Table of Contents


```
clc;
clear all;
close all;
```

4.1.1

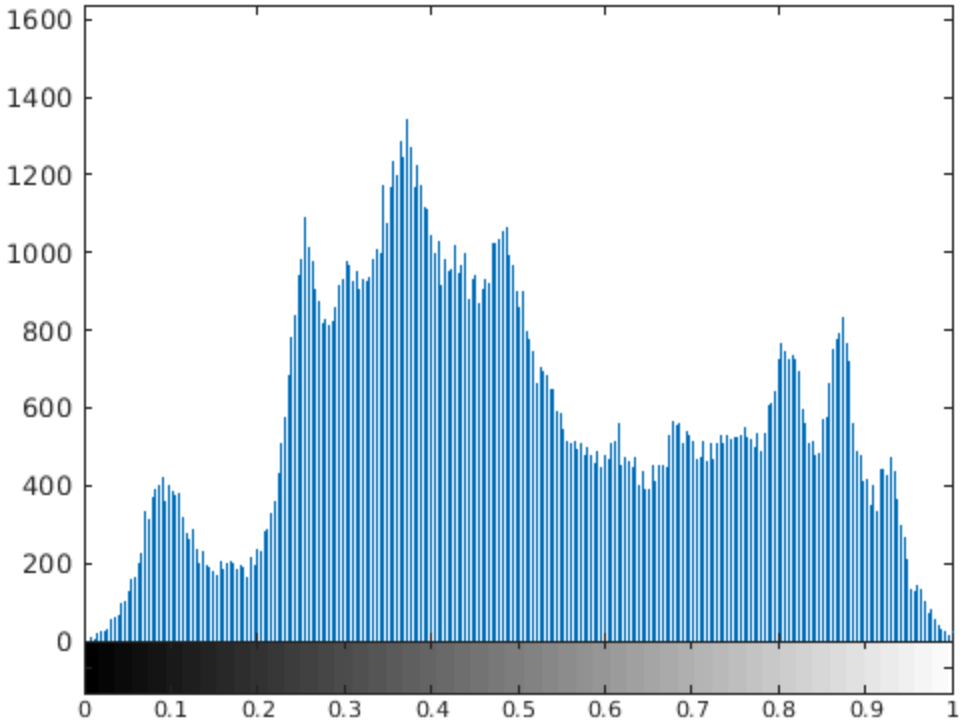
```
img = imread("./assets/lena.png");
imshow(img)
```

4.1.2

```
img = im2double(img);
```

4.1.3

```
figure('name', "Lenas Histogram")
imhist(img)
```



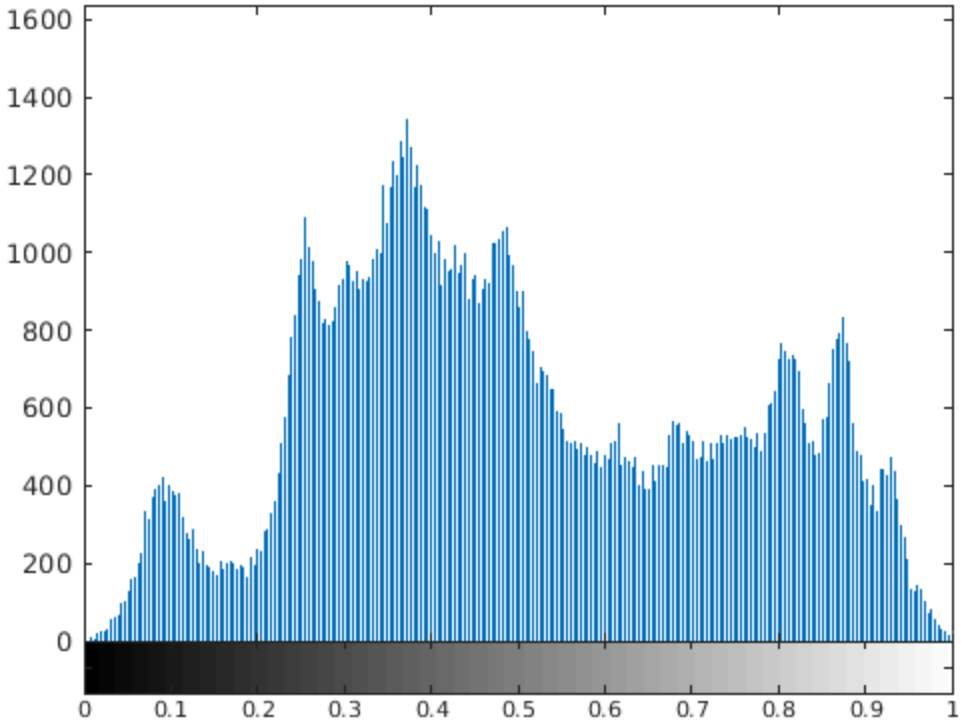
4.1.4

```
enhanced = histeq(img);
figure('name', "Enhanced Contrast");
subplot(1,2,1)
imshow(img(:,:,:));
title('Slice of Original Image');
subplot(1,2,2)
imshow(enhanced(:,:,:));
title('Slice of Enhanced Image');

figure('name', "montage pair");
imshowpair(img,enhanced,'montage');
```

Reasons for using Histogram equalization:

Enhancing Contrast: It improves the visibility of details in an image by stretching the intensity values across a broader range.

Preprocessing for Image Recognition: Histogram equalization can enhance features in images, making them more suitable for subsequent image processing tasks like object recognition.



