vs 1 times low pass vs 2 times low pass filter

Is the Sobel edge filter immune to noise?

No, the Sobel edge filter is not completely immune to noise, but it does reduce it to some extent. The Sobel edge filter can be sensitive to high frequency noise in the image. Noise can cause fluctuations in intensity values in an image. The Sobel filter can falsely detect edges that are masked by noise when there are actually no edges. Techniques such as smoothing or blurring the image before applying the Sobel filter can be used to reduce the effects of noise so the filter will perform better.

Propose a method for reducing the effects of noise when performing High Pass filtering or detecting edges.

When performing high-pass filtering or edge detection, noise can significantly affect the result by introducing spurious edges and reducing accuracy. One method that you can use is to apply a low-pass filter, such as a Gaussian filter, to smooth the image before applying a high-pass filter or edge detection. This allows us to reduce high frequency noise in the image while preserving other features. The level of smoothing should be chosen carefully to balance noise reduction and edge preservation.

3.3 Question 3

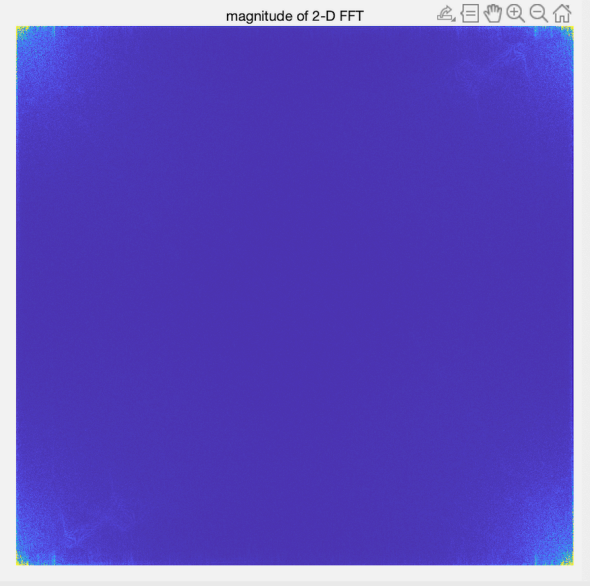
3.3.1 Part a

The fft2 command computes the 2-D Fast Fourier Transform of an image. The 2-D FFT transforms the spatial domain representation of an image into the frequency domain.
The result is displayed in part b below.

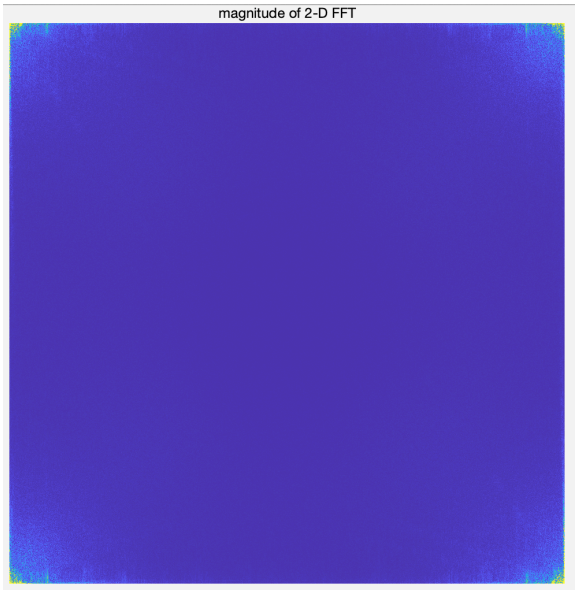
3.3.2 Part b

Where is the origin located?

The origin is the zero-frequency center, and is located at the vertices of the FFT2 image, where the highest magnitude values are displayed.



