

**CZ4003 Computer Vision**

**Laboratory 1**

**Point Processing + Spatial Filtering + Frequency Filtering + Imaging Geometry**

Tan Xian Ren Clement (U1620710G)

Professor Qian Kemao

**Contents**

[**1. Introduction** 3](#_Toc528683388)

[**1.1 Objectives** 3](#_Toc528683389)

[**2. Experiments** 3](#_Toc528683390)

[**2.1 Contrast Stretching** 3](#_Toc528683391)

[**2.2 Histogram Equalization** 7](#_Toc528683392)

[**2.3** **Linear Spatial Filtering** 9](#_Toc528683393)

[**2.4 Median Filtering** 15](#_Toc528683394)

[**2.5 Suppressing Noise Interference Patterns** 16](#_Toc528683395)

[**2.6 Undoing Perspective Distortion of Planar Surface** 23](#_Toc528683396)

# **1. Introduction**

## **1.1 Objectives**

This laboratory aims to introduce image processing in MATLAB context. In this laboratory you will:

a. Become familiar with the MATLAB and Image Processing Toolbox software package.  
b. Experiment with the point processing operations of contrast stretching and histogram equalization.   
c. Evaluate how different Gaussian and median filters are suitable for noise removal.   
d. Become familiar with the frequency domain operations e. Understand imaging geometry.

# **2. Experiments**

## **2.1 Contrast Stretching**

Obtain the image mrt-train.jpg from the edveNTUre website under Course Documents/ Laboratory Material/Images.

1. Input the image into a MATLAB matrix variable by executing:

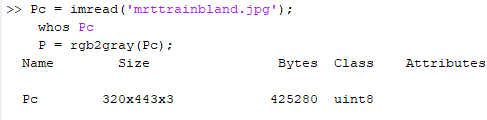
>> Pc = imread(‘mrt-train.jpg’);

>> whos Pc

The whos command is to show whether the image is read as an RGB or gray-scale image. Notice that this is a 320x443x3 uint8 matrix indicating a colour image with byte values. You will need to convert this into a grayscale image by:

>> P = rgb2gray(Pc);

**Result and Comments:**

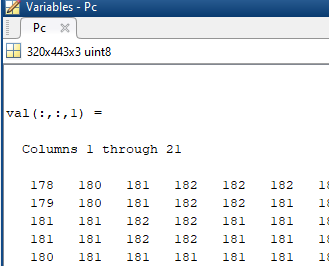
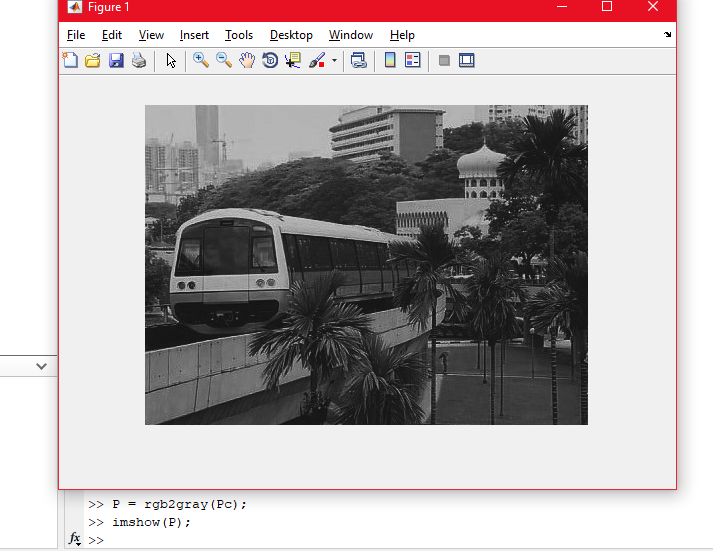


After downloading the image and using the above code, I read the picture into a variable named ‘Pc’. The ‘whos’ command showed the details of the input image.

1. View this image using imshow. Notice that the image has poor contrast. Your initial task will be to investigate different methods for improving the image appearance using point processing. In point processing, the image transformation is carried for each pixel independently of neighboring pixels.

**Result and Comments:**

I then, converted it to gray scale as it was in RGB. This can be determined by looking at the matrix size of the image. And showed the image as shown below.

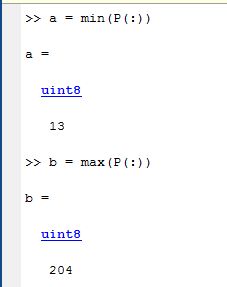
1. Check the minimum and maximum intensities present in the image:

>> min(P(:))

>> max(P(:))

Contrast stretching involves linearly scaling the gray levels such the smallest intensity present in the image maps to 0, and the largest intensity maps to 255.

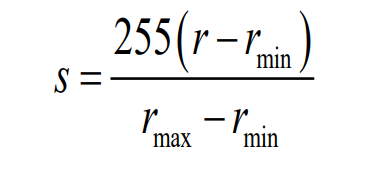
**Result and Comments:**



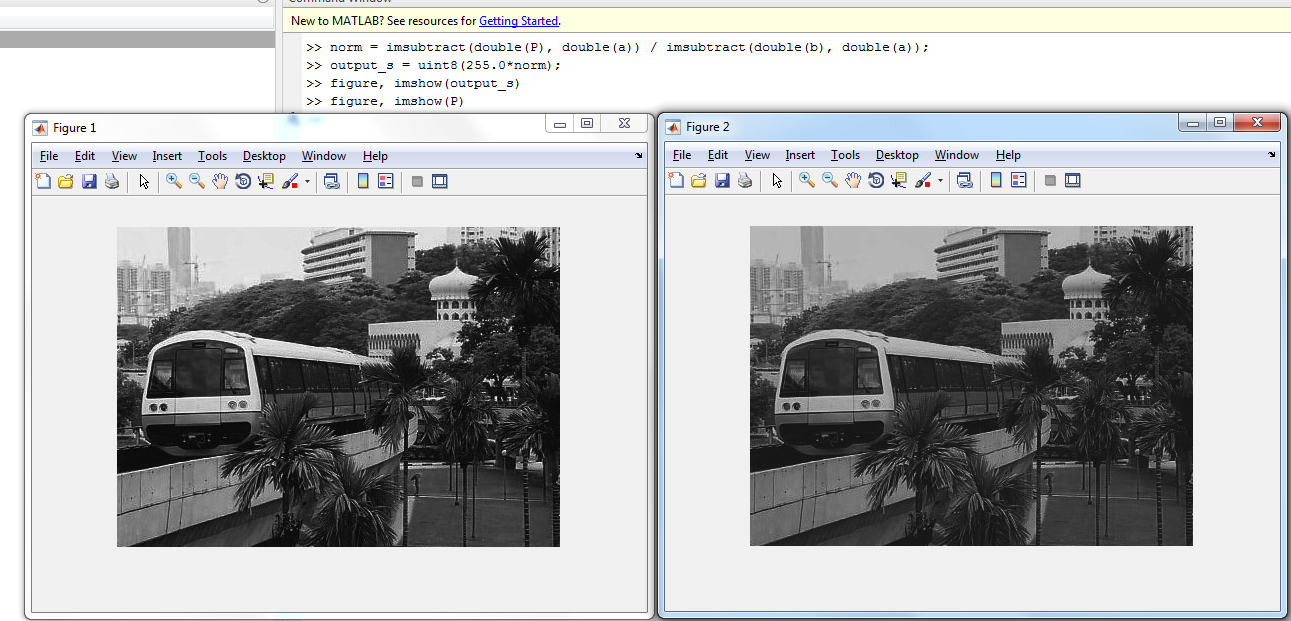
I checked the min and max intensity of the picture and assigned it to variables a and b respectively.