Slice of Original Image

Slice of Enhanced Image

4.1.5

```
figure('name', "Enhanced Lena Histogram")
imhist(enhanced)
```

Factors that may prevent the resulting histogram from being perfectly uniform:

Limited Information in the Original Image:

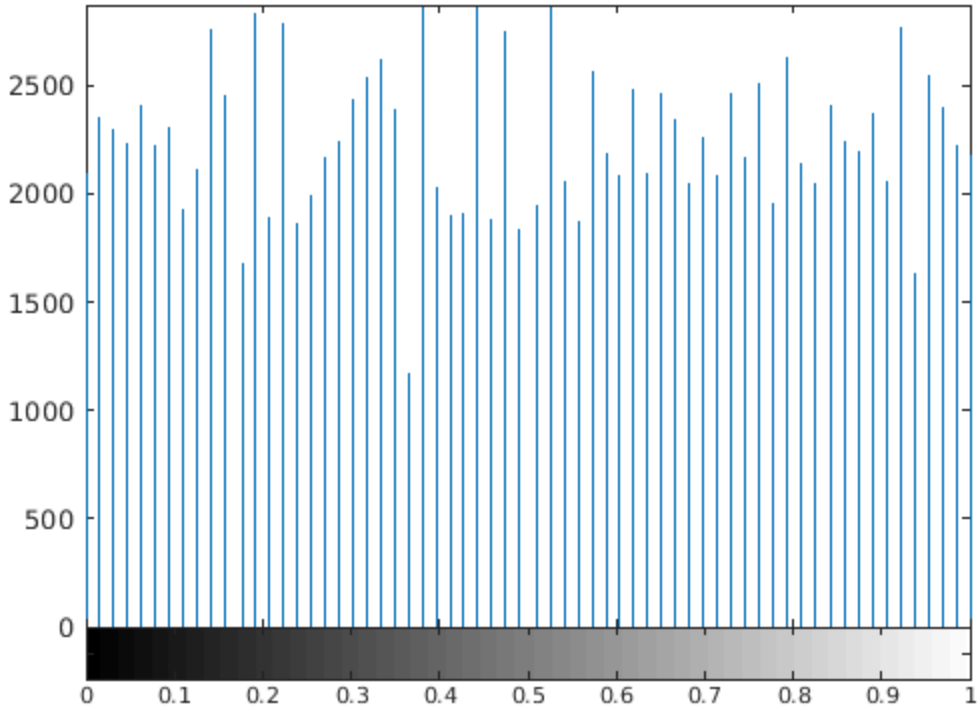
If the original image has a limited range of intensities or if there are regions with low contrast, histogram equalization may not achieve a completely uniform distribution.

Discreteness of Pixel Values:

Image intensity values are often discrete, and the equalization process may not perfectly distribute them due to these discrete values.

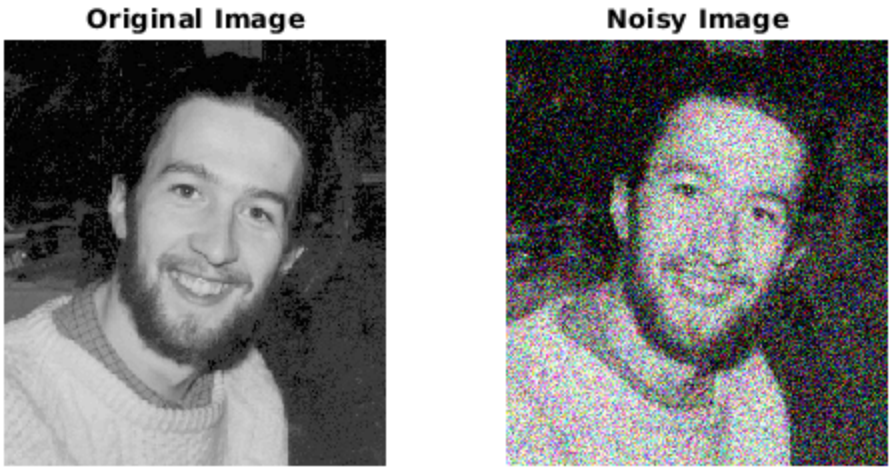
Noise and Local Variations:

Noise or local variations in the image can affect the uniformity of the histogram. Small-scale variations may not be fully addressed by global histogram equalization.



4.2.1, 4.2.2

```
img2 = imread("./assets/Image02.jpg");
noisy = imnoise(img2, 'Gaussian', 0, .04);

figure('name', "Raw Vs. Noisy");
subplot(1,2,1)
imshow(img2);
title('Original Image');
subplot(1,2,2)
imshow(noisy);
title('Noisy Image');
```



4.2.3

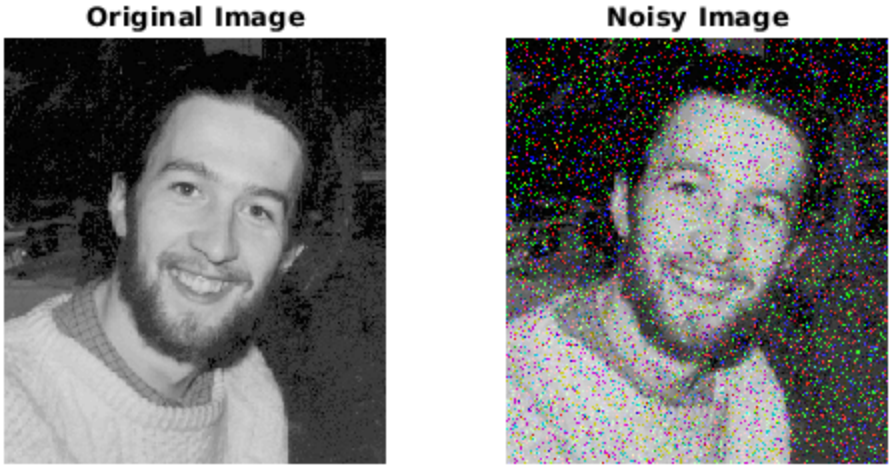
```
kernel = ones(3,3) / 9;
filtered = imfilter(noisy,kernel);

figure('name', "Filtered Image");
subplot(1,3,1)
imshow(img2);
title('Original Image');
subplot(1,3,2)
imshow(noisy);
title('Noisy Image');
subplot(1,3,3)
imshow(filtered);
title('Filtered Image');
```