Bright FFT2

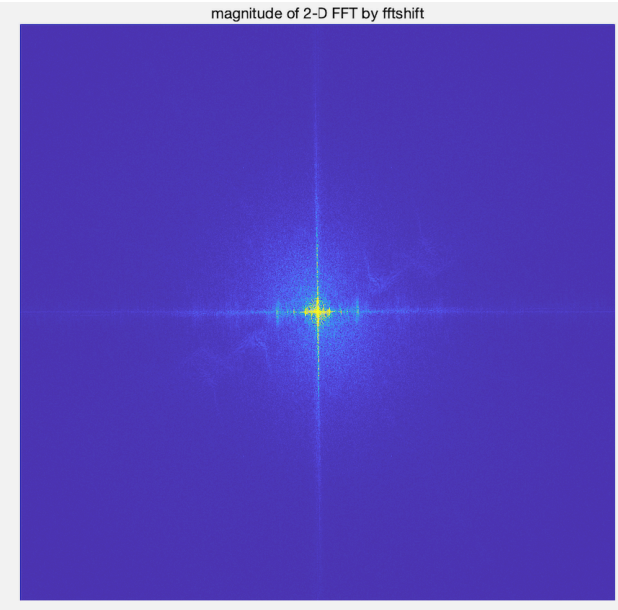


Low FFT2

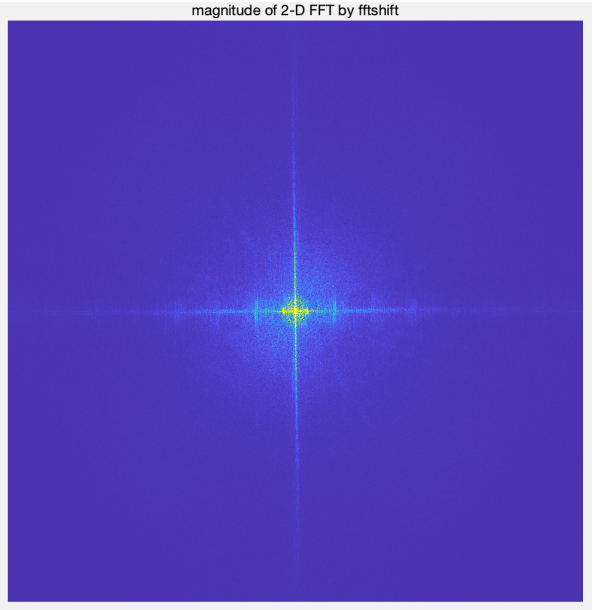
3.3.3 Part c

What is the effect of this command on the location of the origin?

fftshift command shift zero-frequency component to center of spectrum, which results in the low frequencies being displayed at the center and the high frequencies being displayed at the corners.