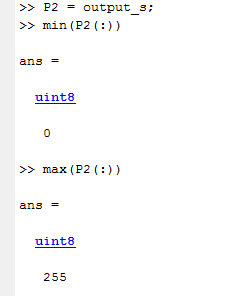
1. Next, write two lines of MATLAB code to do contrast stretching. Hint: This involves a subtraction operation followed by multiplication operation (note that for images which contain uint8 elements, you will need to use imadd, imsubtract, etc. for the arithmetic instead of the usual operators; alternatively you can convert to a double-valued matrix first using double, but you will need to convert back via uint8 prior to using imshow — see below). Check to see if your final image P2 has the correct minimum and maximum intensities of 0 and 255 respectively by using the min and max commands again.

**Result and Comments:**The contrast stretching formula is given as:



With this, I can formulate the formula based on the requirements that it should only contain two lines of MatLab code. Using the variables ‘a’ and ‘b’ which represents min and max respectively, I calculated the normalization equation on the first sentence, followed by the output S. The **uint8 values must be changed to the double datatype** before the equation could work. Before showing the figures, the output must be converted back to uint8.

Figure 1 shows the output after contrast stretching is applied, while Figure 2 is the original image in gray-scale.



The output of the minimum and maximum values of the intensities changed and ranged from 0 to 255 now.

1. Finally, redisplay the image by

>> imshow(P2);

Note again that if you choose to convert your byte images to doubles first, you will need to use the uint8 command is necessary to convert the P2 double values back to byte values. Otherwise, imshow detects P2 to be double and assumes that the intensity range is between 0.0 and 1.0.

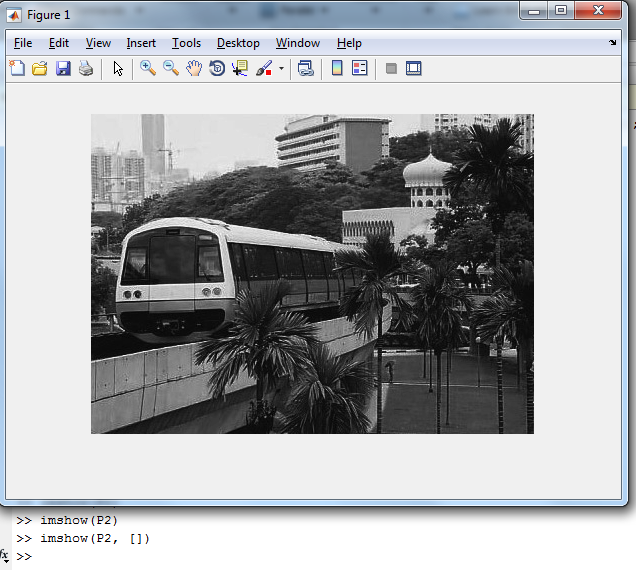
An alternative is to use the imshow as such:

>> imshow(P2,[]);

**Result and Comments:**

This does automatic contrast stretching of the display but does not affect the input matrix. This allows you to ignore whether you are displaying byte or double-valued images.

Redisplaying the image again, we can tell that the picture has been beautified. Both codes ran below can output the figure. The second code will still allow the image to be displayed ignoring whether the image is in uint8 or double-valued images.



## **2.2 Histogram Equalization**

1. Display the image intensity histogram of P using 10 bins by

>> imhist(P,10);

Next display a histogram with 256 bins. What are the differences?

**Result and Comments:**  
Displayed the image P with 10 bins and 256 bins respectively by the following code as shown below.

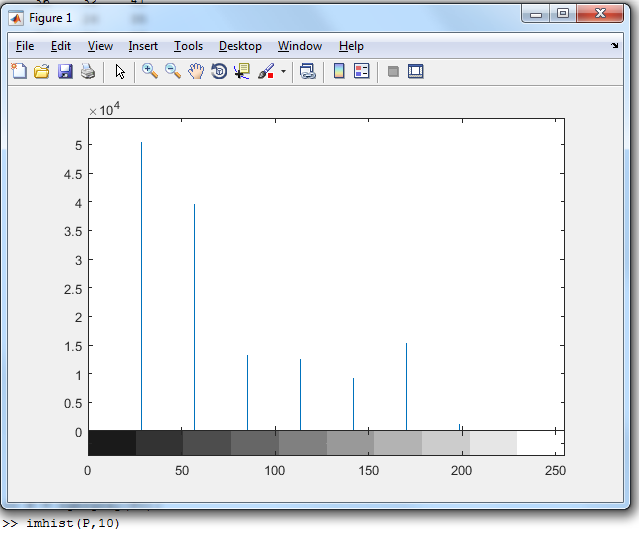


Image P displayed on a histogram with 10 bins

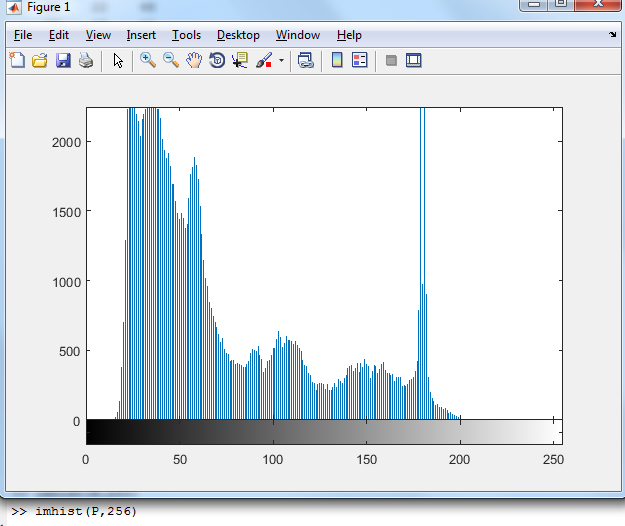


Image P displayed on a histogram with 256 bins

To first understand why there is a difference between them, we need to understand what bins are. Bins represents the number of gray levels to be shown in a histogram. As there are only 10 bins, the total number of pixels given by the entire picture which is 320 x 443 will be spread across these 10 bins. The no. of pixel scale in the first figure are 10^4 and only up to about 2500 in the second figure. The help function in Matlab shows imhist function states the following:

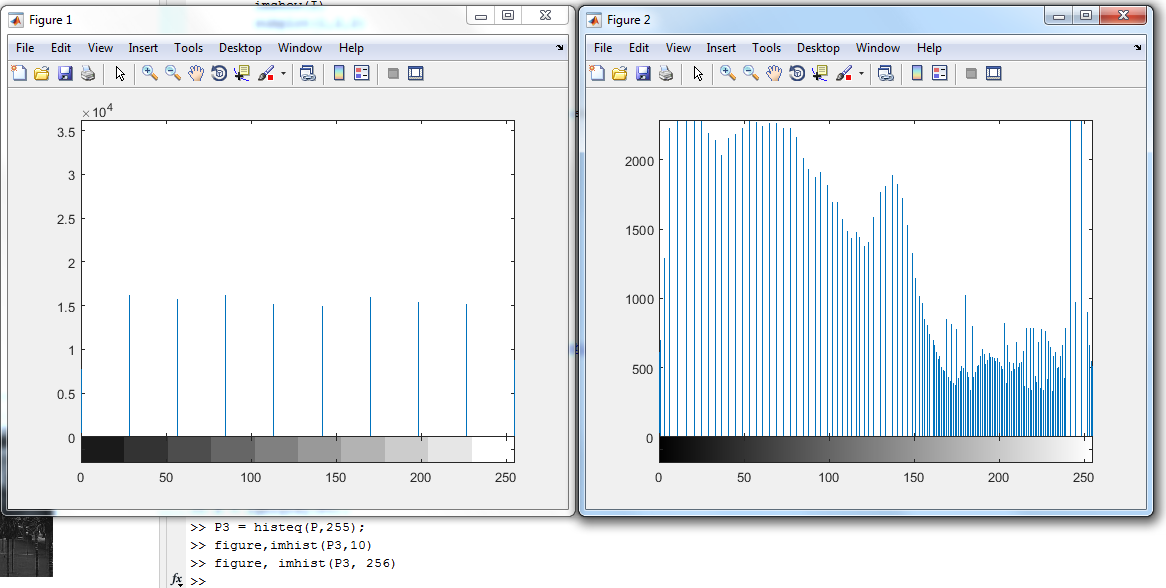
“imhist(I,N) displays a histogram with N bins for the intensity image I above a grayscale colorbar of length N. If I is a binary image, then N can only be 2.”

This shows that it is representing the image using a histogram with N bins. Therefore, there is a difference between the Histograms shown. As the one with 10 bins has only 10 bins to fill, it is more equalized than the one with 256 bins.

1. Next, carry out histogram equalization as follows: >> P3 = histeq(P,255); Redisplay the histograms for P3 with 10 and 256 bins. Are the histograms equalized? What are the similarities and differences between the latter two histograms?

**Result and Comments:**

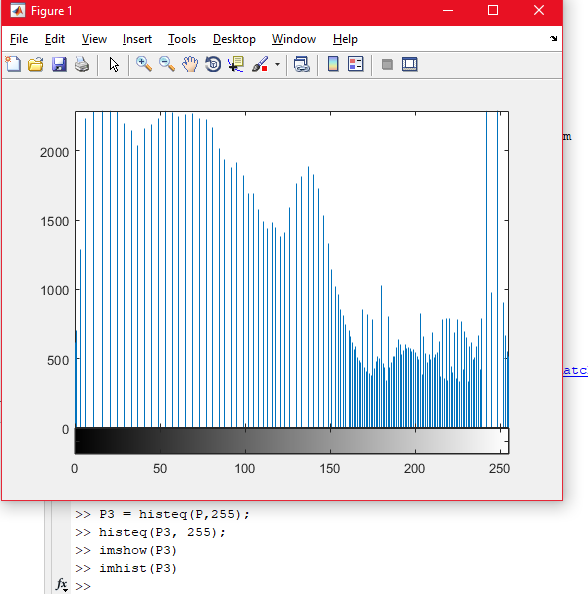
Histogram equalization is done with the following code shown below. With this, the similarities are that the total number of pixels are spread across the 10 and 256 bins. But the difference is that the 10 bins are more equalized than the 256 bins.



1. Rerun the histogram equalization on P3. Does the histogram become more uniform? Give suggestions as to why this occurs.

**Result and Comments:**

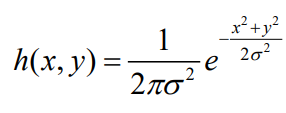
Rerunning the histogram equalization on P3 does not change the output. This follows the definition as stated in the lecture notes. Therefore, the histogram does not become more uniform.



**The histogram equalization method is idempotent**. Doing the method again does not change the histogram. Due to it having the same total number of pixels as well as the same number of bins spread across the probabilities. It will just follow the formula and place the pixels in each bin respective to the previous calculated output bin.