4.2.4

```
kernel = ones(5,5) / 25;
filtered = imfilter(noisy,kernel);

figure('name', "Filtered Image");
subplot(1,3,1)
imshow(img2);
title('Original Image');
subplot(1,3,2)
imshow(noisy);
title('Noisy Image');
subplot(1,3,3)
imshow(filtered);
title('Filtered Image');
```

4.2.5

```
salt_papper_noisy = imnoise(img2, 'salt & pepper', 0.1);

figure('name', "Raw Vs. Noisy");
subplot(1,2,1)
imshow(img2);
title('Original Image');
subplot(1,2,2)
imshow(salt_papper_noisy);
title('Noisy Image');
```

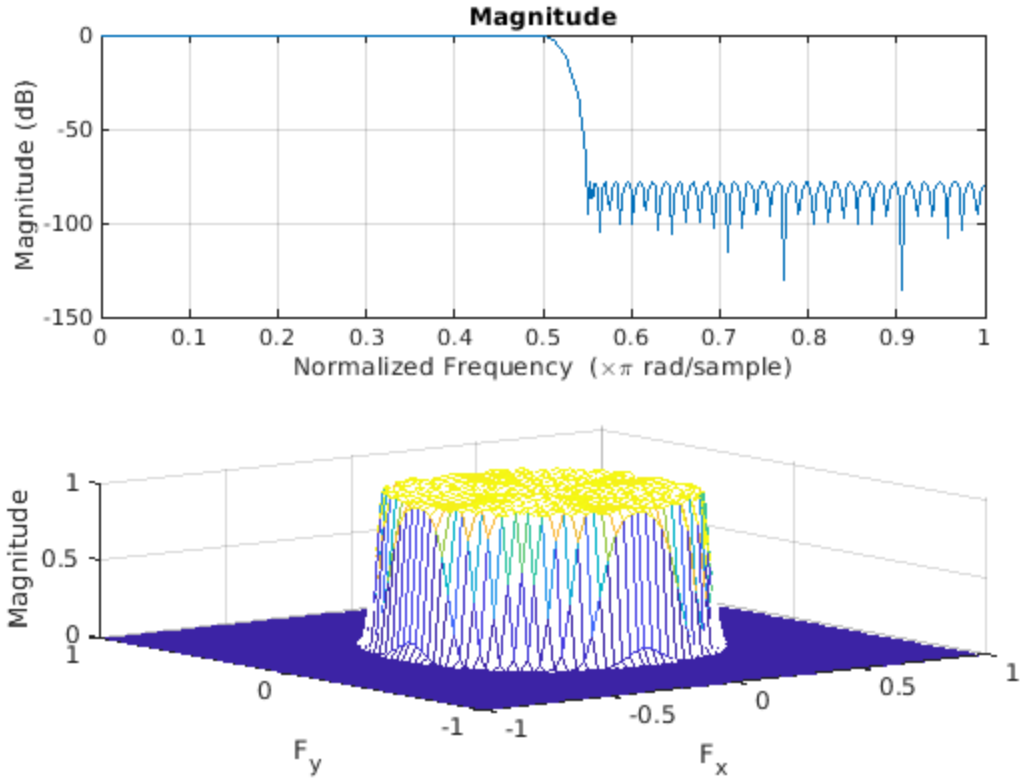


4.2.6

```
kernel = ones(3,3) / 9;
filtered_salt_pepper = imfilter(salt_papper_noisy,kernel);

figure('name', "Filtered Image");
subplot(1,3,1)
imshow(img2);
title('Original Image');
subplot(1,3,2)
imshow(salt_papper_noisy);
title('Noisy Image');
subplot(1,3,3)
imshow(filtered_salt_pepper);
title('Filtered Image');
```

4.2.7

```
load('filter.mat');
filter_FIR = ftrans2(Num);
figure('name', "Filtered Image")
subplot(2,1,1)
freqz(Num);
subplot(2,1,2)
freqz2(filter_FIR);
```



4.2.8

```
FIR_gaussian = imfilter(noisy, filter_FIR);
FIR_salt_pepper = imfilter(salt_papper_noisy, filter_FIR);

figure('name', "Filtered Images")
subplot(2,2,1)
imshow(noisy);
title('Noisy Image');
subplot(2,2,2)
imshow(FIR_gaussian);
title('FIR Gaussian filter');
subplot(2,2,3)
imshow(salt_papper_noisy);
title('Salt & Pepper Noise Image');
subplot(2,2,4)
imshow(FIR_salt_pepper);
title('FIR salt & pepper filter');
```

4.2.10:

```
median_res = medianFilter(salt_papper_noisy,3,3);
figure('name', 'salt&papper noisy image denoised');
imshowpair(salt_papper_noisy, median_res, 'montage');
```

