NANYANG TECHNOLOGICAL UNIVERSITY

CZ4003 COMPUTER VISION

LAB Deliverables

Assignment 1

Point Processing

Spatial Filtering

Frequency Filtering

Imaging Geometry

Sam Jian Shen (U1821296L)

Date of Submission: 6th of October 2020

School of Computer Science and Engineering

Nanyang Technological University

Contents

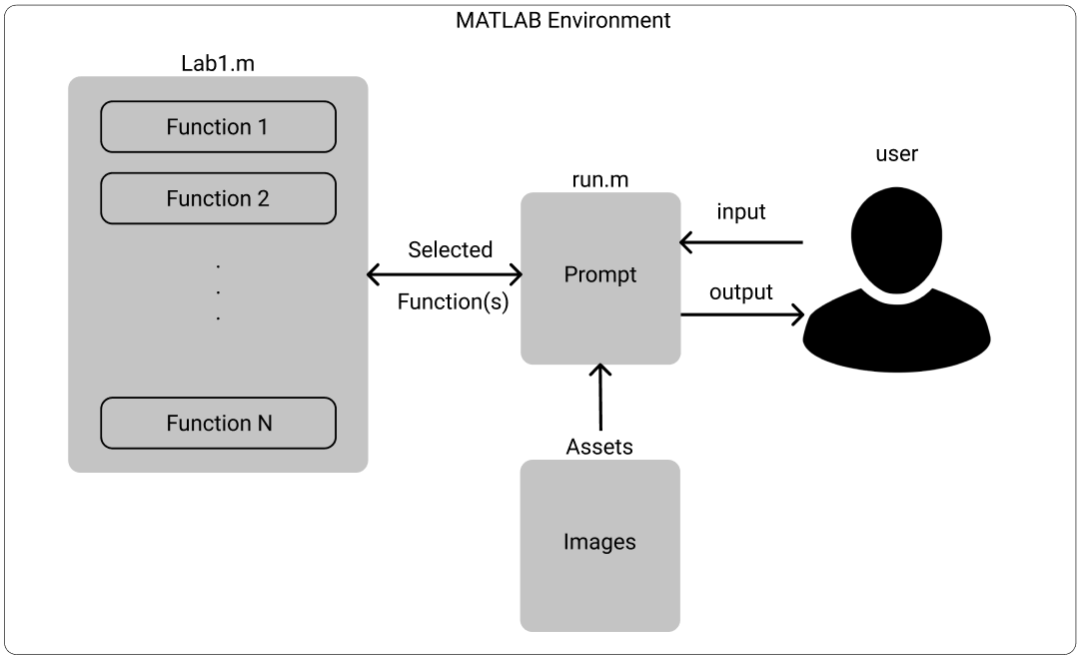
Introduction

Before sharing the analysis and result of this assignment, I would like to share the brief thought process before implementation although this section is not graded.

Common Issues

- Friendliness of using MATLAB
    - By default, most of the MATLAB exercises' functions are Command Window execute instead of script / object-oriented base execution.
    - This result in tedious process when formulate and running the functions. The experience is daunting.
    - I would like to create in a way such that it is more reusable and can be used for future user(s) to test and experience the functions needed for this assignment using MATLAB. This may help user(s) to have a better and enjoyable learning experiences while doing the assignment.

Simple Informal Software Architecture

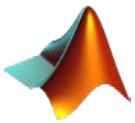


The analysis and result will showcase the highlights and key points of my program (i.e main part of the code will be shown here). Therefore, do try use my program to get the full experience.

Setup and Execute

This section details can be found in README.md

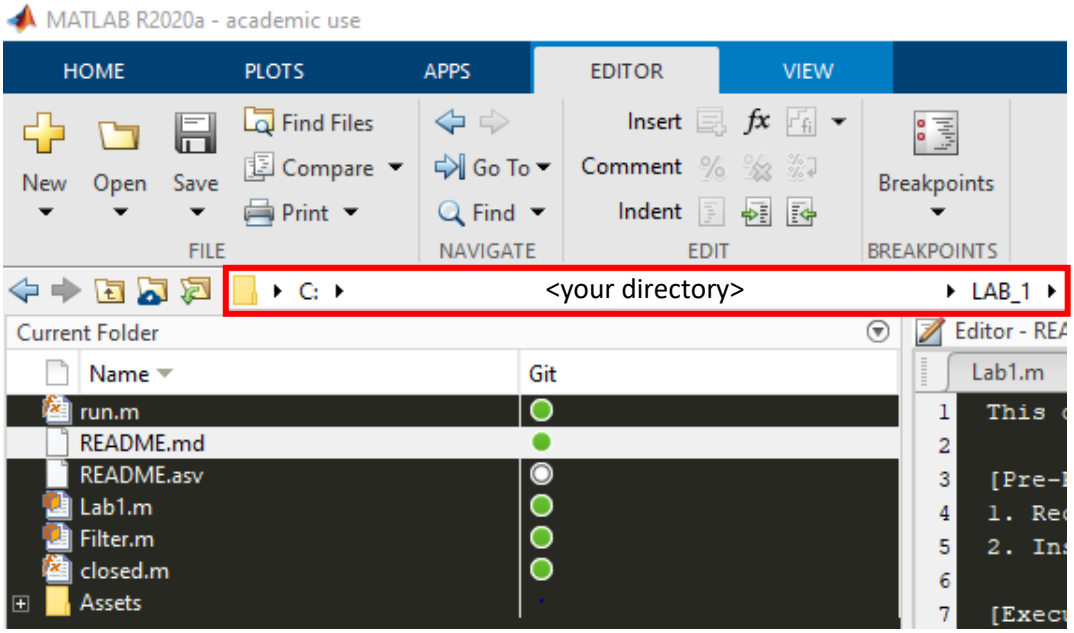
Pre-Requisite



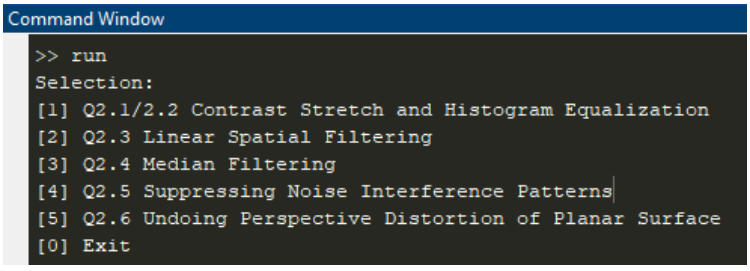
- Recommended MATLAB version **R2020a** and above installed.
- Installed **Image Processing Toolbox** software package from MATLAB Extension

How to Use?

1. Run **MATLAB** program
2. Please ensure the current directory is at **LAB_1** file

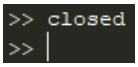


3. At the 'Command Window' type '**run**' and enter



   a. The Command Window various **options** should be visible by then

4. **Follow the prompt instruction(s)** in Command Window accordingly
5. [Optional] At 'Command Window' type '**closed**' to close all opened window



2.1 Contrast Stretching

a. Read Image and output of image information

Code:

```
URL = 'assets\mrttrainbland.jpg';
```

- Get the path and filename of the file.

```
Pc = imread(URL);
```

- Image can read using function imread(<directory to the image and filename>).

```
whos Pc;
```

- 'whos' is a function that return a certain information about the image.

Output:

```
Name         Size                Bytes  Class    Attributes

Pc         320x443x3            425280  uint8
```

The size is referred to height x width x number of channel(s). In this case, the image consist of 3 channels is due to RGB (Red, Green, Blue) where each channel holds an intensity value of a class uint8 from 0 to 255.

b. Display image using 'imshow' in RGB and Grayscale

Code:

```
imshow(img_Origin);
```

Output:

Code:

```
P = rgb2gray(img_Origin);
```

- Convert to gray-level image

Output:

However, it seems not obvious in our naked eye unless we zoom in from the figure.

Origin Image consist of slight green/brown colour, however it is still not obvious

One way to prove it is using 'whos'

```
Name                      Size                Bytes  Class    Attributes

img_ContrastStretch       320x443            141760  uint8
img_GreyScale             320x443            141760  uint8
img_Origin                320x443x3          425280  uint8
```

From here we can see that 'img_GrayScale' has been reduced to one channel which is gray level. Notice the bytes is reduce proportionally to the number of channels of the image.

c. Min-Max Intensity of the current Gray Image

Code:

```
show_MinMax_Intensity_GreyScale = [min(img_GreyScale(:)),max(img_GreyScale(:))] ;
```

- Return maximum and minimum gray-level intensity value in the grayscale image

Output:

```
 min   max
 13    204
```

The lowest intensity is 13 and highest intensity is 204, in order to make full use of gray-level normalize to 0 to 255 range instead. This helps to increase the richness of gray-level such that the image is clearer or more define. One way to do is using contrast stretching.

d. Contrast stretching operation

Code:

```
% Convert to double and re-scale between 0 to 1 range
img_GreyScale_double = im2double(img_GreyScale);
img_GreyScale_min = im2double(min(img_GreyScale(:))); %min(img_GreyScale(:));
img_GreyScale_max = im2double(max(img_GreyScale(:))); %max(img_GreyScale(:));
% Apply contrast stretch formula
P2_double = (255*(imsubtract(img_GreyScale_double,img_GreyScale_min)))/(img_GreyScale_max-img_GreyScale_min); % left minus right
```

Notice that the image must convert to double such that the normalization computation can be process as the existing data type is uint8.

Basically, contrast stretching use the idea of normalization formula multiply by 255 because the range is 0 to 255.

Normalization formula:

$$z = \frac{x \text{ - } \min(x)}{\max(x) \text{ - } \min(x)}$$

To prove whether it does stretch to 0 to 255 we need to use min and max intensity function apply on the image.

Code:

```
show_MinMax_Intensity_GreyScale = [min(img_GreyScale(:)),max(img_GreyScale(:))] ;
show_MinMax_Intensity_ContrastStretch = [min(img_ContrastStretch(:)),max(img_ContrastStretch(:))] ;
disp('    min  max (pixel(s))')
disp(show_MinMax_Intensity_GreyScale);
disp(show_MinMax_Intensity_ContrastStretch);
```

Output:

```
    min  max (pixel(s))
    13   204

     0   255
```

e. Display image after contrast stretching

Code:

```
P2 = uint8(P2_double);
```

- Convert to uint8 data type such that it is in the range of 0 to 255

```
assignin('base','img_ContrastStretch',P2);
```

- Store into workspace with variable name called ‘img_ContrastStretch’

```
imshow(img_ContrastStretch);
```

Output:

Left 2 images are placed for comparison purpose.

Notice the contrast stretch image has darker pixel and brighter pixel range. It has proven the image has enhanced.

Alternatively, we can show the contrast stretch image that omit the datatype whether it is double or byte form. It can still display.

Code:

```
imshow(P2_double,[]);
```

Output:

2.2 Histogram Equalization

a. Display image intensity histograms with different bins value

Code:

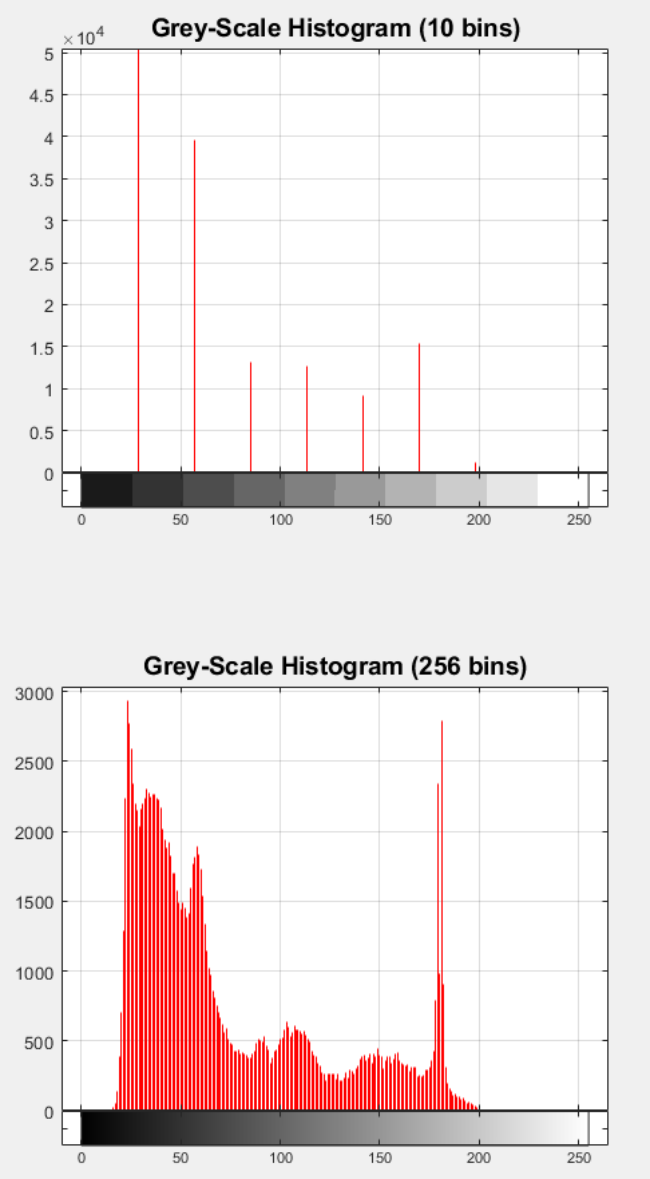
```
imhist(img_GreyScale,10);
```

- 10 bins

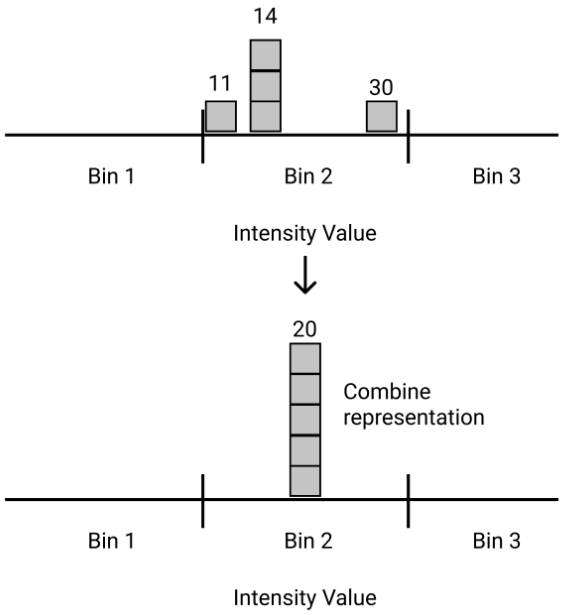
```
imhist(img_GreyScale,256);
```

- 256 bins

Output:



To check whether it is 10bins I added the intensity bar below of each sub-plot and extend the min and max to see the plot better. The 10bins has 10 distinct value of gray-level. Each bin contains the number of occurrences of similar pixels values. To illustrate better how bins, work as follows:



With this concept, it shows the different between 256 bins and 10 bins, having less bins means more general representation of the value. It is useful when looking for cluster values compare to other cluster value. While having more bins means providing more details and features about the image. In this case, 256 bins able to tell all the number of different pixels intensity value.