## **Linear Spatial Filtering**

1. Generate the following filters:



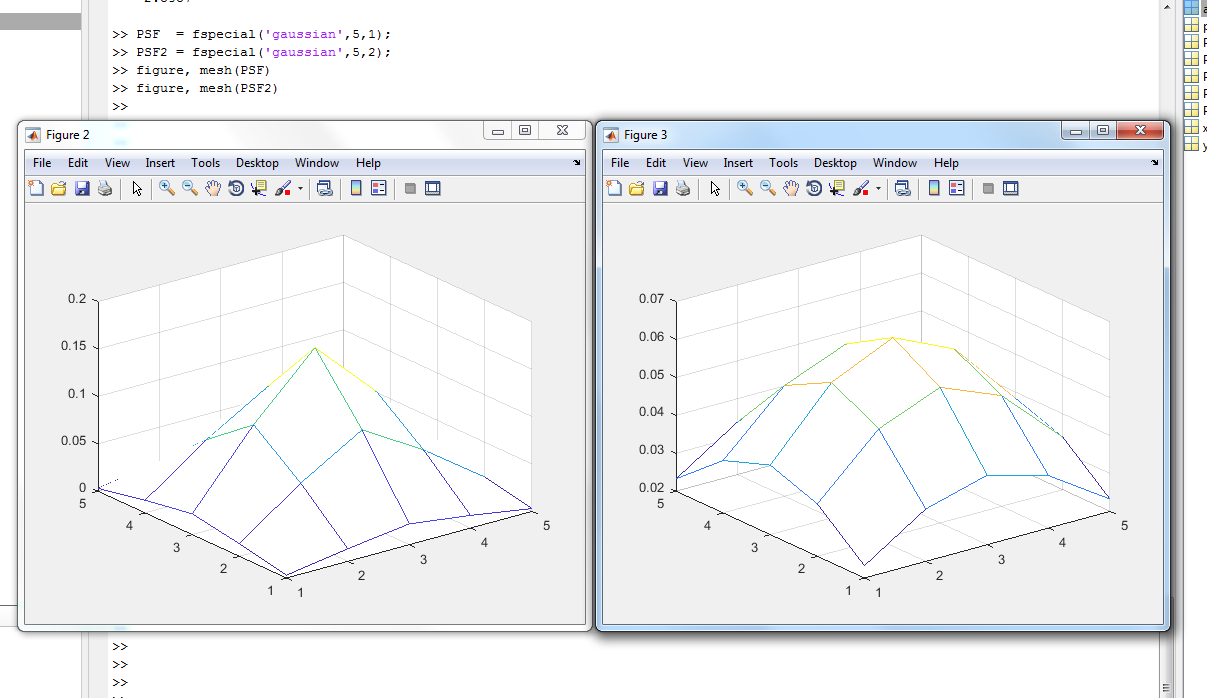
1. Y and X-dimensions are 5 and σ = 1.0
2. Y and X-dimensions are 5 and σ = 2.0

Normalize the filters such that the sum of all elements equals to 1.0. View the filters as 3D graphs by using the mesh function. These filters are Gaussian averaging filters.

**Result and Comments:**

Using the Point-Spread Function (PSF) provided by Matlab, I was able to form the gaussian filter that is a 5x5 matrix which sums up to 1 and has a sigma of 1 as well as a 5x5 matrix with a sigma of 2. This is shown in line 1 and 2 in the below code.

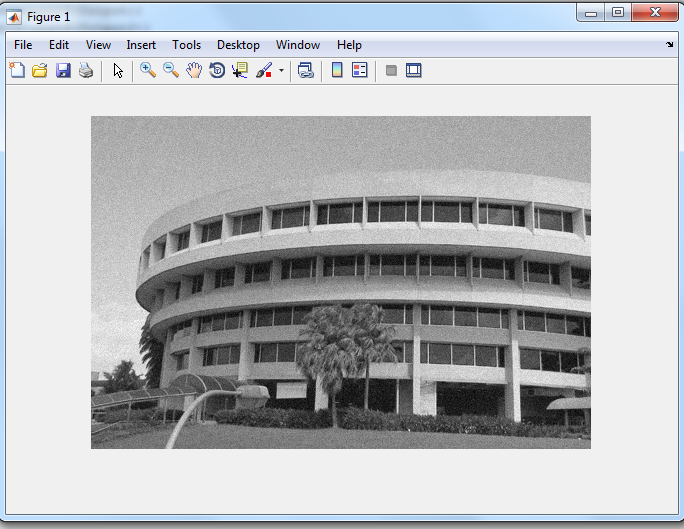
Next the mesh function is called on the PSF. Graph on the left shows the gaussian filter with the sigma of 1. While the graph on the right shows the gaussian filter with the sigma of 2.



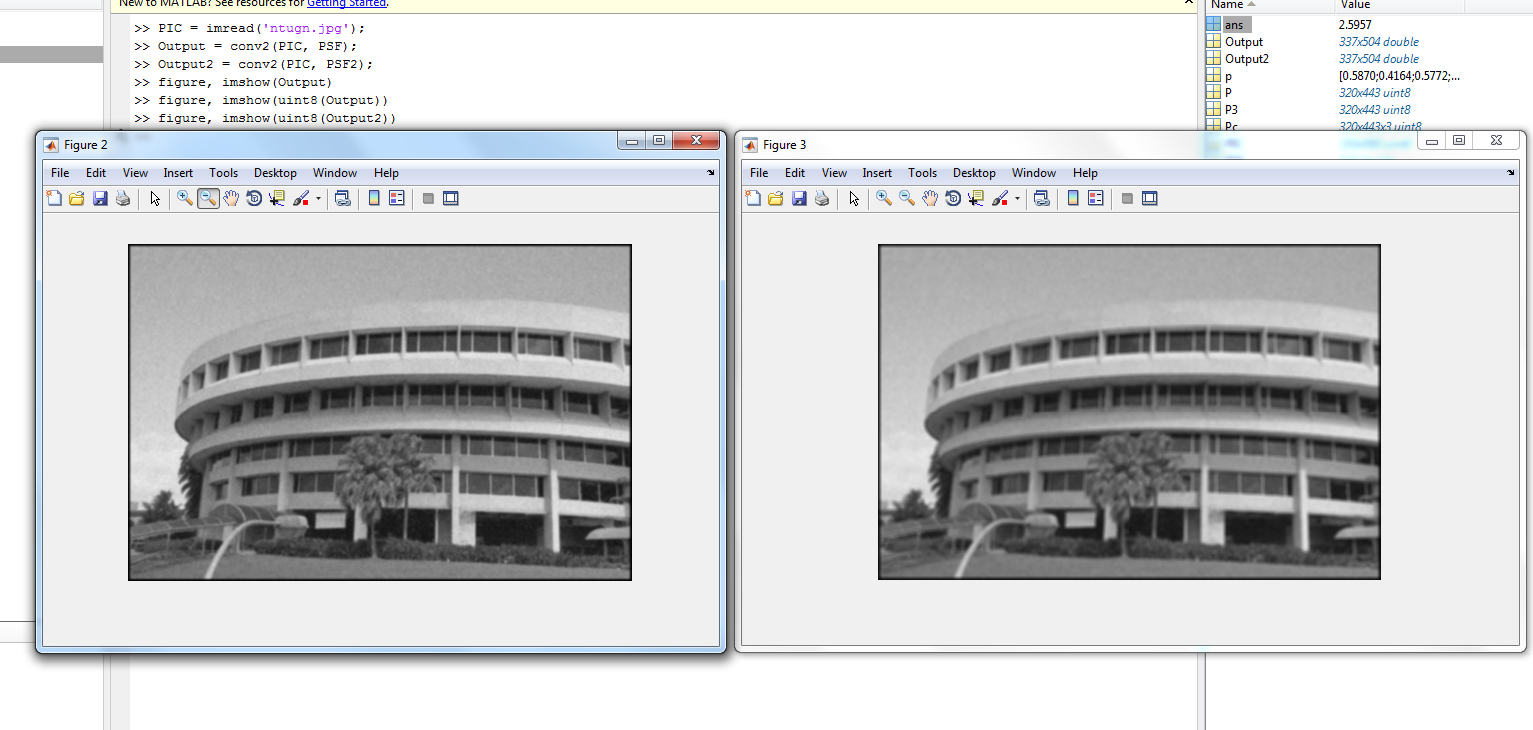
1. Download the image ‘ntu-gn.jpg’ from edveNTUre and view it. Notice that this image has additive Gaussian noise.