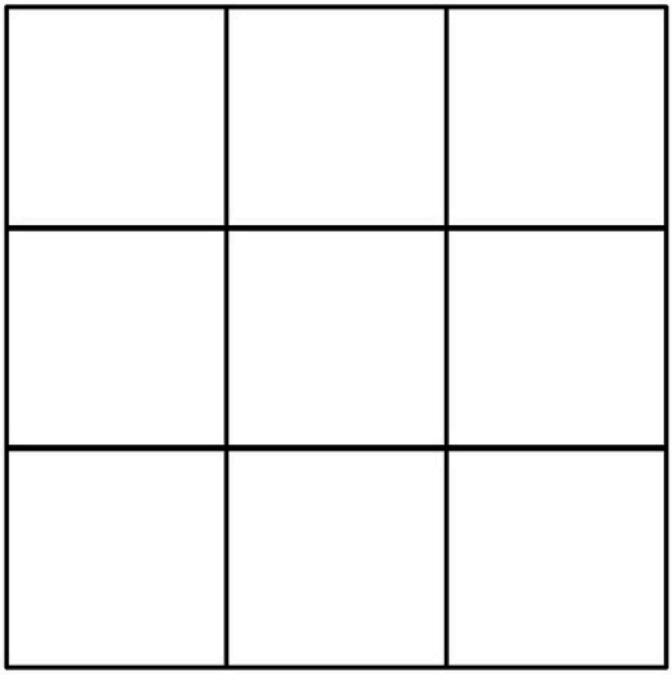
The median filter is more effective against salt-and-pepper noise than an average filter because it is a nonlinear operation that preserves edges. By selecting the median value of neighboring pixels, it is robust to isolated extreme intensity values, common in salt-and-pepper noise. Unlike averaging, the median filter reduces noise without introducing excessive blurring, making it suitable for maintaining image details.

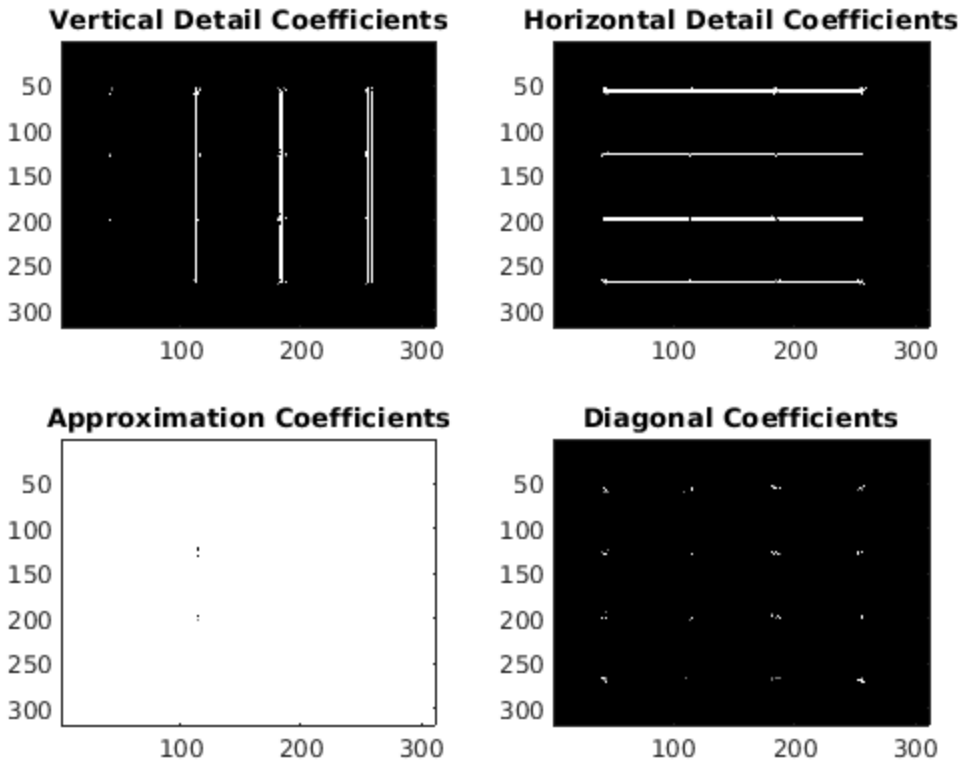
The median filter can introduce a loss of fine details and textures in the image, particularly when the filter window is large. Additionally, the computational complexity of the median filter grows with the size of the window, impacting its efficiency for real-time processing or large images.