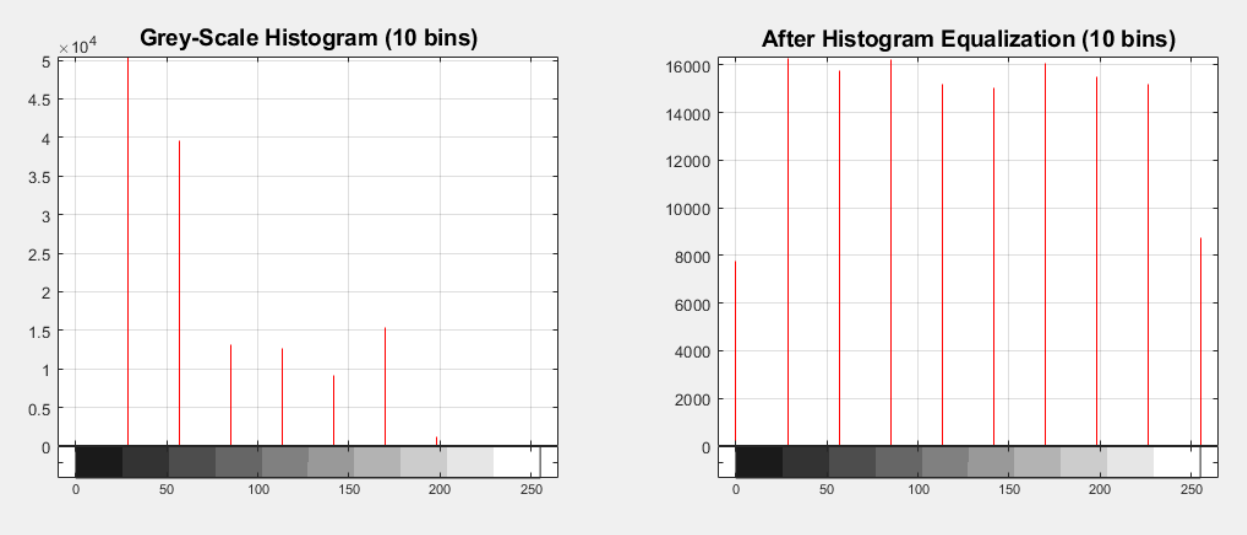
b. Histogram Equalization Operation

Code:

```
P3 = histeq(img_GreyScale,255);
```

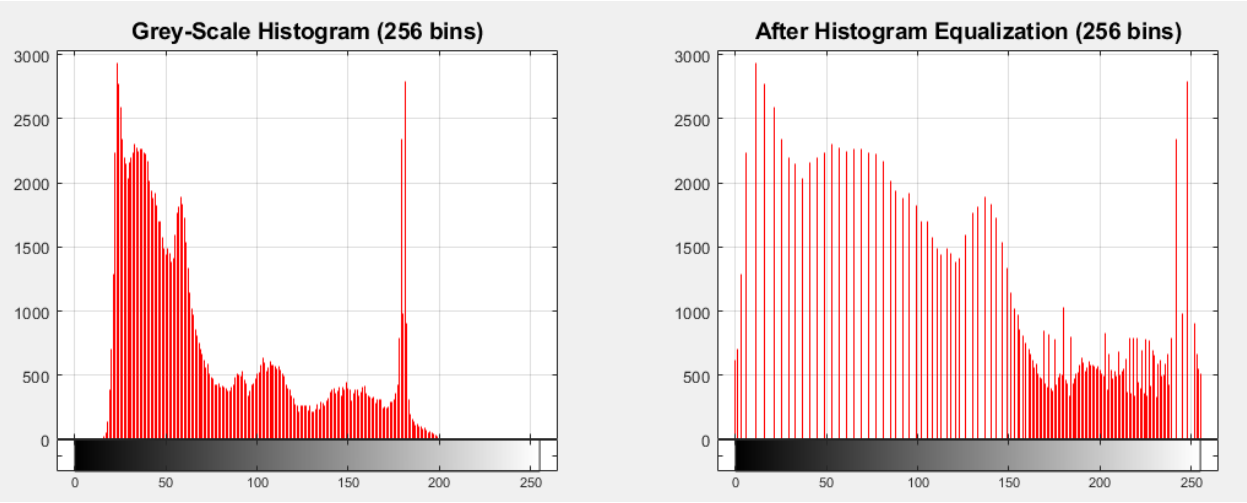
- Using Gray image apply the operation with 255bins + 1 since 0 is the initial.

Output:



Left side of the sub-plot is for illustration purpose.

From 10bins it can be observed that the distribution of pixel is evenly spread after histogram equalization on all possible gray-level between 0 to 255.



Left side of the sub-plot is for illustration purpose.

Notice that we can see how histogram equalization spread the bins to 0 to 255 range, notice smaller occurrence are cluster closer together, bigger occurrence is far apart from each other. Such phenomenon is because of its high occurrence value result is larger change in cumulative distribution frequency.

The similarities of both is that, it retains the min-max level of intensity range. The differences are that the 10bins shows equalized, however 256bins does not shows. This shows that histogram is only approximately flat not absolute flat.

c. Rerun part b again

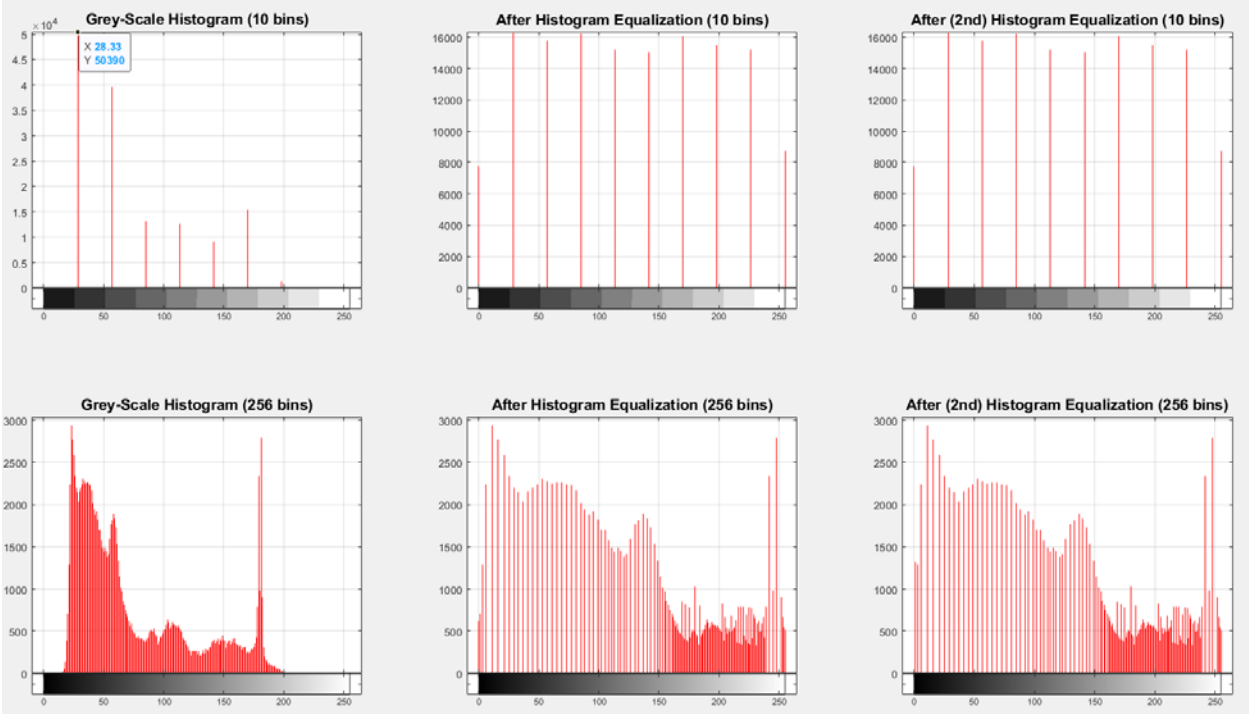
Code:

```
img_HistEq2ndRound = histeq(img_HistEq,255);
% Store into workspace
assignin('base','img_HistEq2ndRound',img_HistEq2ndRound);
```

```
imhist(img_HistEq,10);
```

```
imhist(img_HistEq,256);
```

Output:



Left 2 columns of the sub-plot are there for illustration purpose.

After running 2nd round histogram equalization. There is no improvement in the result and no improvement in terms of uniform. The reason it remains unchanged is because the histogram is already equalized which mean no longer can stretch. Therefore, it is stagnant.

To see how equalized the histogram, using 10bins, display the bins values

[Grey-Scale Histogram (10 bins)]

Bin Count (pixel(s))	Bin Location (greyness)
3.00	0.00
50391.00	28.33
39642.00	56.67
13218.00	85.00
12633.00	113.33
9236.00	141.67
15414.00	170.00
1223.00	198.33
0.00	226.67
0.00	255.00

2nd Histogram Equalization (10 bins)

[Before] Bin Count (pixel(s))	[Before] Bin Location (greyness)	[After] Bin Count (pixel(s))	[After] Bin Location (greyness)
7776.00	0.00	7776.00	0.00
16251.00	28.33	16251.00	28.33
15781.00	56.67	15781.00	56.67
16208.00	85.00	16208.00	85.00
15207.00	113.33	15207.00	113.33
15037.00	141.67	15037.00	141.67
16090.00	170.00	16090.00	170.00
15493.00	198.33	15493.00	198.33
15175.00	226.67	15175.00	226.67
8742.00	255.00	8742.00	255.00

From this view we can clearly see the values changes when histogram equalization is applied. And no changes in value after re-apply histogram equalization function.

2.3 Linear Spatial Filtering

a. Generate filter (kernel)

Code:

```
result = fspecial('gaussian',[x,y],sigma);
```

- It is a pre-defined function in MATLAB that run the given formula to generate gaussian average filter accordingly.

```
imshow(result);
```

- This shows distributed intensity in the 2 dimensions 5 by 5 array, it is 5 because of request dimension for this kernel is 5.

```
mesh(result);
```

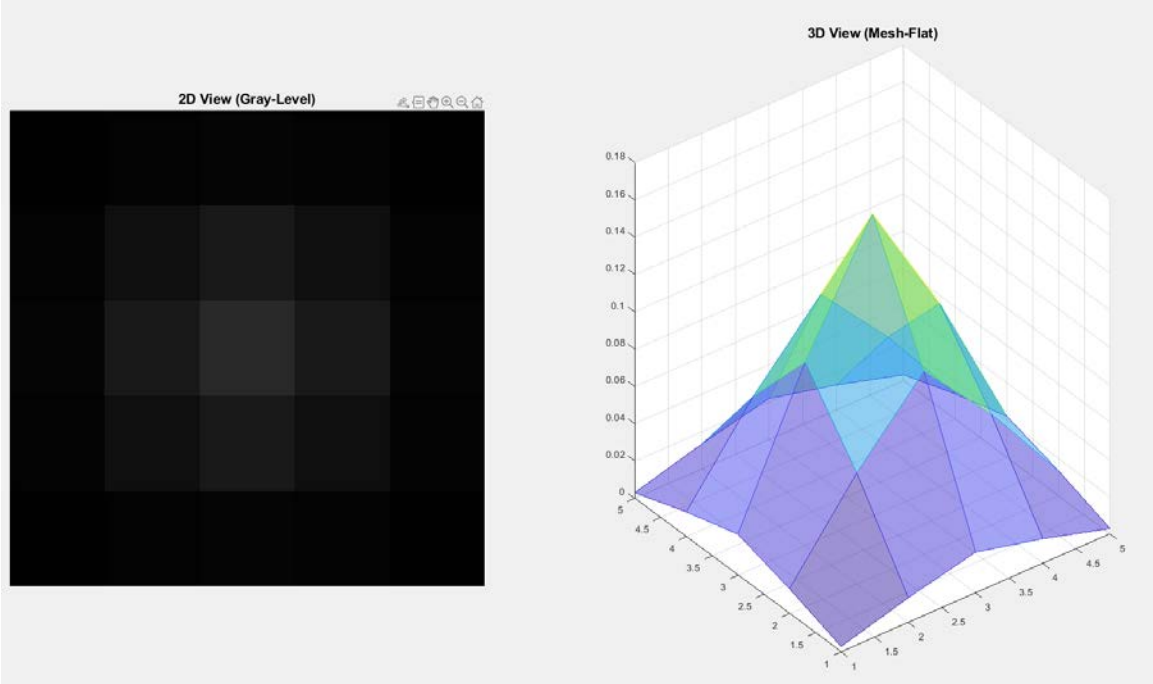
- This shows distributed intensity in 3 dimensions view.