Bright fftshift

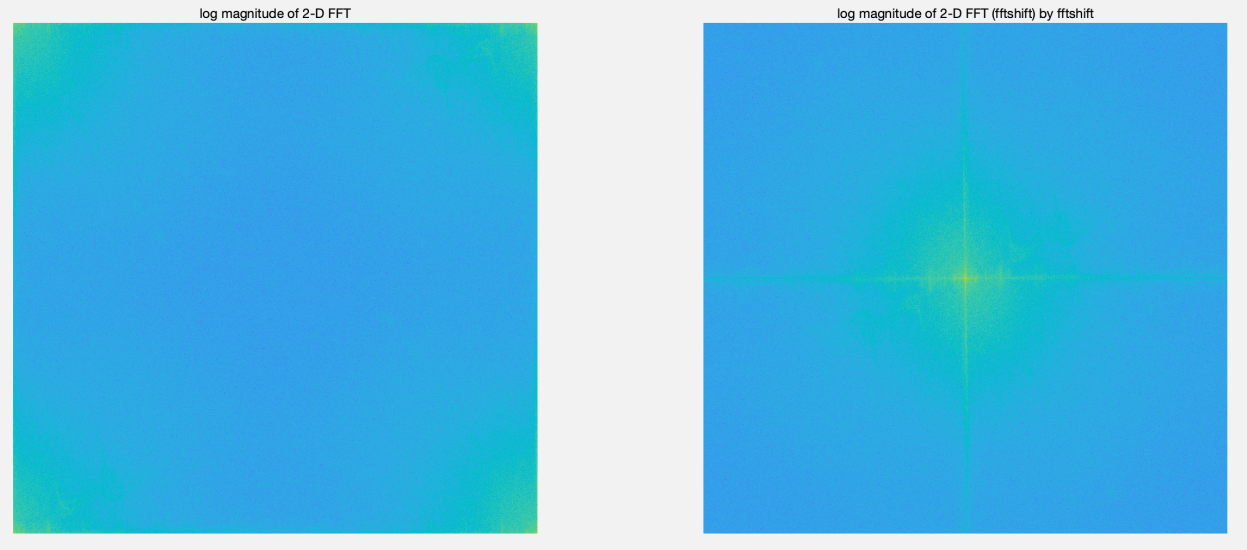


Low fftshift

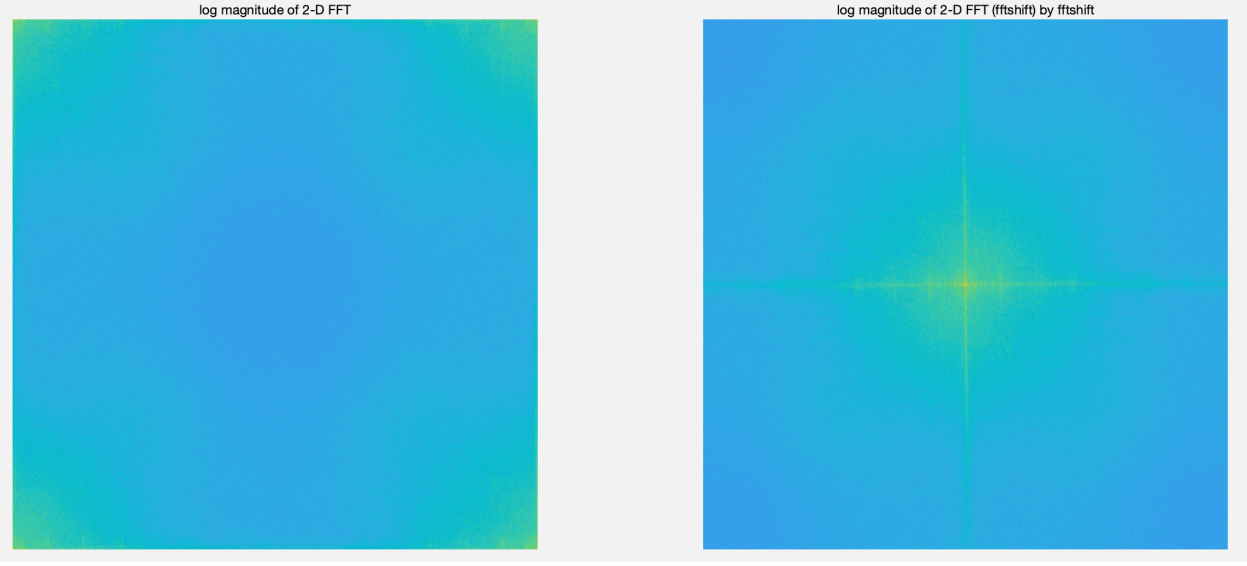
3.3.4 Part d

Which method of display do you prefer (with or without log) and why?

I prefer the log method because the original image has a large range after FFT2. Compresses the dynamic range of values and improves the visibility of image details. Additionally, the logarithmic scale makes it easier to compare the magnitudes of different frequency components.



Bright Log Transform



Low Log Transform

In Step 3, describe the effect of lighting change, linear or nonlinear, on the Fourier transform (magnitude)

From our result in the magnitude by the fft2(), fftshift() and the ones after the logarithm, there is not too much difference, except that the plots by the fftshift() for the bright image experiences minor pattern along the diagonal of the 1st and 3nd quadrant, so the lighting change might be non-linear. Moreover, if the lighting change is linear, one plot can be generated by rotating of the other plot with different lighting, as a result, the linear change between the two images are non linear. After that, I wanted to be more accurate so I computed the correlation of two images' log of the FFT magnitude. If the correlation is 1, then the lighting change is linear, otherwise, it is non-linear. Because the correlation is not 1, we can be more accurate sure that the lighting change is non-linear for the two images.

```
corrcoef_ps =

    0.9845

The lighting change is nonlinear
```

4. APPENDIX

4.1 Question 1

4.1.1 Part a

```
img = imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
```

4.1.2 Part b

Do not need code.

4.1.3 Part c

```
figure
imshow(img)
figure
imagesc(img)
```

4.2 Question 2

4.2.1 Part a

Same as Question 1 Part a and Part c.

4.2.2 Part b

```
I=imread('/Users/yiweihan/Desktop/low_gauss.jpg');
I=rgb2gray(I);
figure
gausFilter = fspecial('gaussian',[3 3],10);
blur=filter2(gausFilter,I,'same')/255;
freqz2(gausFilter);
[isseparable,hcol,hrow] = isfilterseparable(gausFilter);
filtered_img_sep = filter2(hcol, filter2(hrow, I))/255;
figure;
subplot(1, 3, 1);
imshow(I);
title('Original Image');
subplot(1, 3, 2);
imshow(blur);
title('gaussian filter');
subplot(1, 3, 3);
imshow(filtered_img_sep);
title('Filtered Image separable');
```