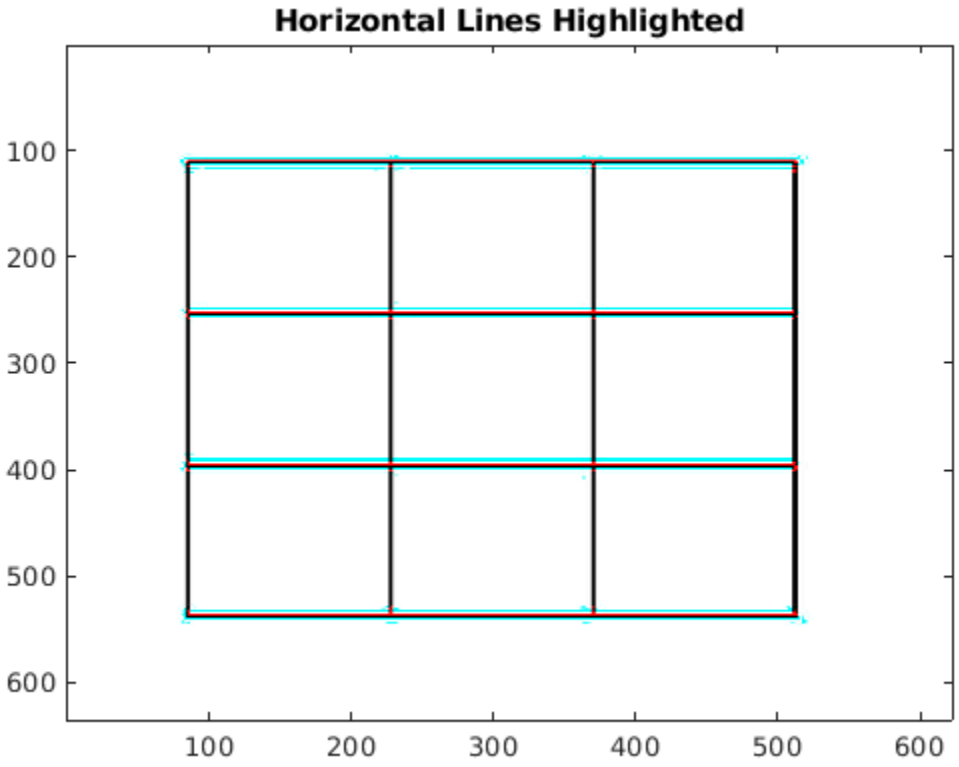
4.3.1

```
img_03 = imread('./assets/Image03.jpg');
figure(1);
imshow(img_03);

image_size = size(img_03);
[cA,cH,cV,cD] = dwt2(img_03,'db1','mode','per');
figure('name', 'wavelet 2D');
subplot(2,2,1)
imagesc(cV);
title('Vertical Detail Coefficients');

subplot(2,2,2)
imagesc(cH);
title('Horizontal Detail Coefficients');

subplot(2,2,3)
imagesc(cA);
title('Approximation Coefficients');

subplot(2,2,4)
imagesc(cD);
title('Diagonal Coefficients');

figure('name', 'Horizontal Lines Highlighted');
tt = imresize(cH, 2);
r = tt(:, :, 1);
g = tt(:, :, 2);
b = tt(:, :, 3);
tt = cat(3, g, zeros([636, 622]), zeros([636, 622]));
```

```
imagesc(tt(1:end-1,:,:) + im2double(img_03));
title('Horizontal Lines Highlighted');
```

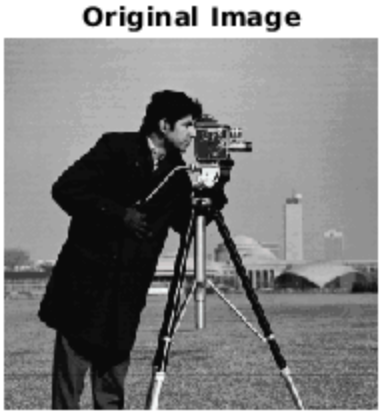



Vertical Detail Coefficients
Horizontal Detail Coefficients
Approximation Coefficients
Diagonal Coefficients
50
100
150
200
250
300
100
200
300

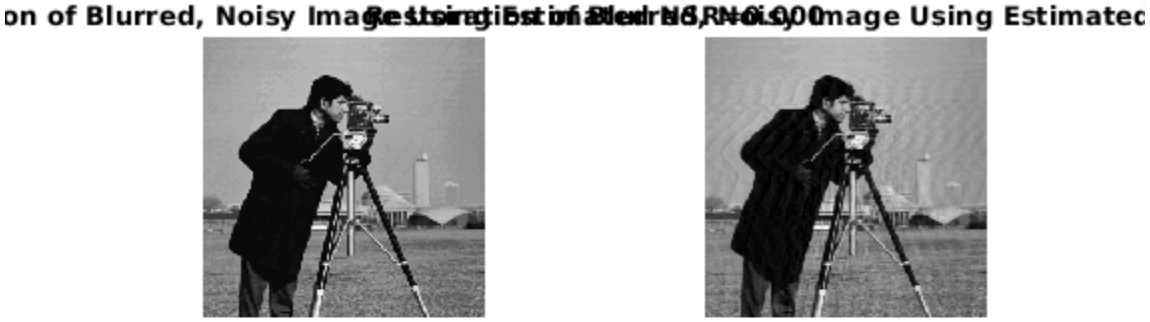

Horizontal Lines Highlighted
100
200
300
400
500
600

4.4.1, 4.4.2

```
close all;
clear all;
clc;

img_04 = im2double(imread('./assets/Image04.png'));
figure('name', 'Original Vs. Blured');
subplot(1,2,1);
imshow(img_04);
title('Original Image');

len = 15;
theta = 20 * pi / 180;
kernel = fspecial('motion', len, theta);
blured = imfilter(img_04, kernel, "conv", "circular");
subplot(1,2,2);
imshow(blured);
title('Blured Image');


figure('name', 'Motion blur filtering')
estimated_nsr = [0, 0.001, 0.01, 0.1];
for i = 1:length(estimated_nsr)
    wnr3 = deconvwnr(blured, kernel, estimated_nsr(i));
    subplot(2,2,i);
    imshow(wnr3);
    title(sprintf('Restoration of Blurred, Noisy Image Using Estimated NSR=
%.3f',estimated_nsr(i)));
end
```