Output:

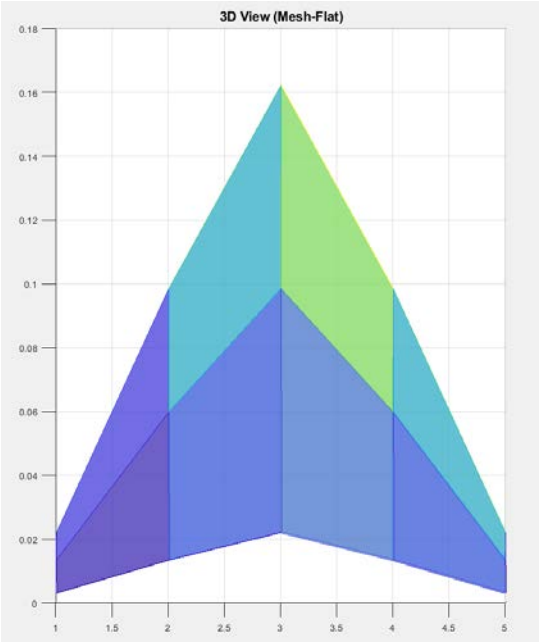
Sigma = 1

```
Individual Pixel(s)/Element(s) Intensity Value:
    0.0030    0.0133    0.0219    0.0133    0.0030
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0219    0.0983    0.1621    0.0983    0.0219
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0030    0.0133    0.0219    0.0133    0.0030

Sum of all Pixel(s)/Element(s) in Y-Axis (Column):
    0.0545    0.2442    0.4026    0.2442    0.0545

Sum of all Pixel(s)/Element(s):
     1
```

This shows the approximate intensity value individually, center has higher influence/weight as compare to it's surrounding. All elements are normalized to 1 as seen from the above the sum of all elements is equal to 1.



From 2 dimensions view we can see the center 9 boxes has stronger weights compare to last row or column of the. From 3 dimensions view, we can observe a peak in the distribution.

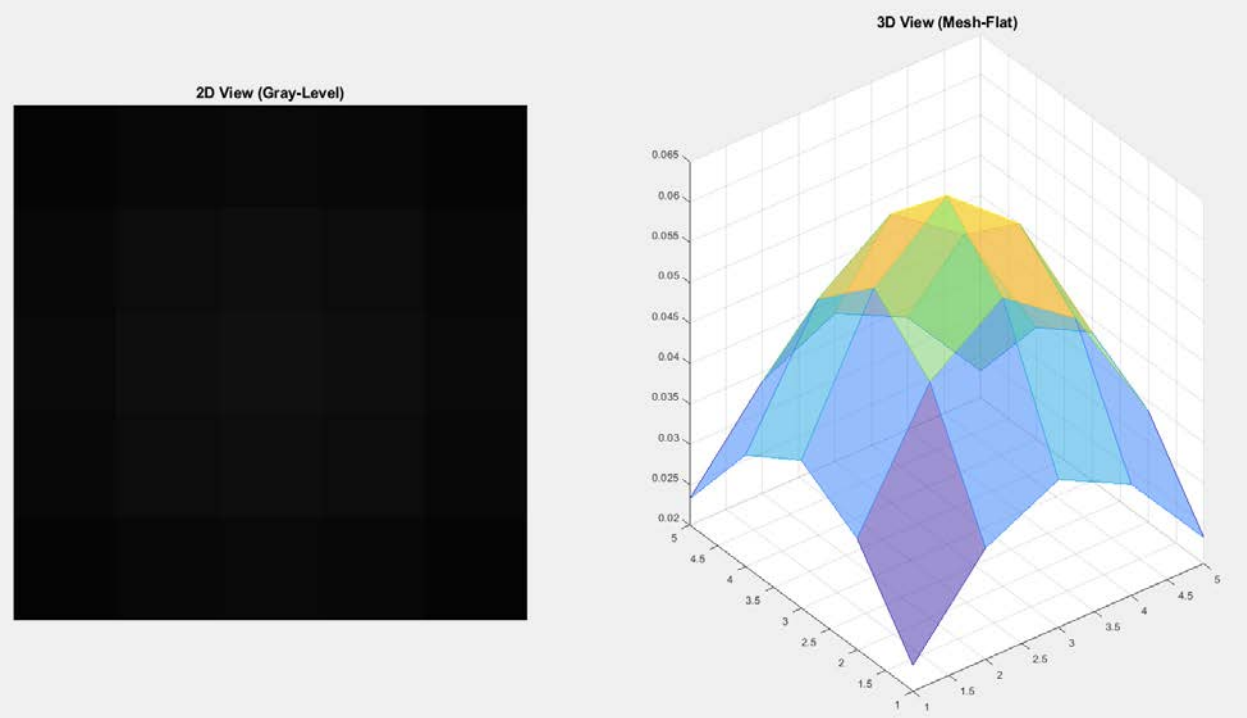


By rotating the 3D to side view, we can see the distribution of gaussian average filter clearly.

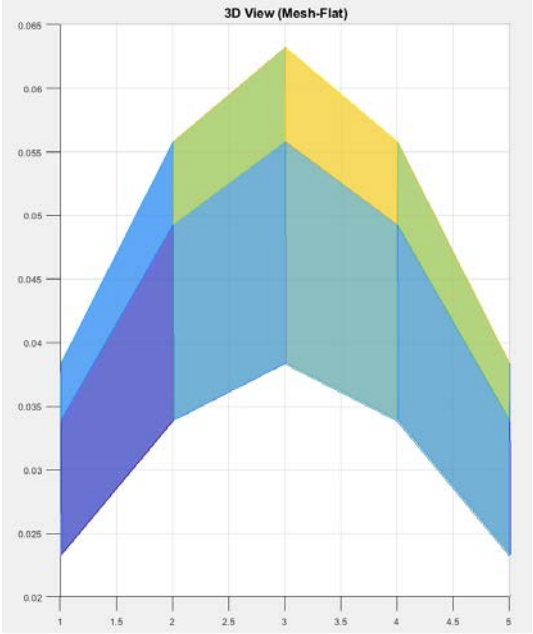
Sigma = 2

```
Individual Pixel(s)/Element(s) Intensity Value:
    0.0232    0.0338    0.0383    0.0338    0.0232
    0.0338    0.0492    0.0558    0.0492    0.0338
    0.0383    0.0558    0.0632    0.0558    0.0383
    0.0338    0.0492    0.0558    0.0492    0.0338
    0.0232    0.0338    0.0383    0.0338    0.0232

Sum of all Pixel(s)/Element(s) in Y-Axis (Column):
    0.1525    0.2218    0.2514    0.2218    0.1525

Sum of all Pixel(s)/Element(s):
    1.0000
```

This shows the approximate intensity value individually, center has higher influence/weight as compare to its surrounding. All elements are normalized to 1 as seen from the above the sum of all elements is equal to 1.



From 2 dimensions view we can barely the weights because the distribution is gentler. From 3 dimensions view, we can observe a peak in the distribution.



By rotating the 3D to side view, we can see the distribution of gaussian average filter clearly.

In comparison

- the higher the sigma value the gentler the distribution. Therefore, the weights are more widely spread-out evenly.
- the lower the sigma value the steeper the distribution. Therefore, the weights are less spread-out evenly.

b. View gaussian noise image ('ntugn.jpg')

Code:

```
URL = 'assets\ntugn.jpg';
```

- Getting the directory path

```
Pc = imread(URL);
```

- Read the image

```
assignin('base','img_Origin',Pc)
```

- Store into a workspace

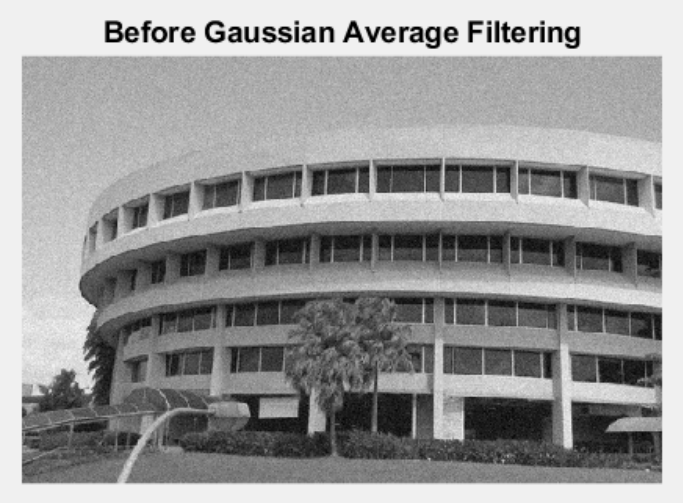
```
img_Origin = evalin('base','img_Origin');
```

- Retrieve from workspace

```
imshow(img_Origin);
```

- Display the selected image

Output:



Notice the noise are subtle and spread out gently throughout the image

c. Apply filter image and display the image

Code:

```
img_Origin_double = im2double(img_Origin);
```

- In order to process it need to convert to double to normalize equally as the kernel where the range is 0 to 1

```
img_conv = conv2(kernel, img_Origin_double);
```

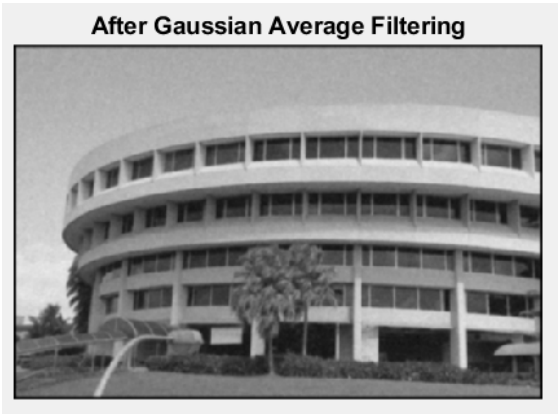
- Apply convolution function with image and kernel (filtering)

```
imshow(img_conv);
```

- Display the result of the image after filtering

Output:

Sigma = 1



Sigma = 2

Notice a big problem in the output which is the black faded border. The reason for such case is because when the filter is applied on the extreme elements, the padding is perceived as zeros therefore most weights are influence by the zero's elements. This return a low intensity value which result in blackish border.

A work around for this having replicate or extension of pixel value beyond the dimension of the image. In MATLAB there is this function called 'imfilter', it contains a properties call 'replicate' that does this exact task.

Code:

```
img_filter = imfilter(img_Origin_double,kernel, 'replicate');
```

- The function that provide image extension.

```
imshow(img_filter);
```

- Display the result

Output:

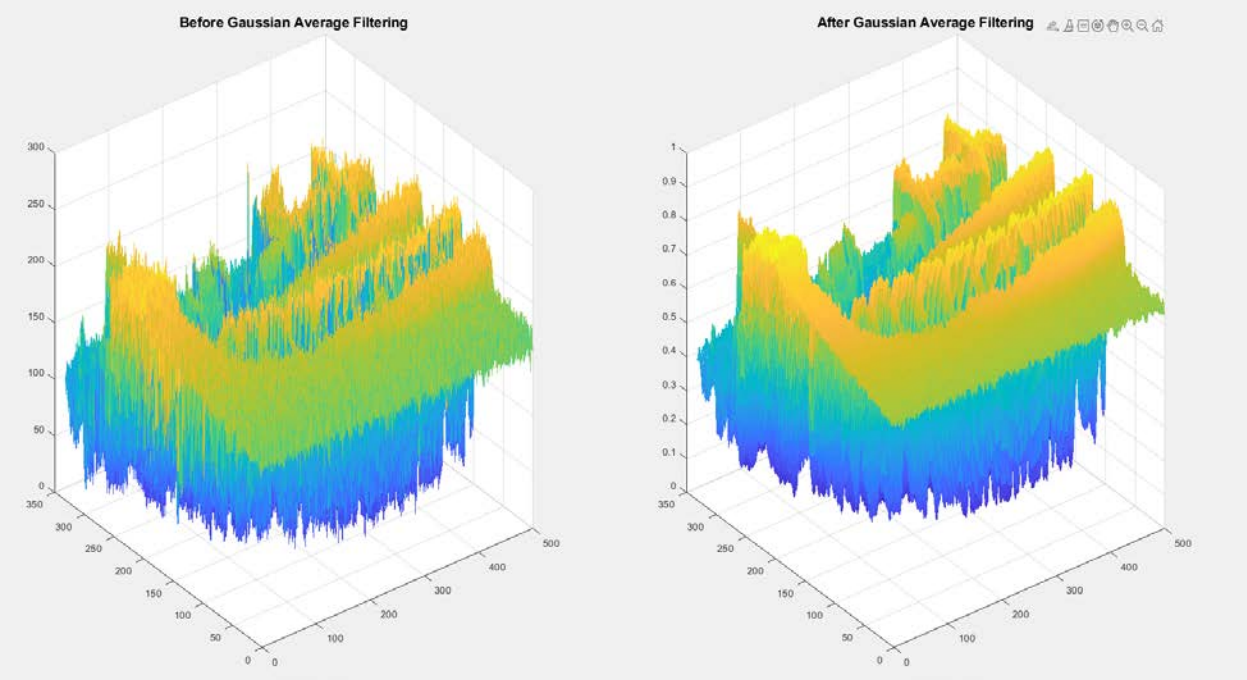
Sigma = 1

Sigma = 2

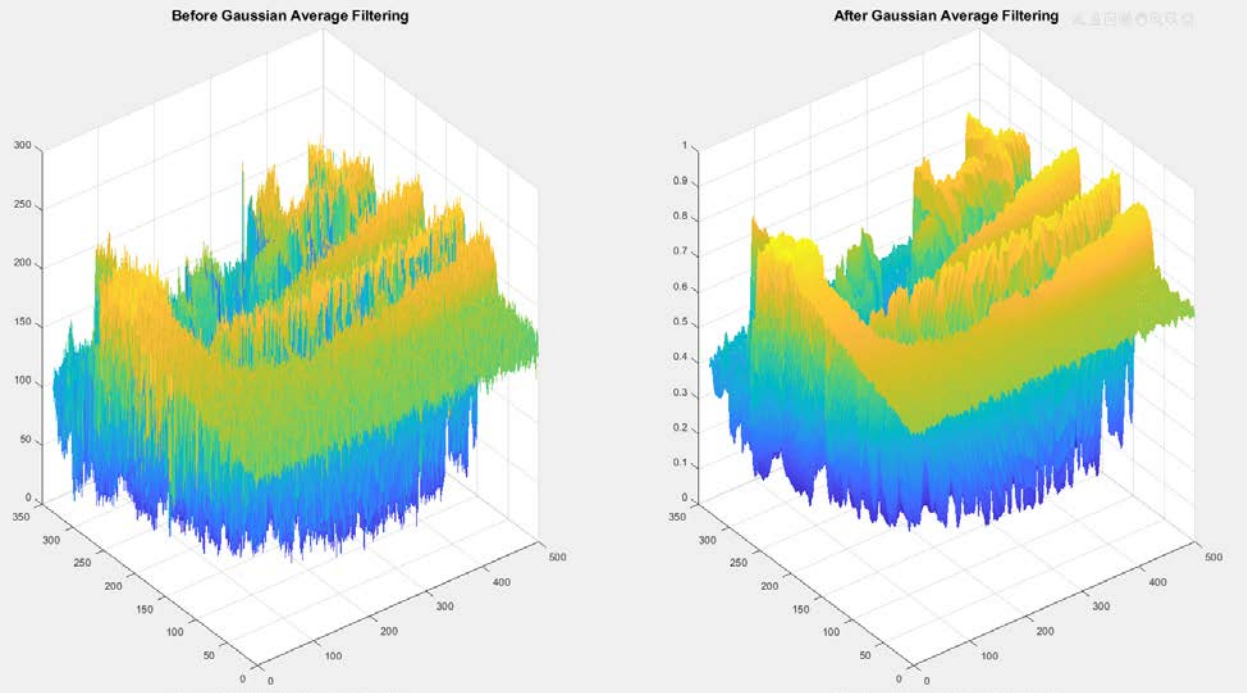
Base on observation, both images manage to suppress the gaussian noise. The image details richness is more prominent when sigma is equal to 1 as compare to 2. This is due to the fact the steep weights factor from the center point of the kernel. Therefore, sigma = 2 appear to be blurrier and less detail.

To understand the result further, I observed them in mesh view instead.

Sigma = 1



Sigma = 2



Using mesh view we can clearly see the effectiveness of the filter used. It manages to suppress the gentle spike on the left-hand side of the plot. The noise is suppress as seen on the right-hand side of the plot.

In comparison, sigma 2 can be seen to be better at tolerate noise compare to sigma 1.

This does not exactly tell how the kernel work, so I take the cross section of both mesh and convert them into 2D plot

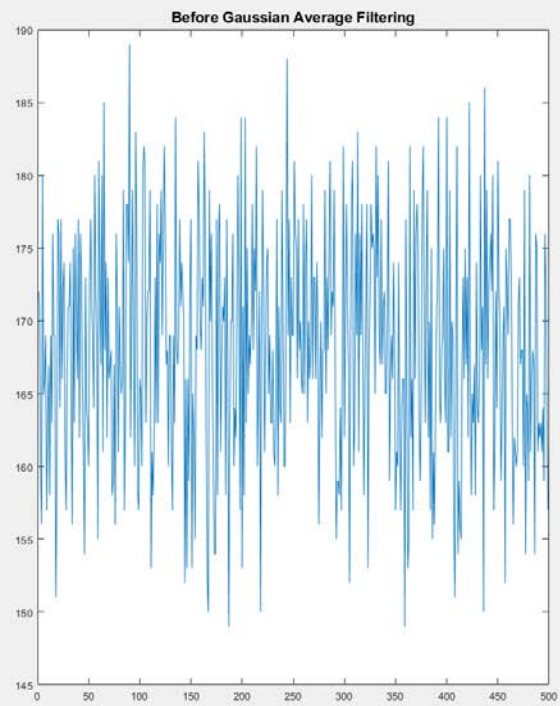
Code:

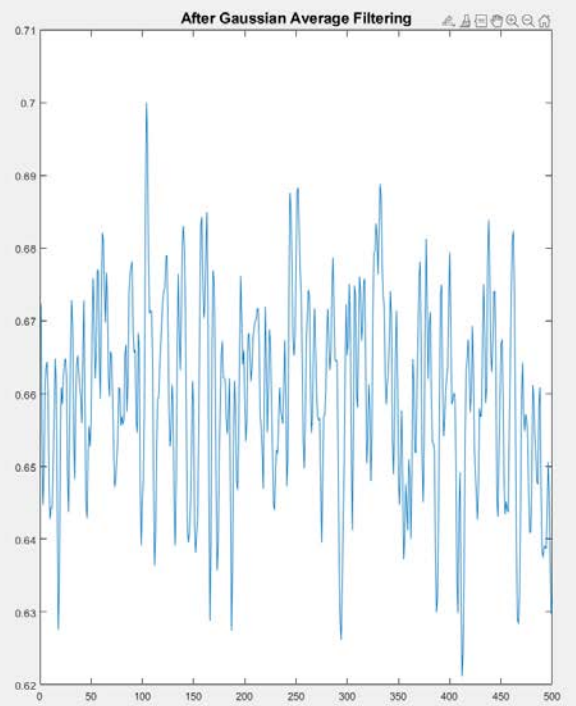
```
plot(img_Origin(1,:));
ussian Average Filterir
plot(img_filter(1,:));
```

- Plot both cross section of origin and filter image (selected from in the 1st row only)

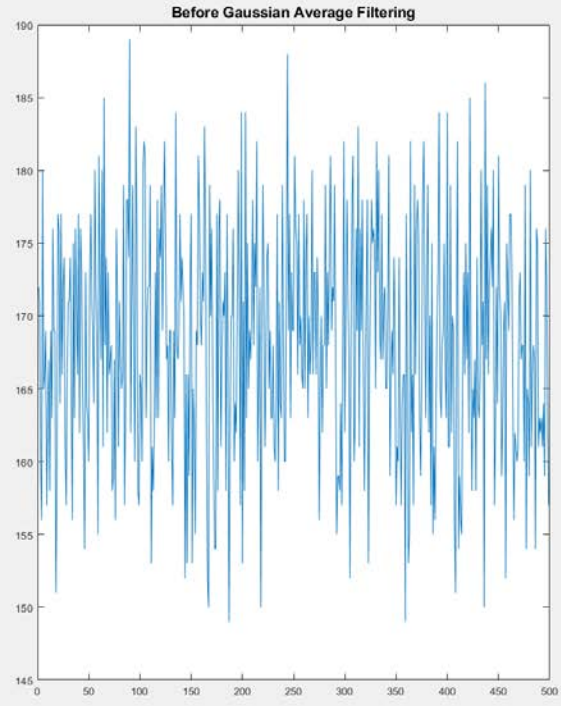
Output:

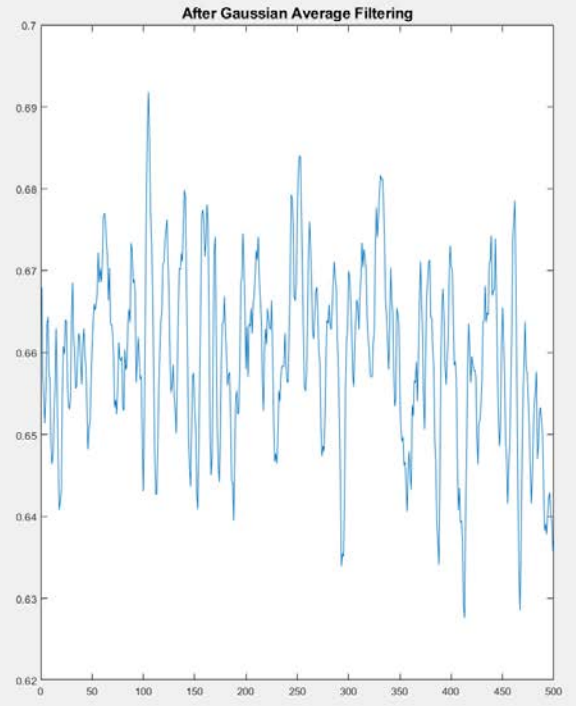
Sigma = 1





Sigma = 2





Notice I forget to normalize them equally. However, the scale should not affect the interpretation of the result.

From here we can observe the spike is mostly suppress and notice the details are flattened. This is the reason the image details are lost.

However, the result from cross section is not accurate since it is influence by other rows as the kernel used is 2D instead of 1D, so I create a 1D kernel to ensure it is accurate. In order to simulate gaussian kernel I used the sum of all row values from the 2D kernel where sigma is equal to 2 as shown below.

Code:

```
img_filter2 = imfilter(img_Origin_double,[0.1525 0.2218 0.2514 0.2218 0.1525], 'replicate');
```

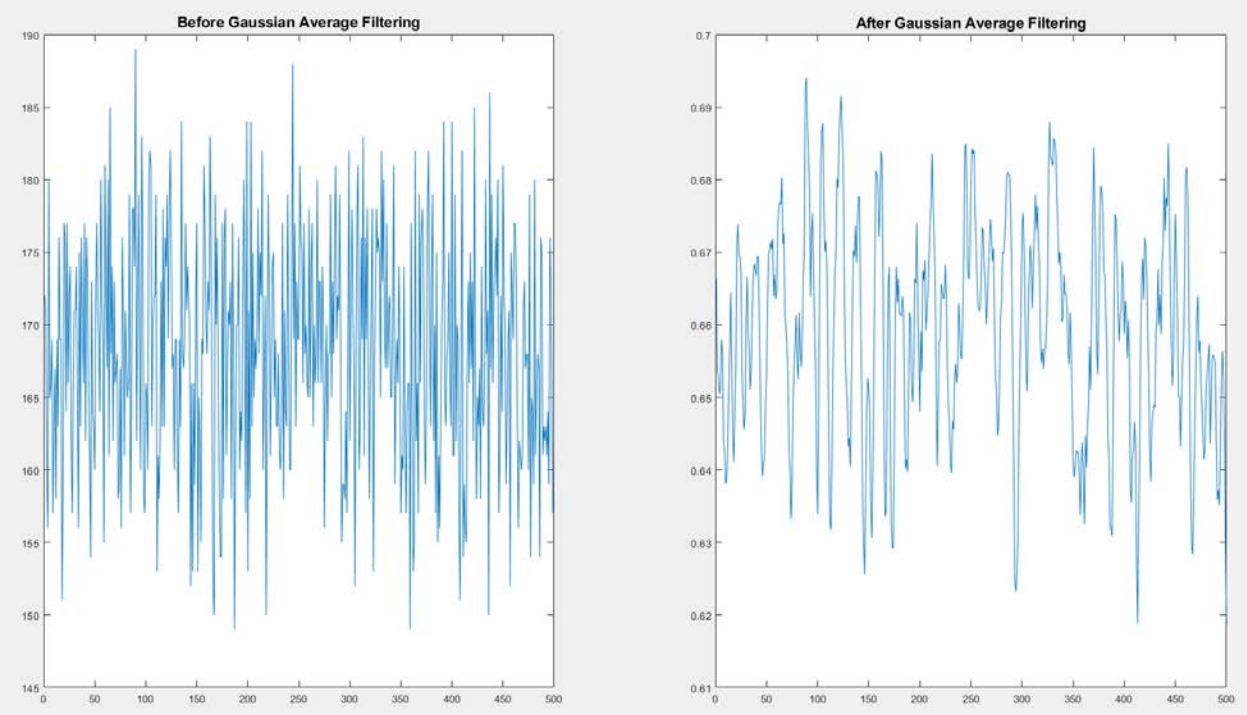
- Create 1D filter/kernel

```
plot(img_Origin(1,:));
ussian Average Filtering
plot(img_filter2(1,:));
```

- Display the result

Output:

Sigma = 2



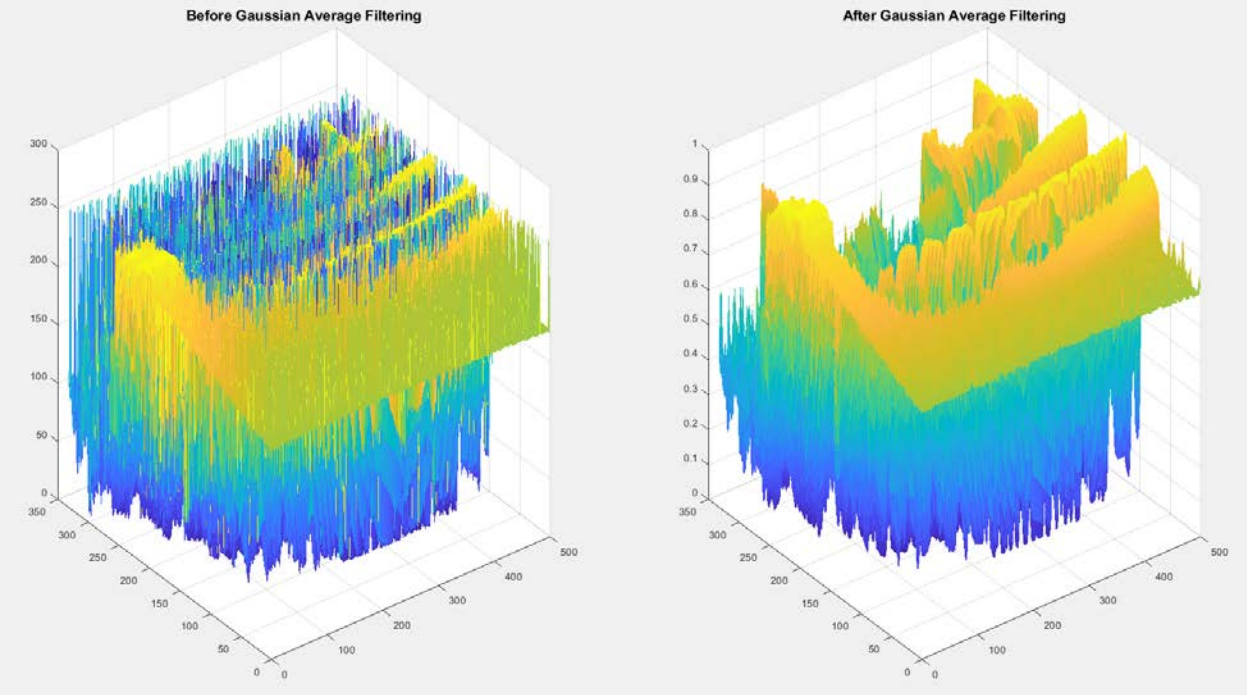
The details loss is less because lack of influence from the other rows. This is an accurate representation of how gaussian filter work in 1D kernel.

d. View speckle noise image (‘ntusp.jpg’)

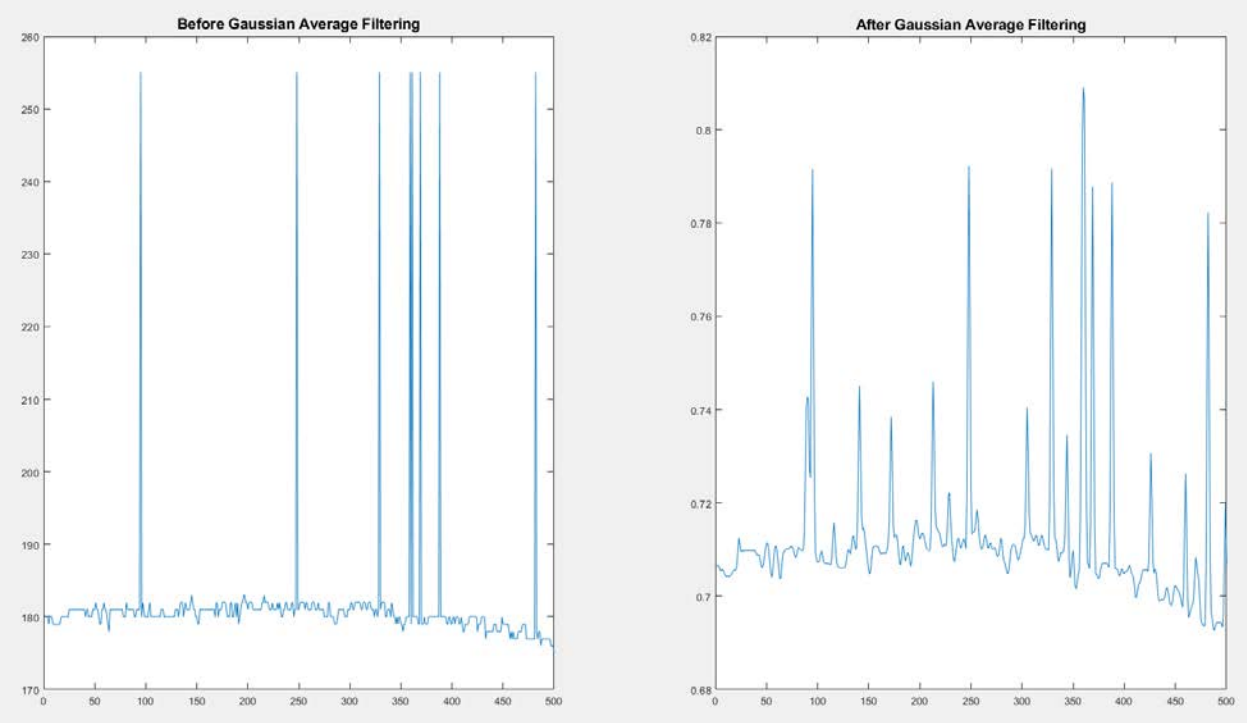
This section will only show the output since the code is relatively the same as previous one.

Sigma = 1

Mesh

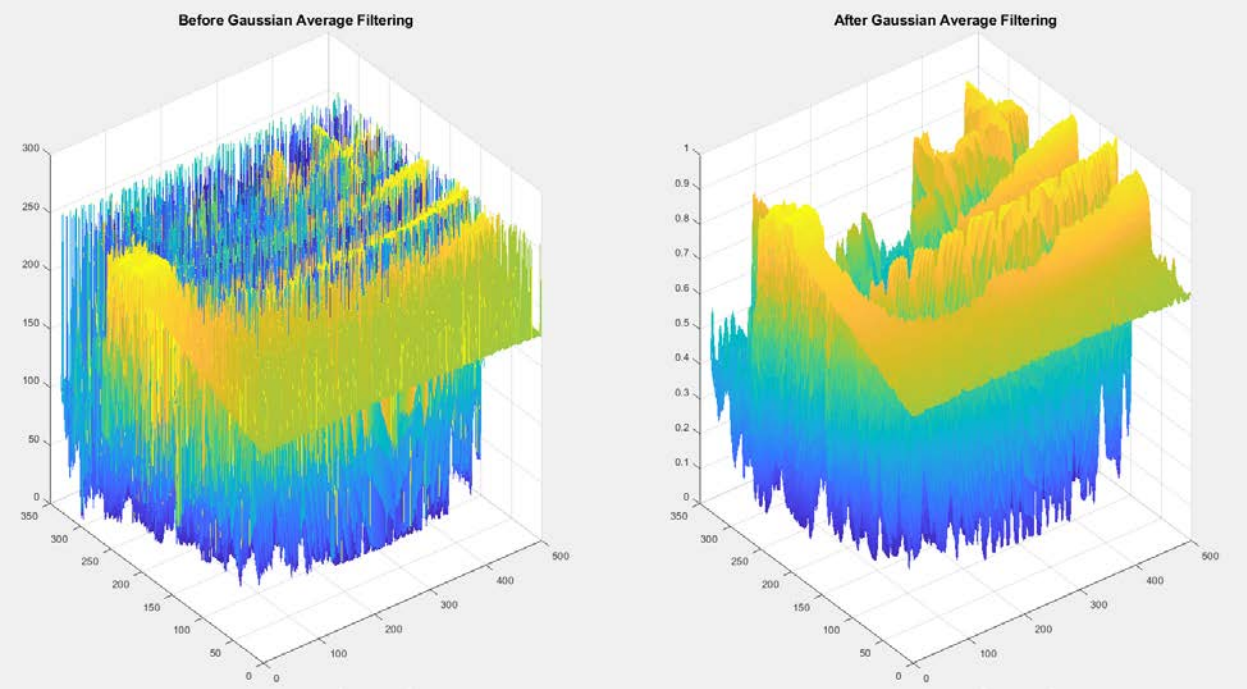


Cross Section

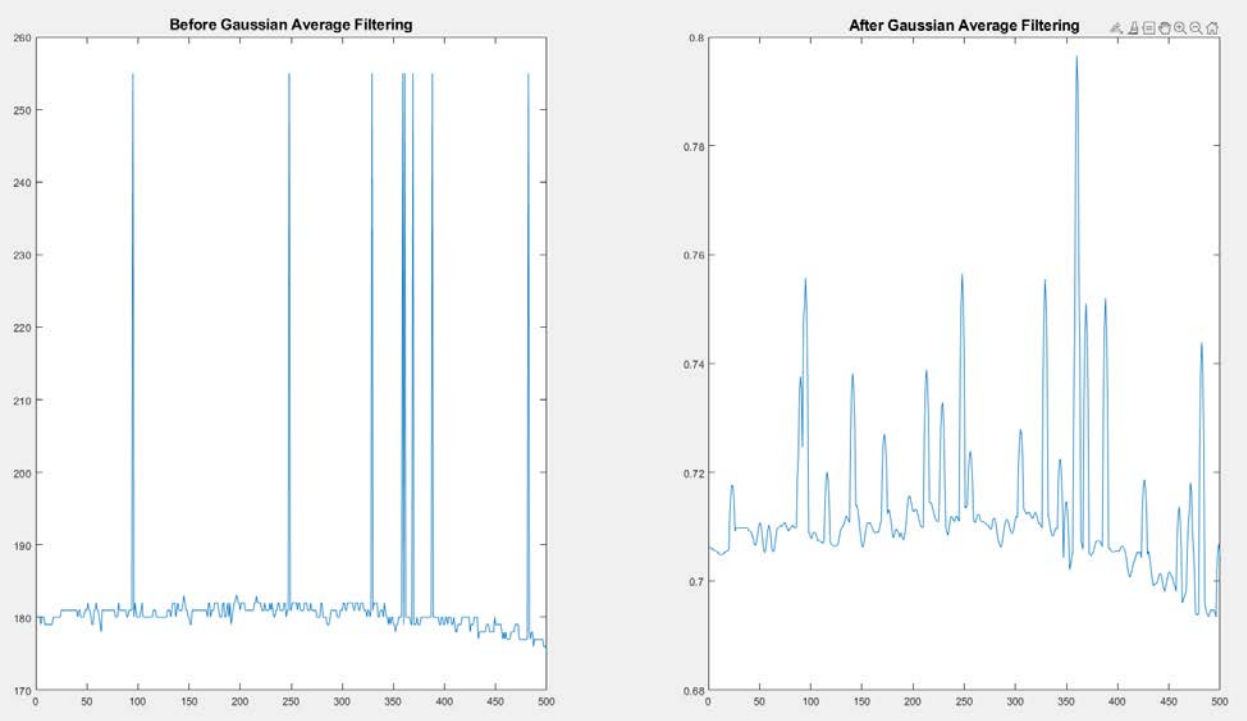


Sigma = 2

Mesh



Cross Section



Speckle noise is a challenge for gaussian average filter. Base on observation, the spike still exists. Although higher value sigma still performs better when comes to suppress the noise. The reason why the spike still exist is because the deviation of change from the surrounding intensity is far too great. As a result, the average would be around the middle between them.

Another scenario that would totally fail the filtering process is when the noise distributed in a chunk/block. This filter may need to increase kernel dimension to work better or may require other more effective solution(s).

e. Which type of noise (gaussian or speckle) is the filter better at handling.

Gaussian Average filter is better handling with gaussian noise as compare to speckle noise because the deviation of change in intensity is less significant.

2.4 Median Filtering

The code is mostly the same as Gaussian Average Filtering except apply median filtering code. Therefore, those section without code which mean it is the same as gaussian average filtering.

b. View gaussian noise image ('ntugn.jpg')

Output:

c. Apply filter image and display the image

Code:

```
img_med = medfilt2(img_Origin, [dimension,dimension]);
```

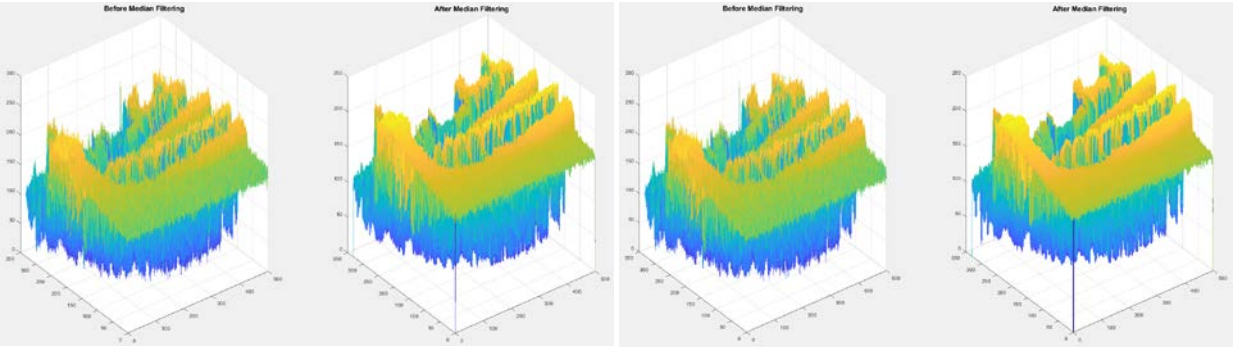
- Apply Median filter function with given dimension 3x3 and 5x5

```
imshow(img_med);
```

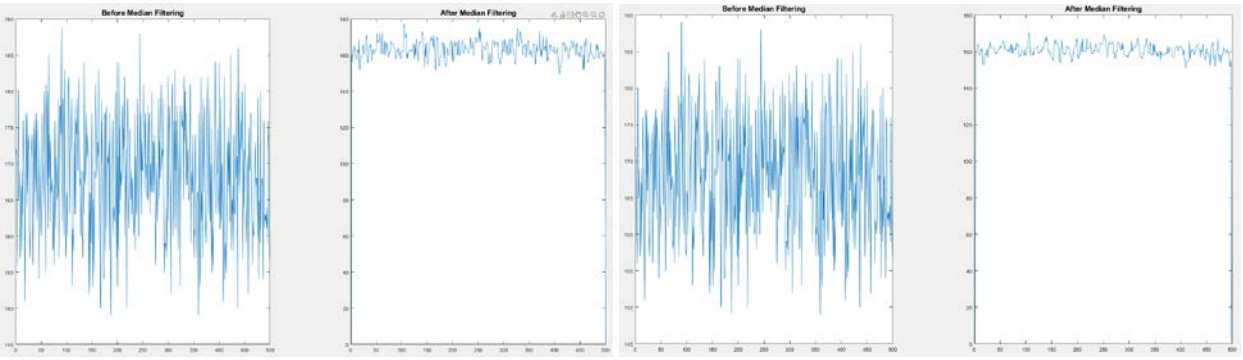
- Show the result of after median filter.