**Result and Comments:**

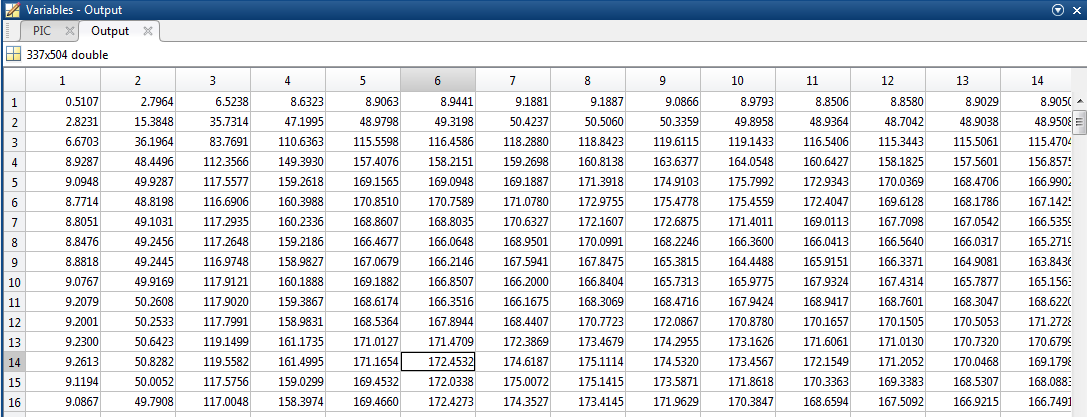
Additive noise is shown on the following figure ‘ntugn.jpg’. The **code is shown below in section c**. along with the convolutions of the 2 different Gaussian averaging filters.

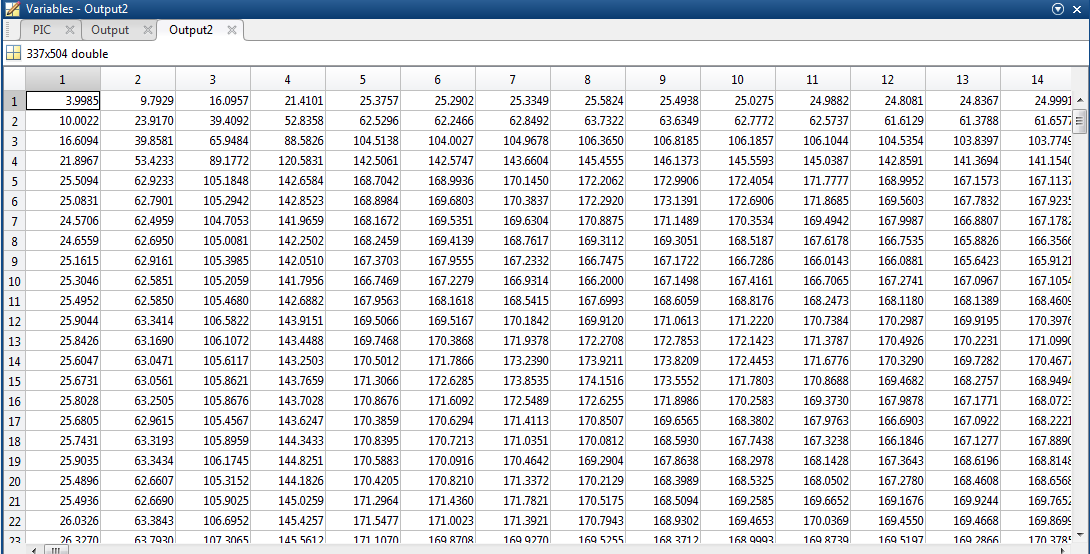


The image is filtered with the gaussian averaging filter and the output is shown below along with the codes written.



**The gaussian averaging filter is quite good at removing the additive gaussian noise**. However, we can look inside the image matrix after it has convolved with the point-spread function. It seems like there is an outer dark line and I have no idea what the cause is.





This is evident in the Output matrix with the point-spread function with sigma = 1. The outer matrices points, (1,1), (1,2), (1,3) etc. all has pixel intensity values close to approximately below 10. This is similar for the output2 matrix with the point-spread function with sigma = 2 as shown below. The values below are approximately less than 30 which is still considered dark.

Out of curiosity, I wanted to find out the reason why the outer layers of the matrix were such small values, (an outer line was drawn) as when I tried by using manual convolution by using a calculator, I didn’t get the exact replica of the final image values of the position (0,0) (1,1 in Matlab). I noticed that the conv2 Matlab function has 3 settings:

* 'full' — Return the full 2-D convolution. (Default)
* 'same' — Return the central part of the convolution, which is the same size as A.
* 'valid' — Return only parts of the convolution that are computed without zero-padded edge

By ‘Full’ 2-D convolution, the **function calculates the values for the zero-padding as well and a new image of an increased size is formed**. As it is a 5x5 filter / kernel we are using for the convolution a zero-padding of 2k+1 where k=2 is the number of neighbors we are interested in in relation to the center pixel. The new image produced has larger dimensions as compared to the original image as it is padded twice with zeros. The (1,1) position results in the following computation: 172 x 0.0029690167435050 = 0.5107

The table below illustrates the above. The Full 2-D Conv operation computes all of the zero padding highlighted in yellow. An example, can be seen if we were to look at the position (3,3) of the new matrix is taken by taking the dot-product of the two matrices below. Thereby producing the outlines.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.00296901674395050 | 0.0133062098910137 | 0.0219382312797146 | 0.0133062098910137 | 0.00296901674395050 |
| 0.0133062098910137 | 0.0596342954361801 | 0.0983203313488458 | 0.0596342954361801 | 0.0133062098910137 |
| 0.0219382312797146 | 0.0983203313488458 | 0.162102821637127 | 0.0983203313488458 | 0.0219382312797146 |
| 0.0133062098910137 | 0.0596342954361801 | 0.0983203313488458 | 0.0596342954361801 | 0.0133062098910137 |
| 0.00296901674395050 | 0.0133062098910137 | 0.0219382312797146 | 0.0133062098910137 | 0.00296901674395050 |