Original Image


Blured Image

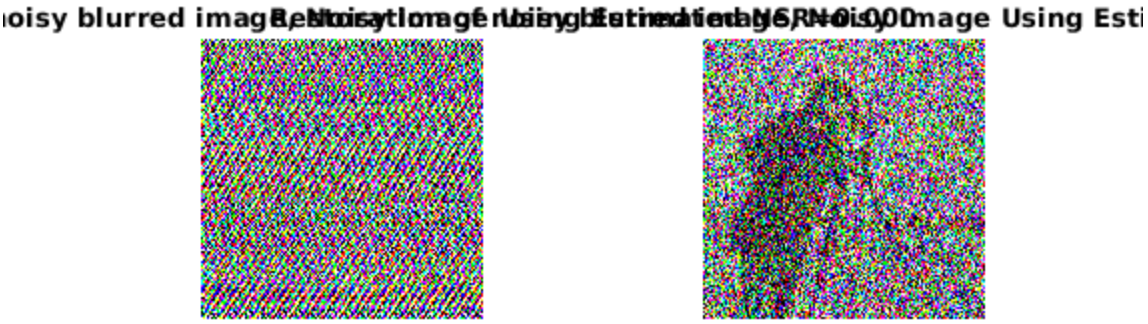

on of Blurred, Noisy Image Using Estimated
on of Blurred, Noisy Image Using Estimated

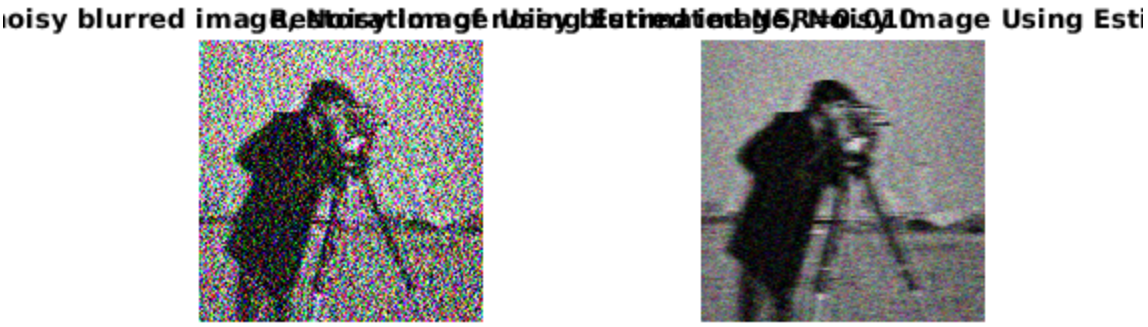
4.4.3:

```
blurred_plus_noise = imnoise(blured,"gaussian",0, 0.01);
figure('name', 'Blurred image + Noise');
imshowpair(blured, blurred_plus_noise, 'montage');
```

4.4.4:

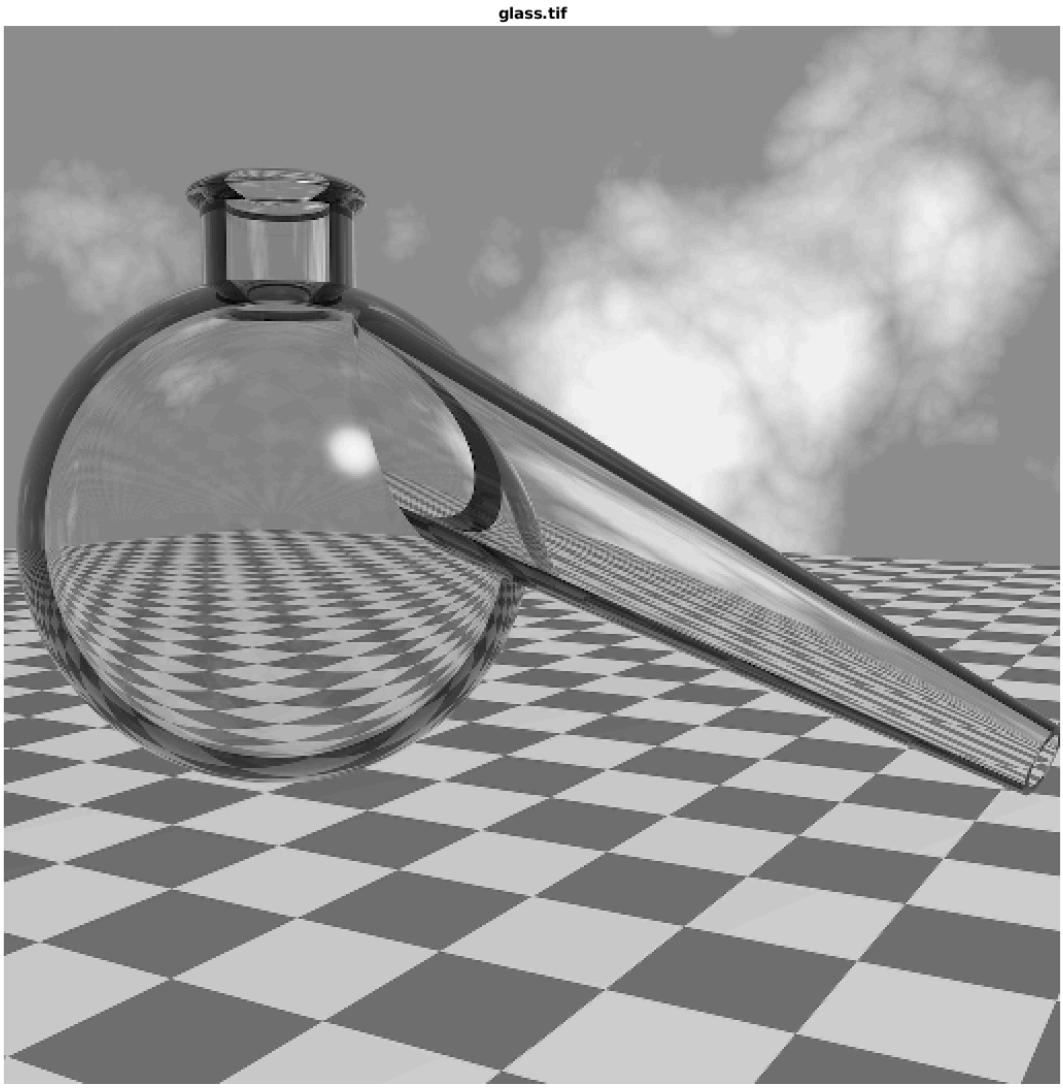
```
figure('name', 'mMtion blur filtering for noisy blurred image')
estimated_nsr = [0, 0.001, 0.01, 0.1];
for i = 1:length(estimated_nsr)
    wnr3 = deconvwnr(blurred_plus_noise, kernel, estimated_nsr(i));
    subplot(2,2,i);
    imshow(wnr3);
    title(sprintf('Restoration of noisy blurred image, Noisy Image Using
 Estimated NSR=%.3f',estimated_nsr(i)));
end
```