Output:

Mesh



Cross Section

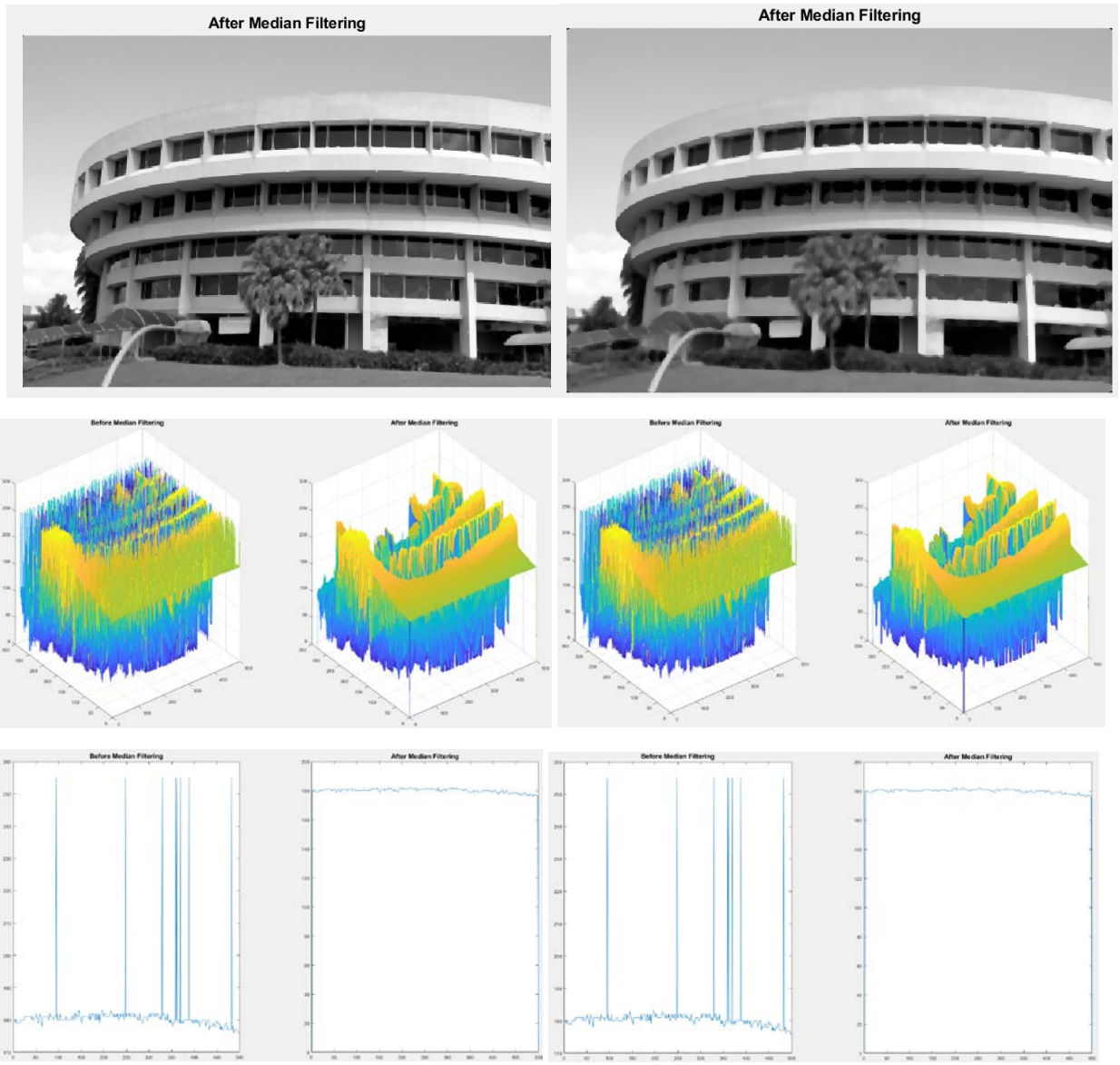


Left is 3x3 median filter, Right is 5x5 median filter

Median filtering manages to suppress gaussian noise equally well as compare to gaussian average filtering. However, median filter seems to loss details more easily when increase in filter's dimension as compare with gaussian average filter with the same dimension size.

d. View speckle noise image ('ntusp.jpg')

Output:



Left is 3x3 median filter, Right is 5x5 median filter

Base on result, median filter are robust to speckle noise, with low dimension the filter can handle the noise well as compare to gaussian average filter.

e. Which type of noise (gaussian or speckle) is the filter better at handling.

Median filter is effective in handling both type of noises, it seems to handle speckle noise better.

f. How does gaussian filtering compare with median filtering with different type of noise.

Gaussian filtering handle gaussian noise better while median filtering handle speckle noise better.

The details loss in gaussian filtering is less than median filtering, the main reason was gaussian filter give every pixel a weight factor. Unlike median filter where no weight was considered to any other pixels. Speckle noise are considered outlier and unwanted pixel. Therefore, no weight should be considered, and median filter is best fit for such case.

g. The tradeoffs of gaussian filtering and median filtering.

The tradeoffs are choosing either noise removal or information loss. Gaussian filter has less information loss, but it does not remove noise fully as each noise has a weight factor. On the other hand, median filter suppress noise well where these pixels hold no weight value but having more information loss.

2.5 Suppressing Noise Interference Patterns

a. Display ‘pckint.jpg’

Code:

```
URL = 'assets\pckint.jpg';
```

- Get directory path and filename

```
Pc = imread(URL);
```

- Read the image

```
assignin('base','img_Origin',Pc)
```

- Store the image into workspace

```
img_Origin = evalin('base','img_Origin');
```

- Retrieve image from workspace

```
imshow(img_Origin);
```

- Display the image

Output:

Notice prominent diagonal lines parallel pattern that reduce the quality of image.

b. Transform image with Fast Forward Transform [fft2], compute and display power spectrum in 'fftshift'

Code:

```
% Convert spatial domain to spectrum domain using FFT
img_FFT = fft2(img_Origin);

% Get the absolute value of FFT
S = abs(img_FFT);

% Adjust power spectrum to power of 0.1
fft_power = S.^0.1;
```

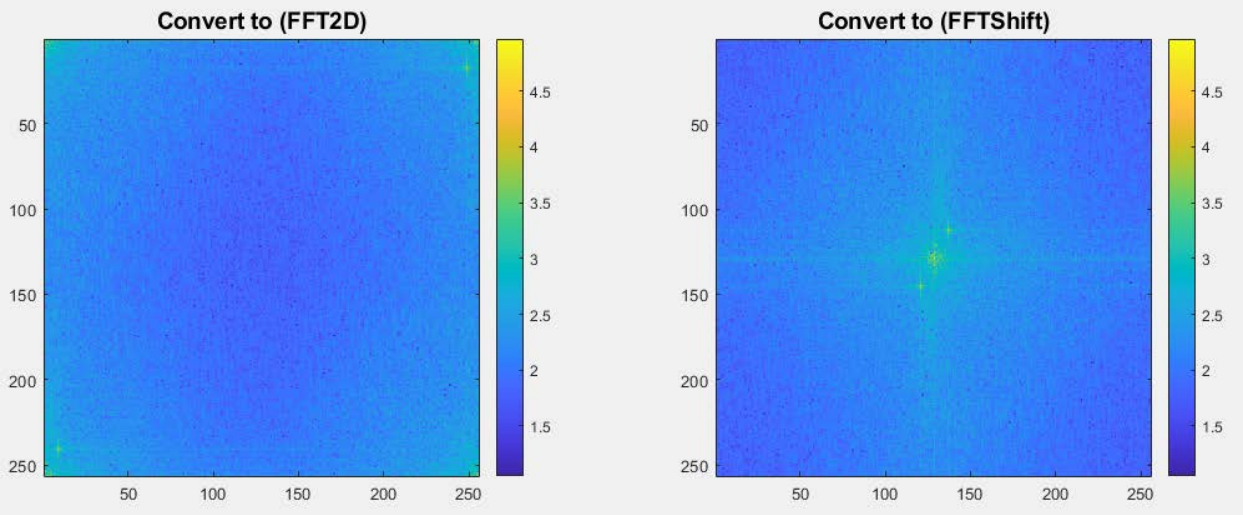
```
imagesc(fft_power) , colorbar, axis image ;
```

- Display the image with color bar and correct aspect ratio with image with 'axis image'

```
imagesc(fftshift(fft_power)), colorbar, axis image;
```

- Display the image in 'fftshift' where frequency position is relocated (low to high, high to low)

Output:



Notice that peak frequency is symmetric in the power spectrum domain except for DC due to the nature of complex number computation.

c. Display and Mark the peaks coordinate in power spectrum without 'fftshift'

In order to mark, instead of 'ginput' which is hard to identify correctly, I use mesh to mark the coordinate and hard code them into the program.

Code:

```
% Duplicate a copy for mesh display
fft_power_copy = fft_power;
subplot(1,1,1), mesh(fft_power_copy);
```

- Show mesh view of the power spectrum to identify the peaks

```
% Offset 2 of the target coordinates such that the surrounding 5by5 will be set zeros is plus minus 2 offset
offset = 2;
```

- Set offset for 5x5 zeros

```
% 5 Peaks coordinates (1,1)(9, 241)(249,17)(255,1)(255,256)
x1 = 1; % DC value
y1 = 1;
x2 = 9;
y2 = 241;
x3 = 249;
y3 = 17;
x4 = 255;
y4 = 1;
x5 = 255;
y5 = 256;
```

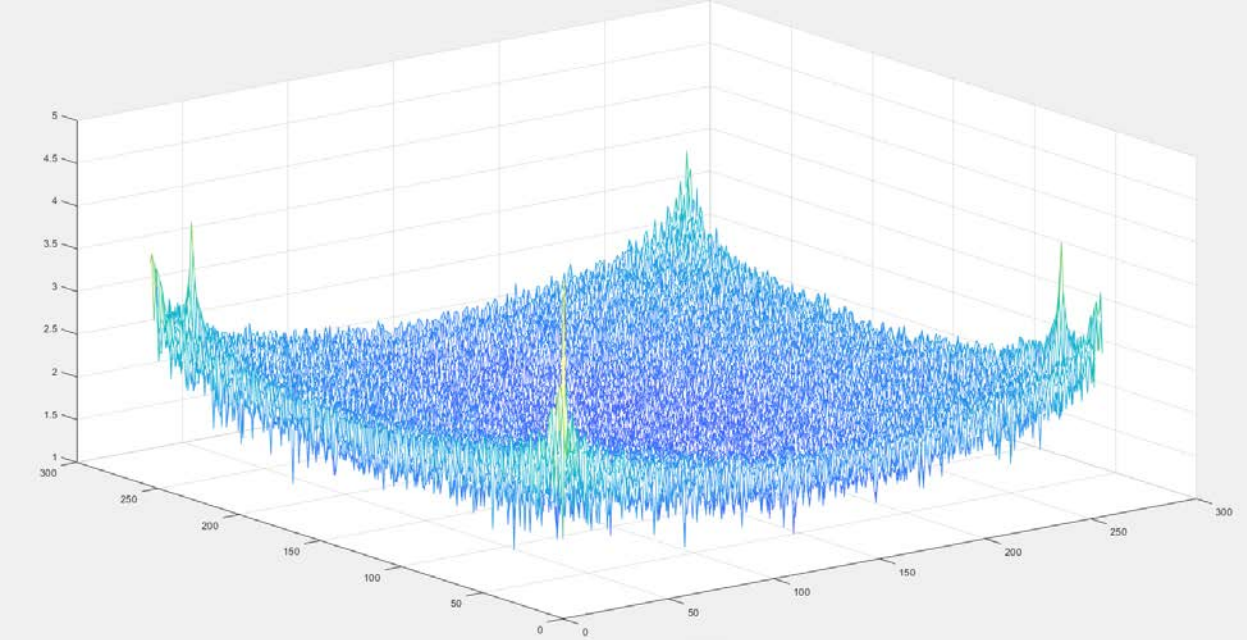
- Hard code the peaks values (Note: this is not the most efficient way to do)