Table 1. Shows the actual non-rounded up values of the filter / kernel we are using copied from Matlab

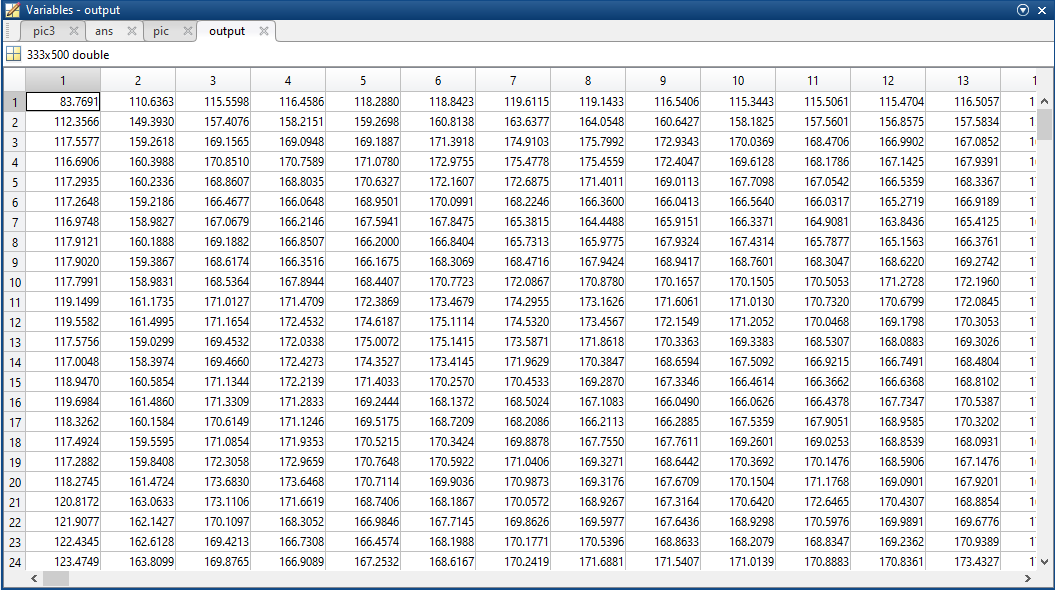
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 172 |

Table 2. The Original image, highlighted in yellow shows the actual zero-padded and an additional zero-padding to calculate the value highlighted in green

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0.510670879959486 | 2.79636996446988 | 6.52378035050633 |
| 0 | 0 | 2.82309111516544 | 15.3848070653372 | 35.7314188439549 |
| 0 | 0 | 6.67025739022101 | 36.1963773438702 | 83.7691005830807 |

Table 3. The final result of the Full 2-D Conv operation on the above matrices

The value highlighted in orange in Table 3. is where the real image (x,y) values start. We can prove this by using the ‘same’ parameter in conv2. As proved, the (1,1) of the output is the same as the value highlighted in orange.



The image above shows the output matrix of the finally computed picture. With this, we can now continue with our processing of the images using the Gaussian average filter.



Evidently, the black outlines are removed, and we have our image applied with the gaussian average filter. Figure 1 above shows the Image with the variance = 1, while Figure 2 above shows the Image with variance = 2. **The larger the variance, the more blurred the image will be.**

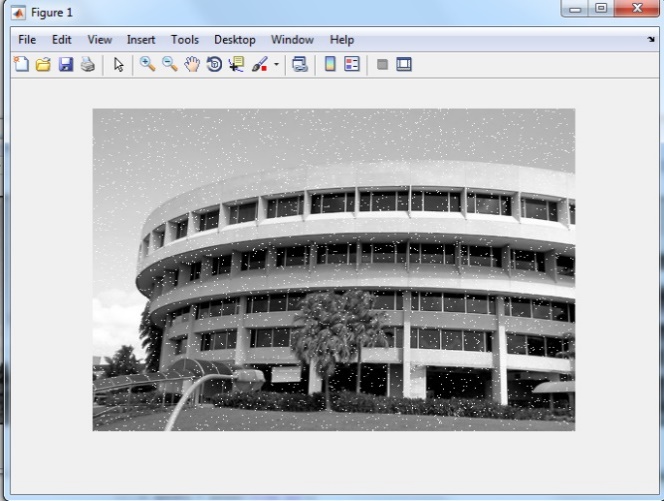
**The additive gaussian noise are removed but with the trade-off of the edges not being as detailed and the image is blurred.**

1. Filter the image using the linear filters that you have created above using the conv2 function, and display the resultant images. How effective are the filters in removing noise? What are the trade-offs between using either of the two filters, or not filtering the image at all?

**Result and Comments:**

Figure below shows the “ntusp.jpg” provided in the NTU blackboard. Speckle noise are spots that can be found in the image.

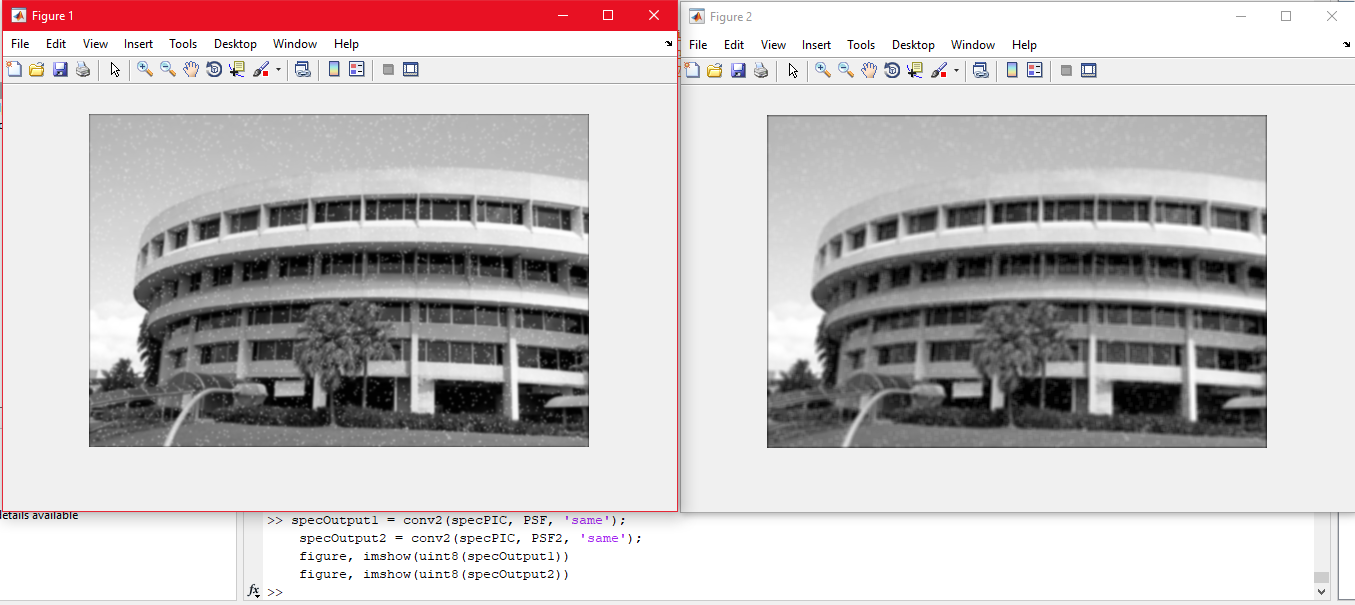
1. Downloaded the image ‘ntusp.jpg’ from edveNTUre and view it. Notice that this image has additive speckle noise.