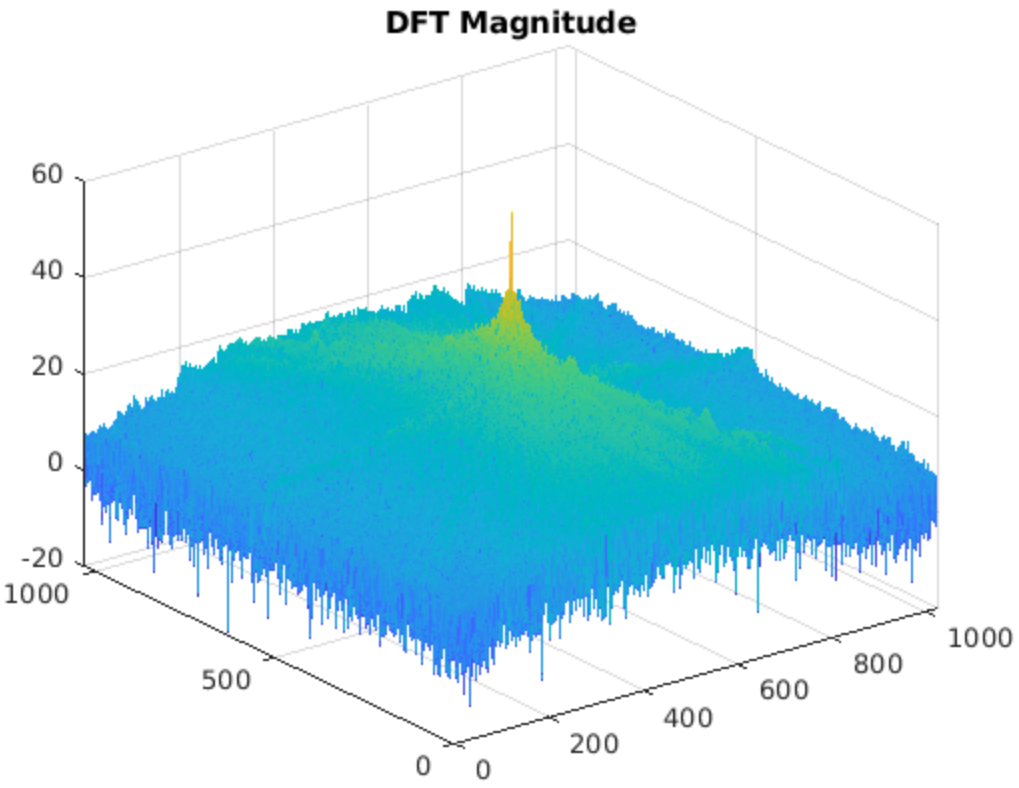
4.5.1:

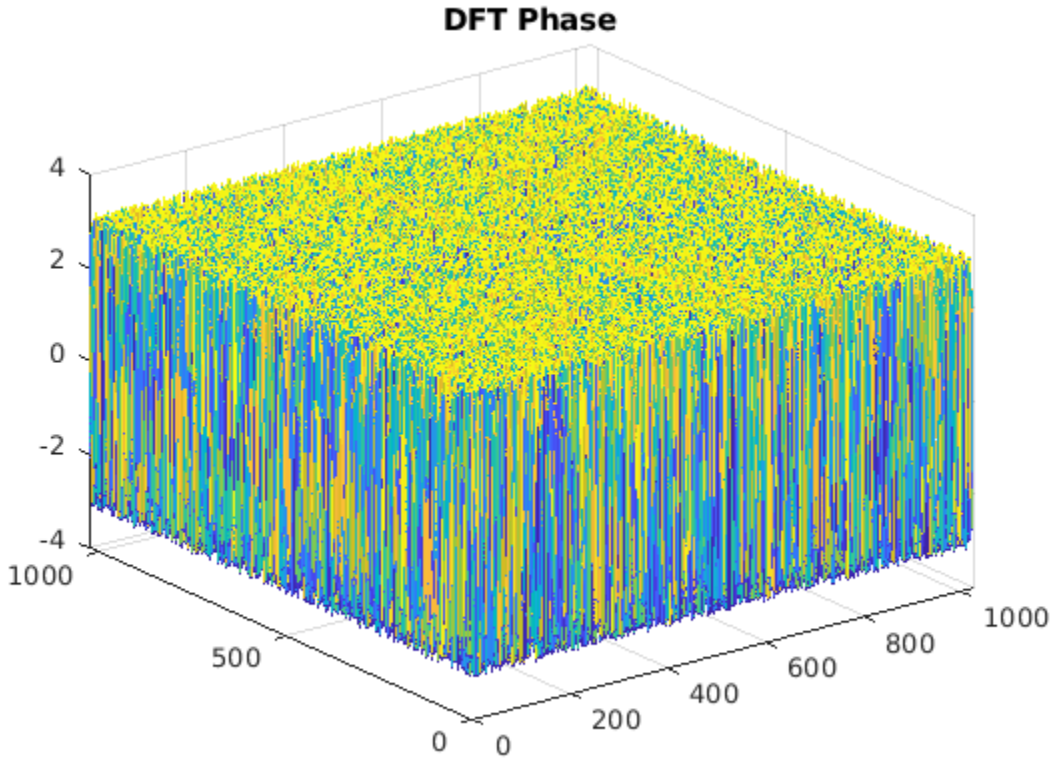
```
img = im2double(imread('./assets/glass.tif'));
figure('name', 'glass.tif')
imshow(img);
title('glass.tif');
```