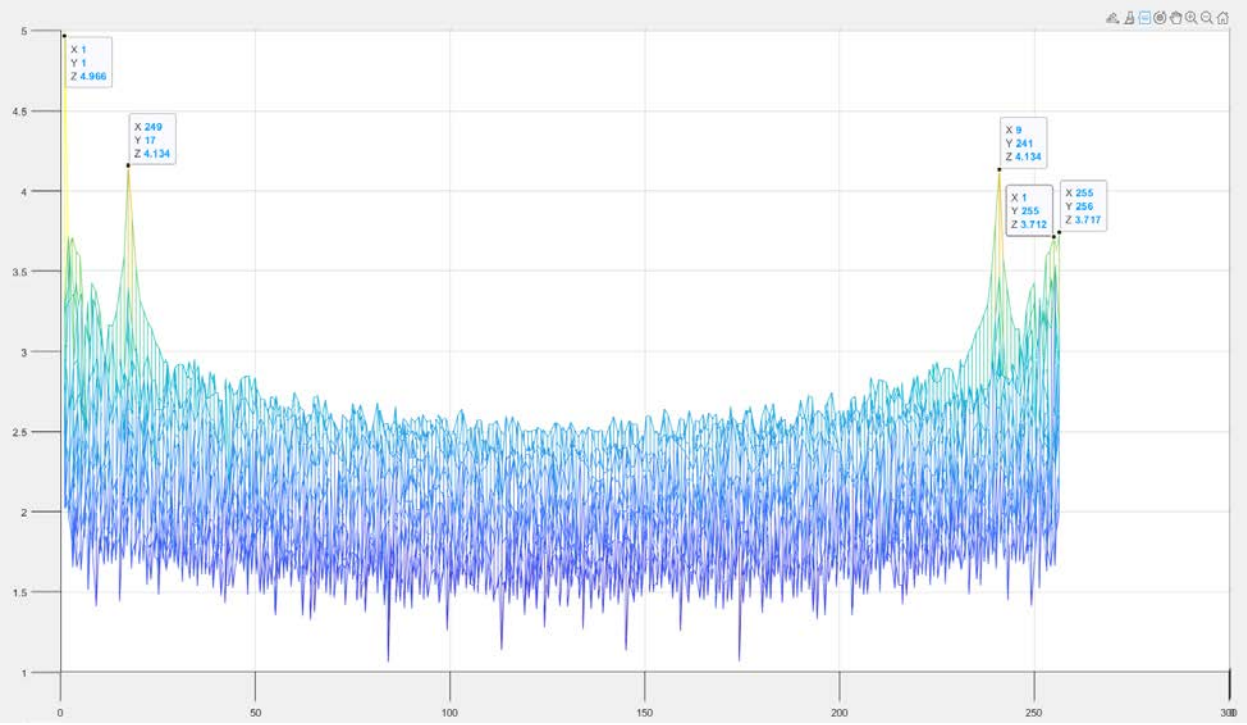
```
% Increase to min dimension of threshold (zero or neg value)
x1min = max(x1-offset,1);
x1max = max(x1+offset,1);
y1min = max(y1-offset,1);
y1max = max(y1+offset,1);
x2min = max(x2-offset,1);
x2max = max(x2+offset,1);
y2min = max(y2-offset,1);
y2max = max(y2+offset,1);
x3min = max(x3-offset,1);
x3max = max(x3+offset,1);
y3min = max(y3-offset,1);
y3max = max(y3+offset,1);
x4min = max(x4-offset,1);
x4max = max(x4+offset,1);
y4min = max(y4-offset,1);
y4max = max(y4+offset,1);
x5min = max(x5-offset,1);
x5max = max(x5+offset,1);
y5min = max(y5-offset,1);
y5max = max(y5+offset,1);
```

```
% Reduce to max dimension of threshold
x1min = min(col,x1min);
x1max = min(col,x1max);
y1min = min(row,y1min);
y1max = min(row,y1max);
x2min = min(col,x2min);
x2max = min(col,x2max);
y2min = min(row,y2min);
y2max = min(row,y2max);
x3min = min(col,x3min);
x3max = min(col,x3max);
y3min = min(row,y3min);
y3max = min(row,y3max);
x4min = min(col,x4min);
x4max = min(col,x4max);
y4min = min(row,y4min);
y4max = min(row,y4max);
x5min = min(col,x5min);
x5max = min(col,x5max);
y5min = min(row,y5min);
y5max = min(row,y5max);
```

- Adjust out of dimension values such that it is max or min values of the image dimension

Output:





Rotate mesh to side view to identify the peaks.

d. Set zero around the coordinate in 5x5 corresponding to the above chosen peaks.

Code:

```
% Apply the coordinates and set zeros accordingly
%fft_suppress(ylmin:ylmax,xlmin:xlmax) = 0; % DC value not
%used
fft_suppress(y2min:y2max,x2min:x2max) = 0;
fft_suppress(y3min:y3max,x3min:x3max) = 0;
fft_suppress(y4min:y4max,x4min:x4max) = 0;
fft_suppress(y5min:y5max,x5min:x5max) = 0;
fft_suppress(y6min:y6max,x6min:x6max) = 0;
fft_suppress(y7min:y7max,x7min:x7max) = 0;
```

- DC value is not use as it would unnecessarily reduce the quality of the image in terms of brightness level, in this case the image is dimmed if DC is considered.

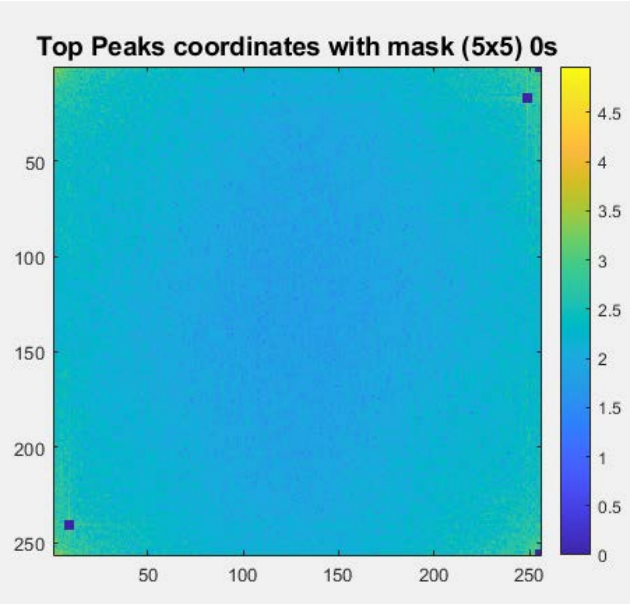
```
% Get the absolute value of FFT
newS = abs(fft_suppress);

% Adjust power spectrum to power of 0.1
fft_power_suppress = newS.^0.1;
```

- Compute the power spectrum to be used for display

```
imagesc(fft_power_suppress), colorbar, axis image;
```

- Display the new power spectrum of 5x5 zeros masks

Output:



e. Compute IFFT2 and display the resultant image

Code:

```
inverse_img = ifft2(fft_suppress);
```

- Inverse the image using IFFT2

```
inverse_img_real = uint8(real(inverse_img));
```

```
imshow(inverse_img_real);
```

- Display the resultant image

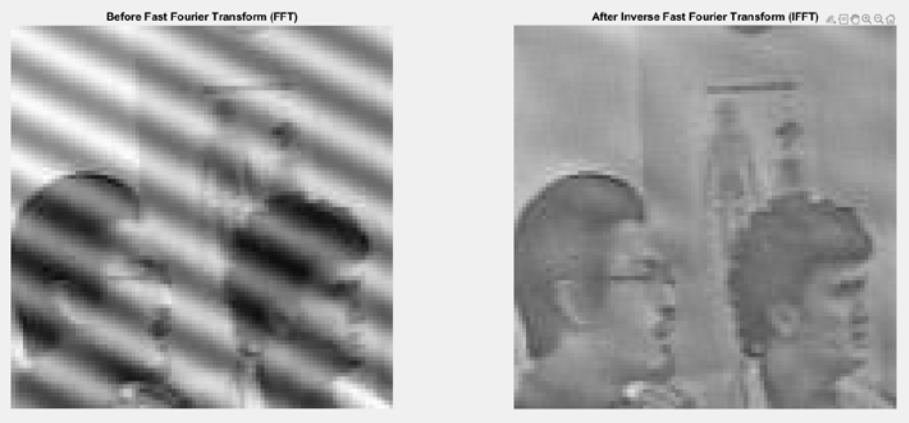
Output:

```
Highest Intensity Value: 4.9664
Inverse Image Single Sample Output:
   1.9688e+02 - 1.6998e+01i

Inverse Image Single Sample Output (Real Number Component):
  196.8843

Inverse Image Single Sample Output (Normalize to uint8 (int between 0 to 255):
   197
```

The sample of conversion back from IFFT to image process and measurement of highest intensity value for references purpose

- The result has shown that the operation has effective remove the pattern in complex features area where higher frequency has been located. While low frequency the pattern is still visible.

- Another way to eliminate the parallel pattern better is by segmentation of image into n number equal dimension pieces and merge back into the same image.



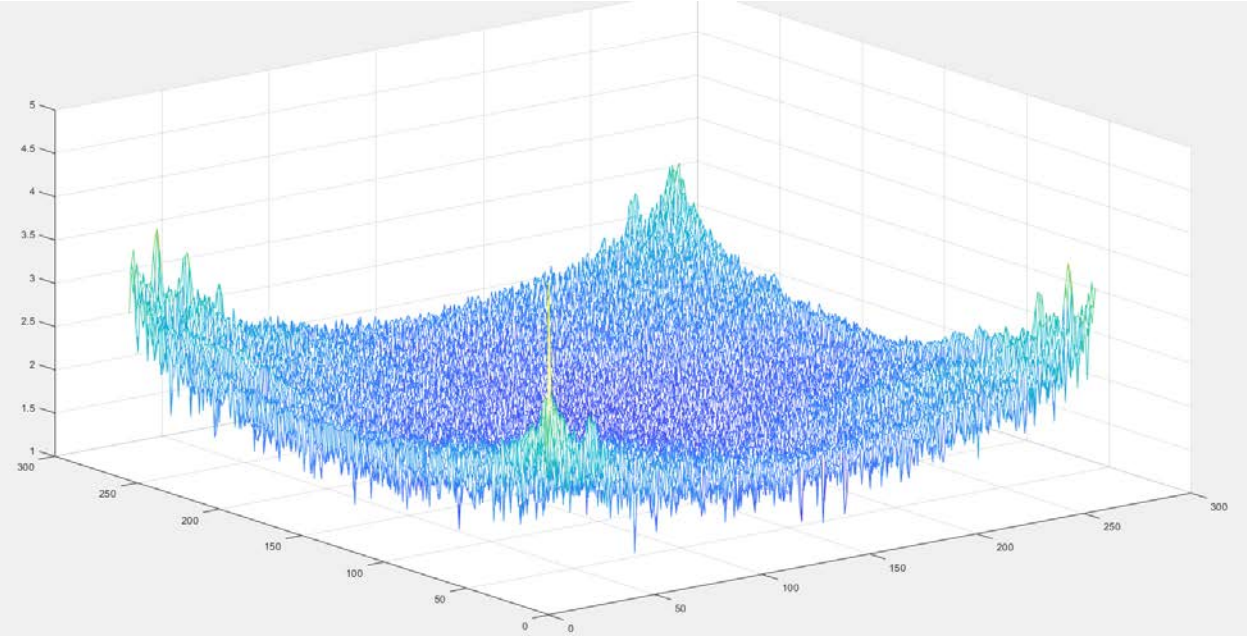
Notice the segmentation result is better, this can show the potential improvement of the image.

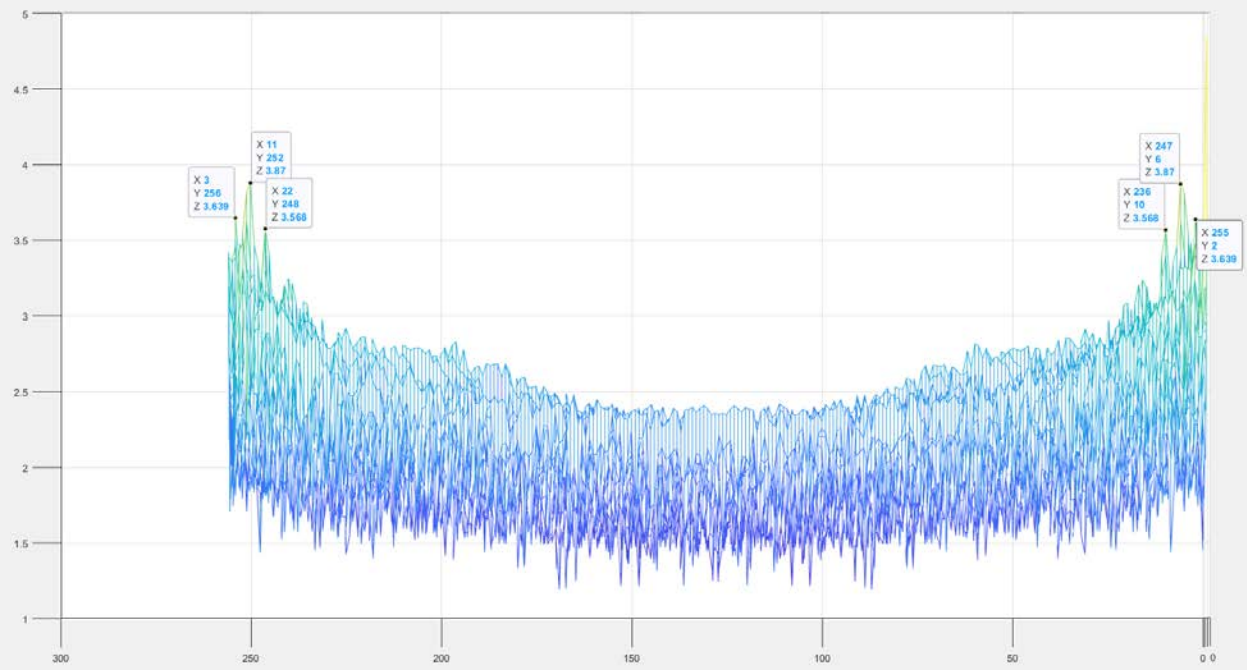
To further improve the image, histogram equalization is used since the min-max intensity value is not between 0 to 255 as shown below. From here we can see the quality of image is much better compare to origin of the image.

```
Minimum Intensity Value:
   63

Maximum Intensity Value:
   224
```

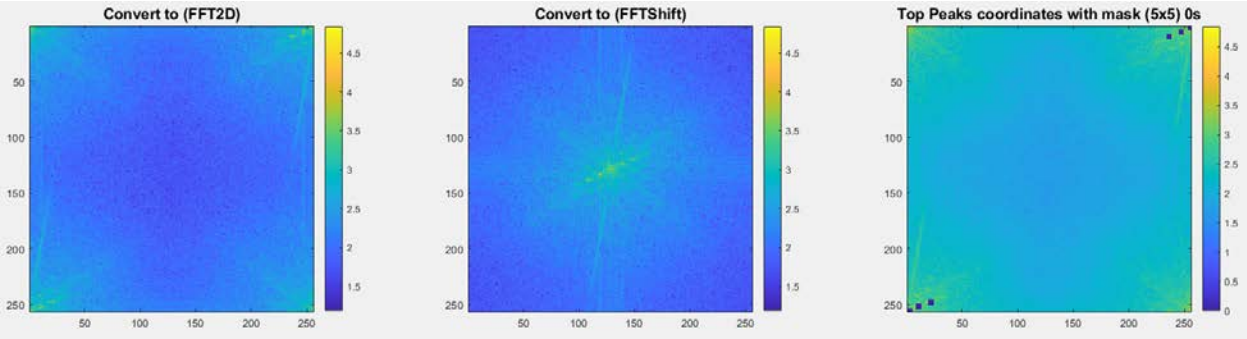
f. Attempt to free primate fence from ‘primatecaged.jpg’

Similarly, the code is the same approach except when choosing the peaks coordinates.