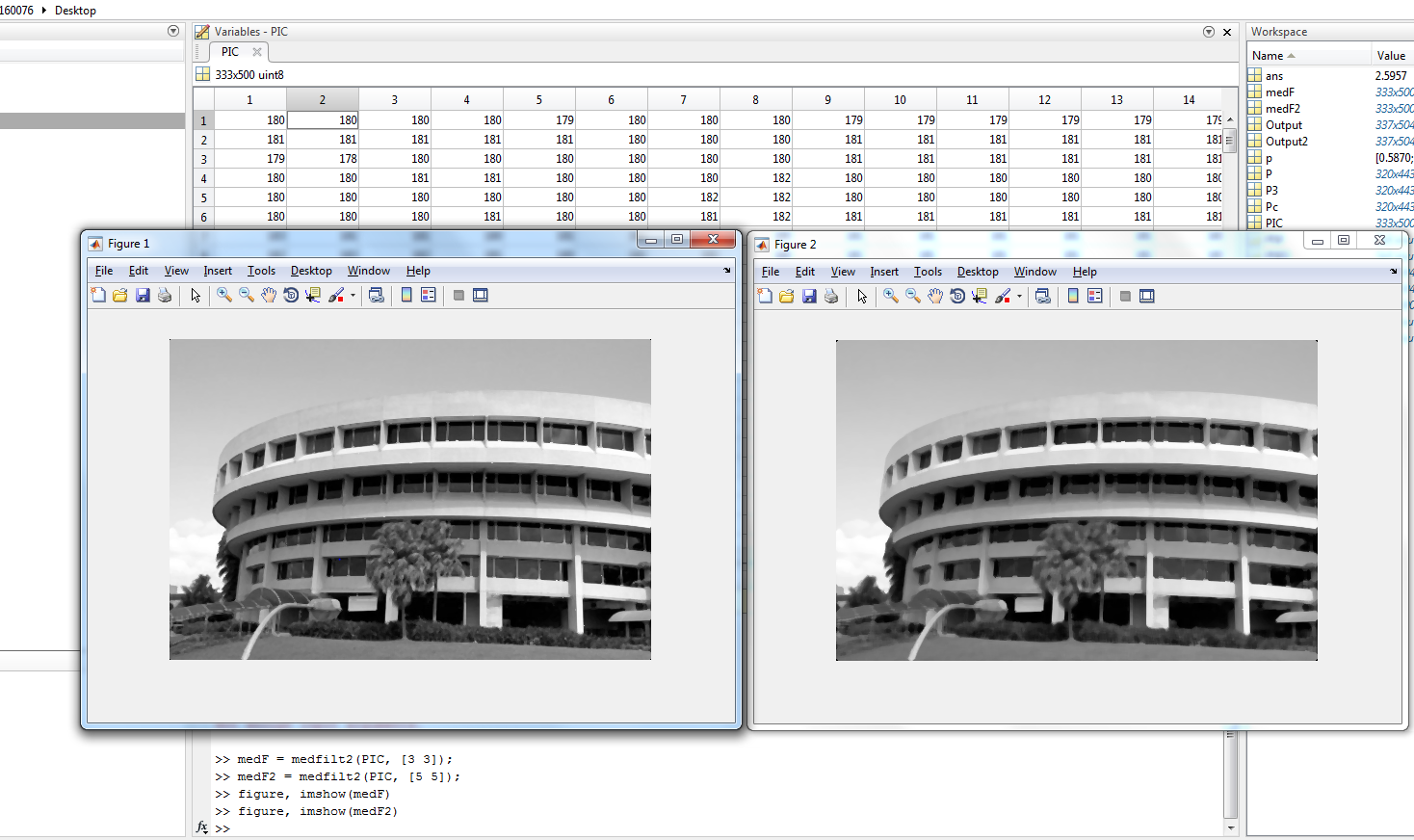


**Result and Comments:**

Applying the same step as above, we look closely we can see that the speckles are not entirely gone. The code is shown below. We can conclude that the gaussian filter is not good at removing speckle noises.



## **2.4 Median Filtering**



**Result and Comments:**  
The medfilt2 function in Matlab takes in two variables: an image and a filter of N x M size. The 3x3 median filter output is shown on the left while the 5x5 median filter output is shown on the right.

The speckles are totally removed as shown in both images. This proves that median filter is better than gaussian averaging filters at removing speckles.

**What are the trade-offs between Median Filter and Gaussian Filter?**

On the left below, we have the gaussian additive noise image applied with gaussian filter, while on the right we have the speckles image applied with the speckles. We can conclude that the Median filter does very well at removing “salt-and-pepper” noises while the Gaussian blurs the image to reduce noise. Applying the Gaussian filter, we can note that the edges are less apparent as compared to the Median filter which preserves the edge detail more than the Gaussian filter.

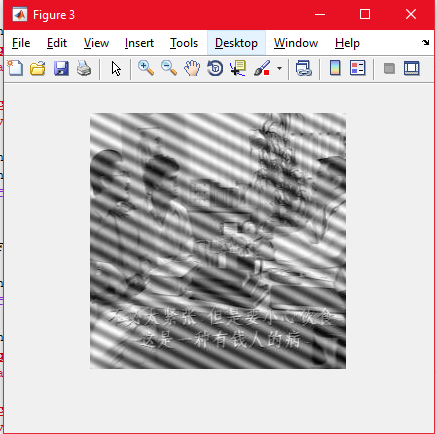


## **2.5 Suppressing Noise Interference Patterns**

a.Download the image ‘pck-int.jpg’ from edveNTUre and display it from MATLAB. Notice the dominant diagonal lines destroying the quality of the image.