4.5.2:

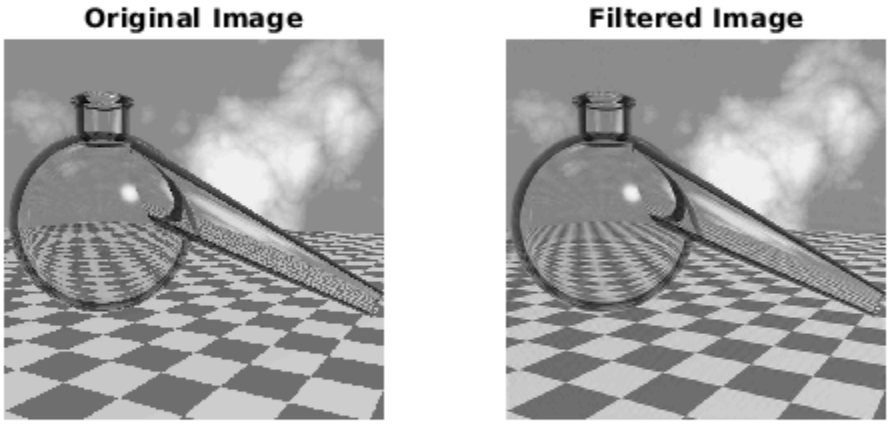
```
img_dft = fftshift(fft2(img));

figure('name', 'DFT Magnitude');
mesh(10 * log10(abs(img_dft)));
grid on;
title('DFT Magnitude');

figure('name', 'DFT Phase');
mesh(angle(img_dft));
grid on;
title('DFT Phase');
```


DFT Magnitude
60
40
20
0
-20
1000
500
0
0
200
400
600
800
1000


DFT Phase
4
2
0
-2
-4
1000
500
0
0
200
400
600
800
1000

4.5.4:

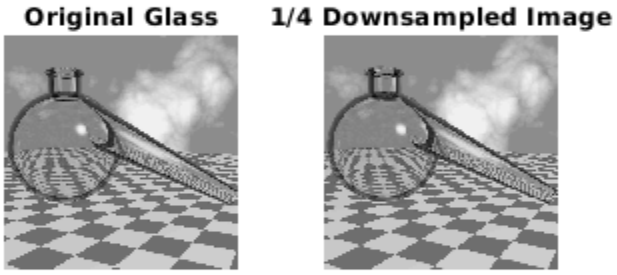
```
output_image = FFT_LP_2D(img, 0.1*pi);

figure('name', 'Original vs Filtered Image');
subplot(1,2,1);
imshow(img);
title('Original Image');
subplot(1,2,2);
imshow(output_image);
title('Filtered Image');
```



4.5.6:

```
img_downsamp = imresize(img, 1/4, 'nearest');

figure('name', 'Original vs Downsampled Image');
subplot(1,3,1);
imshow(img);
title('Original Glass');
subplot(1,3,2)
imshow(img_downsamp);
title('1/4 Downsampled Image');
img_downsamp_filtered = FFT_LP_2D(img_downsamp,0.2*pi);
subplot(1,3,3)
```

```
imshow(img_downsamp_filtered);
title('Filtered 1/4 Downsampled Image');
```



4.5.3:

```
function output_image = FFT_LP_2D(input_image, cutoff_frequency)
    [ix,iy,iz] = size(input_image);
    hr = (ix-1)/2;
    hc = (iy-1)/2;
    [x, y] = meshgrid(-hc:hc, -hr:hr);

    mg = sqrt((x/hc).^2 + (y/hr).^2);
    lp = double(mg <= cutoff_frequency);

    IM = fftshift(fft2(double(input_image)));
    IP = zeros(size(IM));
    for z = 1:iz
        IP(:,:,z) = IM(:,:,z) .* lp;
    end
    output_image = abs(ifft2(ifftshift(IP),'symmetric'));
end
```

4.2.9:

```
function denoise_im = medianFilter(img,k1,k2)
```

```
    [m,n,l] = size(img) ;
    k11 = floor(k1/2) ;
    k22 = floor(k2/2);
    denoise_im = img;
    for i = 1 : m
        if(mod(k1,2) == 0 ) || (mod(k2,2) == 0)
            break;
        end
        for j = 1 : n
            for p = 1 : l
                temp = img(max(i - k11,1):min(i+k11,m) , max(j - k22,1):min(j
 + k22,n) , p);
                temp = reshape(temp,[],1) ;
                denoise_im(i,j,p) = median(temp);
            end
        end
    end
end
```

Published with MATLAB® R2023a