4.2.3 Part c

```
close all;
clear all;
clc;
I = imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
I = rgb2gray(I);
s=fftshift(fft2(im2double(I)));
[a,b]=size(s);
d0=10;
a0=round(a/2);
b0=round(b/2);
for i=1:a
    for j=1:b
        distance=sqrt((i-a0)^2+(j-b0)^2);
        h=1-(exp(-(distance^2)/(2*(d0^2))));
        s(i,j)=h*s(i,j);
    end
end
s=real(ifft2(ifftshift(s)));
subplot(1, 2, 1);
imshow(I);
title('Original Image');
subplot(1, 2 ,2);
imshow(s,[]);
title('High Pass Image');
```

```
clc,clear,close all
warning off
feature jit off
D0 = 10;
n = 3;
x = 0:.5:80;
y = 0:.5:80;
[X,Y] = meshgrid(x,y);
for i=1:size(X,1)
    for j=1:size(X,2)
        D = sqrt( (X(i,j)-35).^2 + (Y(i,j) -35).^2  );
        Z(i,j)= 1- exp(-D.^2./2./D0./D0);
    end
end
figure('color',[1,1,1])
mesh(X,Y,Z)
title('Frequency Response of High Pass Filter');
```

4.2.4 Part d

```
I = imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
I=rgb2gray(I);
K = medfilt2(I, [3 3]);
figure;
subplot(1, 2, 1);
imshow(I);
title('Original Image');
subplot(1, 2, 2);
imshow(K);
title('Median Filtered Image');
```

4.2.5 Part e

```
clc;clear all;
img = imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
img = rgb2gray(img);
fs8 = fspecial('sobel');
J8=imfilter(img,fs8);
fs82 = transpose(fspecial('sobel'));
J82 = imfilter(J8,fs82);
figure;
subplot(1, 2, 1);
imshow(img);
title('Original Image');
subplot(1, 2, 2);
imshow(J82);
title('Sobel Edge Filter image');
```

4.2.6 Part f

```
I=imread('/Users/yiweihan/Desktop/IMG_2298.jpg');
I=imnoise(I,'gaussian');
imwrite(I,'/Users/yiweihan/Desktop/low_gauss.jpg')
```

4.2.7 Part g

```
I=imread('/Users/yiweihan/Desktop/IMG_2298.jpg');
I=imnoise(I,'salt & pepper',0.1);
imwrite(I,'/Users/yiweihan/Desktop/low_impulse.jpg')
```

4.2.8 Part h

```
I=imread('/Users/yiweihan/Desktop/low_gauss.jpg');
I=rgb2gray(I);
figure
gausFilter = fspecial('gaussian',[3 3],10);
blur=filter2(gausFilter,I,'same')/255;
blur2=filter2(gausFilter,blur,'same');
figure;
subplot(1, 3, 1);
imshow(I);
title('Original Image');
subplot(1, 3, 2);
imshow(blur);
title('gaussian filter');
subplot(1, 3, 3);
imshow(blur2);
title('gaussian Filter again');
```

4.3 Question 3

4.3.1 Part a ~ Part d

```
I=imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
I = rgb2gray(I);
% Part a Compute 2D FFT
F=fft2(I);
% Part b Display magnitude of 2-D FFT
figure;
subplot(1,2,1);
imagesc(abs(F))
[~,Clim]=caxis;
caxis([0,Clim*1e-3])
axis off
title('magnitude of 2-D FFT')
subplot(1,2,2);
% Part c Use fftshift
imagesc(abs(fftshift(F)))
caxis([0,Clim*1e-3])
title('magnitude of 2-D FFT by fftshift')
axis off
%Part d Use log and Repeat
figure;
subplot(1,2,1);
imagesc(log((1)+abs(F)))
axis off
title('log magnitude of 2-D FFT')
subplot(1,2,2);
imagesc(log((1)+abs(fftshift(F))))
title('log magnitude of 2-D FFT (fftshift) by fftshift')
axis off
```

4.3.2 Compute Whether Linear Lighting Change

```
img1 = imread('/Users/yiweihan/Desktop/IMG_2297.jpg');
img2 = imread('/Users/yiweihan/Desktop/IMG_2298.jpg');
img1_gray = rgb2gray(img1);
img2_gray = rgb2gray(img2);
ft_img1 = fft2(img1_gray);
ft_img2 = fft2(img2_gray);
ps_img1 = abs(ft_img1);
ps_img2 = abs(ft_img2);
n = log(1+fftshift(ps_img1));
m = log(1+fftshift(ps_img2));
a = fftshift(ps_img1);
b = fftshift(ps_img2);
corrcoef_psnolog = corr2(a, b)
if corrcoef_psnolog == 1
    disp('The lighting change is linear');
else
    disp('The lighting change is nonlinear');
end
corrcoef_pslog = corr2(n, m)
if corrcoef_pslog == 1
    disp('The lighting change is linear');
else
    disp('The lighting change is nonlinear');
end
```