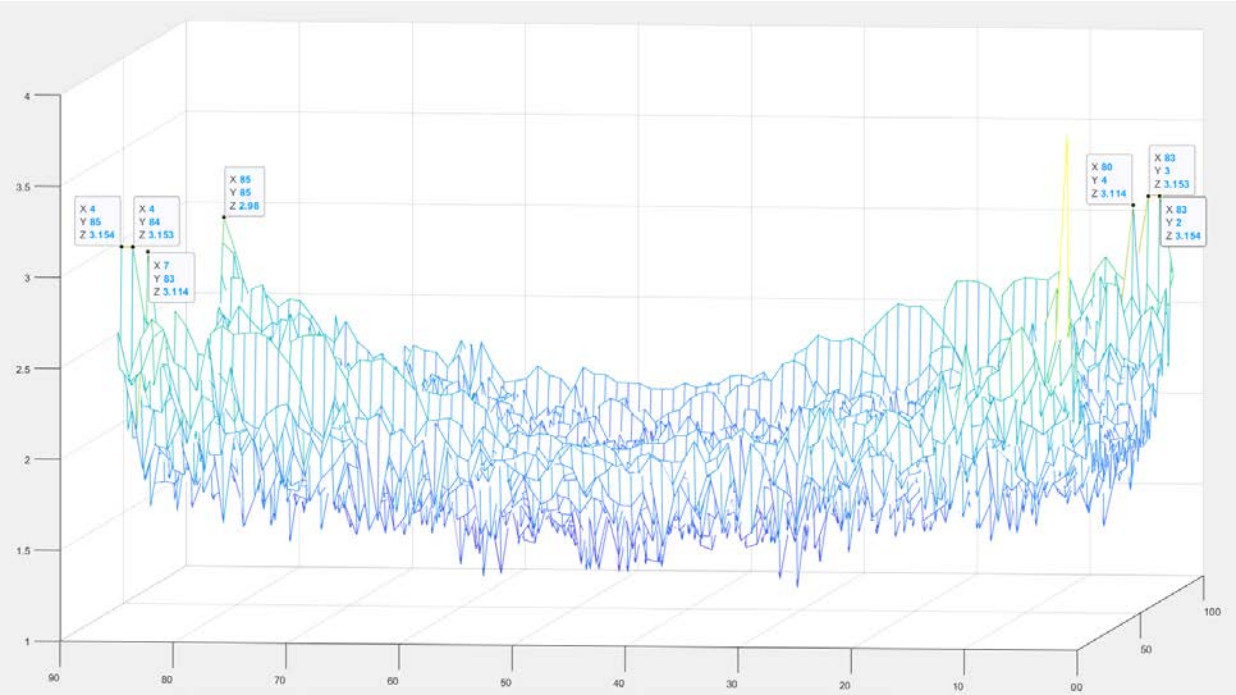


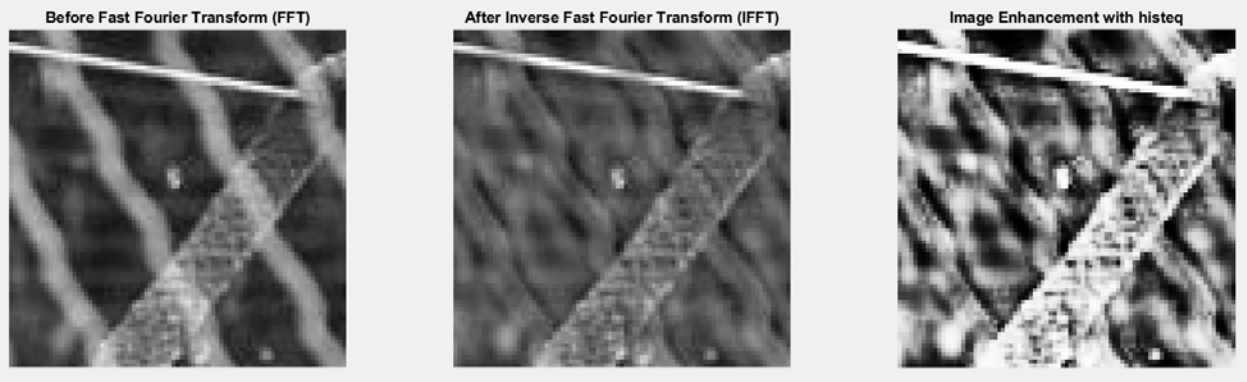
In this case, 6 peaks were selected.

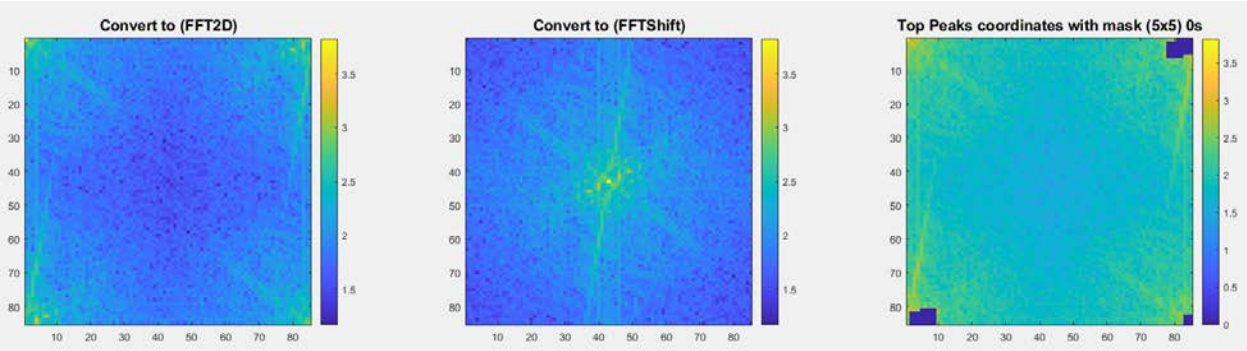


In unparallel pattern, choosing peaks and set the surrounding to zero may not be a viable solution as the unparallel pattern consist of different range of frequency. How about trying segmentation approach?

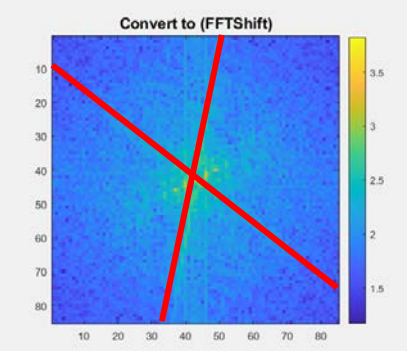
Segmentation Approach







Notice segmentation approach does not really remove the unparallel patterns. Since the unparallel pattern seems to be a composite of many different frequency range as seen in FFTShift, this can be observe as there is quite a few prominent lines. Other possible way to improve is by finding the equation of the line in the power spectrum and apply 5x5 zeros for all the points along the prominent line as shown below. This part is not tested as it is out of my knowledge and time invest may not be viable.



To conclude, getting the right frequency to filter away the type of pattern require a lot of ‘dirty’ works and experiences.

2.6 Undoing Perspective Distortion of Planar Surface

a. Display image

Given the image consist of a slanted view of a book which is 210mm x 297mm

Code:

```
URL = 'assets\book.jpg';
```

- Get the directory and filename

```
Pc = imread(URL);
```

- Read the image with provided string path

```
assignin('base','img_Origin',Pc)
```

- Store image into workspace

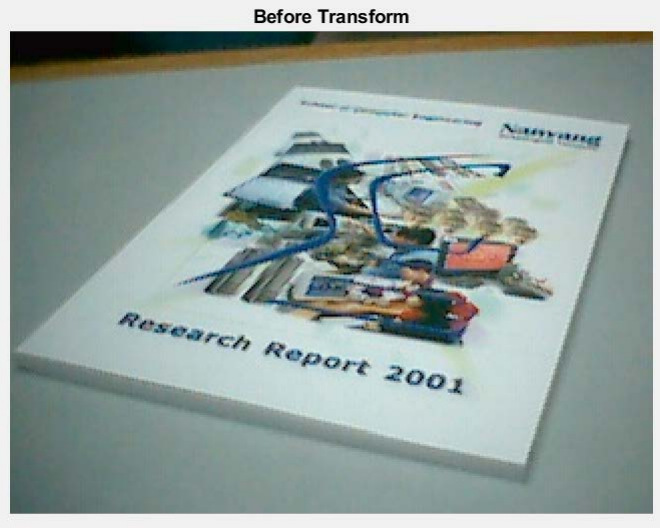
```
img_Origin = evalin('base','img_Origin');
```

- Retrieve image from workspace

```
imshow(img_Origin);
```

- Display the image.

Output:



The image consists of a A4 book as mentioned in the instruction.

b. Using ginput to locate 4 corners of the book

Code:

```
[X, Y] = ginput(4);
```

- 4 refer the number of inputs

Input:

Output:

```
X and Y Coordinate:
  143.8022   28.0389
  308.7866   46.9798
  256.9486  215.9517
    5.7336  158.1324
```

Note: The order of measure corner matters depends on how 4 points of x and y coordinates is defined.

In this case, define in top left (0,0), top right (210,0), bottom right (210,297), bottom left (0,297) coordinates

```
% Specify the coordinates for the output
x = [0; 210; 210; 0];
y = [0; 0; 297; 297];
```

c. Setup the matrices for projective transformation base on given equation

Code:

```
% Formulate the formulate
v = zeros(8, 1);
A = zeros(8, 8);
for i = 1:4
    A(2*i-1 : 2*i, :) = [X(i) Y(i) 1 0 0 0 -x(i)*X(i) -x(i)*Y(i); 0 0 0 X(i) Y(i) 1 -y(i)*X(i) -y(i)*Y(i)];
    v(2*i-1 : 2*i) = [x(i), y(i)];
end

u = A\v;

% Calculate U
U = reshape([u;1],3,3)';

% Show U
disp('U 2d-array values:')
disp(U);

% Calculate w
w = U*[X'; Y'; ones(1,4)];
w = w./ (ones(3,1) * w(3,:));

% Show w
disp('w 2d-array values:')
disp(w);
```

Output:

```
Transformation Matrix Values:
   1.0e+04 *

    0.0145    0.0029    0.0001         0         0         0         0         0
         0         0         0    0.0145    0.0029    0.0001         0         0
    0.0307    0.0047    0.0001         0         0         0   -6.4426   -0.9970
         0         0         0    0.0307    0.0047    0.0001         0         0
    0.0255    0.0215    0.0001         0         0         0   -5.3645   -4.5141
         0         0         0    0.0255    0.0215    0.0001   -7.5870   -6.3842
    0.0006    0.0159    0.0001         0         0         0         0         0
         0         0         0    0.0006    0.0159    0.0001   -0.1703   -4.7113
```

```
U 2d-array values:
    1.5298    1.6416 -269.1757
   -0.4408    3.8717  -48.5936
    0.0002    0.0056    1.0000

w 2d-array values:
         0  210.0000  210.0000         0
   -0.0000         0  297.0000  297.0000
    1.0000    1.0000    1.0000    1.0000
```

The left image is the transformation matrix values. The right image ‘w’ is the transformed coordinates.

d. Warp the image

Code:

```
% Transform the image
T = maketform('projective', U');
img_transform = imtransform(img_Origin, T, 'XData', [0 col], 'YData', [0 row]);
```

- Transform image by matrix transformation

e. Display the transform image

Code:

```
imshow(img_transform) , axis normal;
```

- Display image and set axis normal or else it will show in square aspect ratio

Output:

From the image above it manages to transform into front view of the book. The image proportion is well done especially the bottom half of the image as it has more details in closer view. Whereas on further side of the book is very blurry and distorted. This result is expected since the actual image is not visible as well.

The resultant image quality able to retain most of the detail from the original image is impressive.

-end of report-