**Result and Comments:**





The image is affected by a **periodic noise**. The periodic noise are repeated patterns that seems to have been added on top of the original image.

b. Obtain the Fourier transform F of the image using fft2, and subsequently compute the power spectrum S. Note that F should be a matrix of complex values, but S should be a real matrix. Display the power spectrum by

>> imagesc(fftshift(S.^0.1));

>> colormap(‘default’);

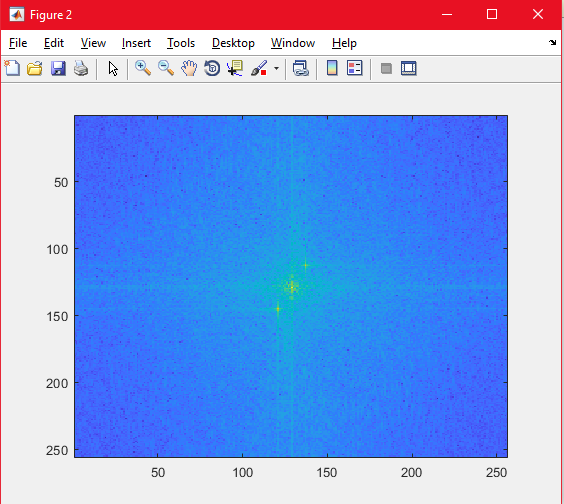
fftshift is used to shift the origin of the Fourier transform to the centre of the image. Note that the origin corresponds to the DC (or zero frequency) component. The power to 0.1 is only used to nonlinearly scale the power spectrum such that it is easier to visualize the frequency components. Notice that there are two distinct, symmetric frequency peaks that are isolated from the central mass. These frequency components correspond to the interference pattern.

**Result and Comments:**

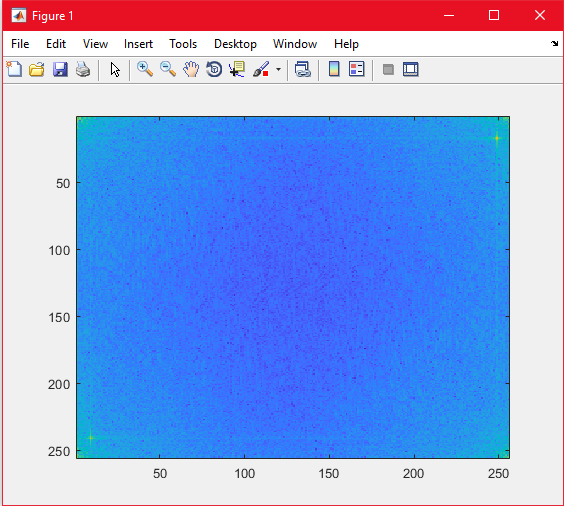
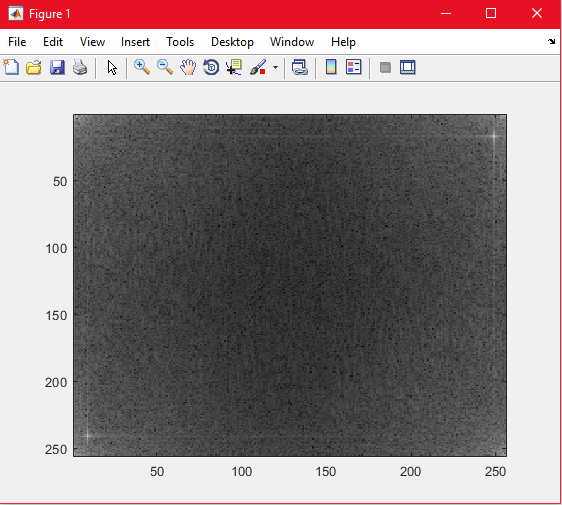


The FFT2 function is called which converts the input image from the spatial domain to the frequency domain. Theoretically the complex numbers of the image are zero, however, the reason why we need to take the absolute value of the complex array after performing the Fourier Transform is due to the reason in which how the computers represent zeroes. They usually represent it in the form of approximately e^-14 which is still a small number. Therefore, to prevent any errors from occurring before displaying I took the absolute value of variable F and store it to the variable S.

The below images are shown using imagesc which shows the power spectrum. As the default colormap is a bit bright and hard to see, I decided to convert it to the gray colormap instead for a better view. The two frequency components separated from the center (which is the zero-frequency component - DC component) of figure 2 represents the interferences.

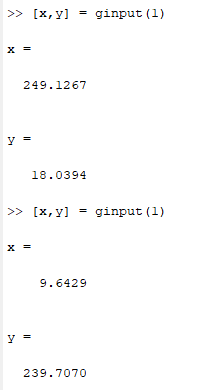
 

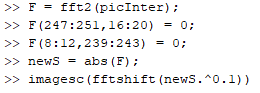
1. Redisplay the power spectrum without fftshift. Measure the actual locations of the peaks. You can read the coordinates off the x and y axes, or you can choose to use the ginput function.

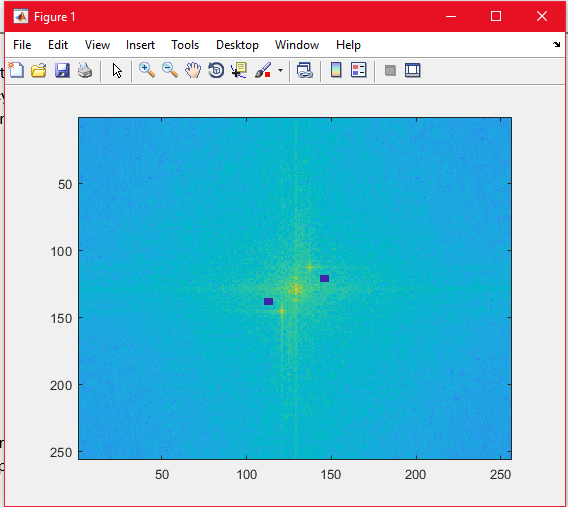
**Result and Comments:**

Redisplaying the figures without FFT-Shift and setting it in ‘default’ and ‘gray’ colormaps allow us to see clearly where the frequency components are. The two components are now on the top right hand corner as well as the bottom left hand corner.



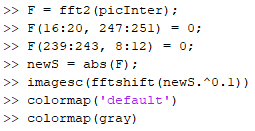


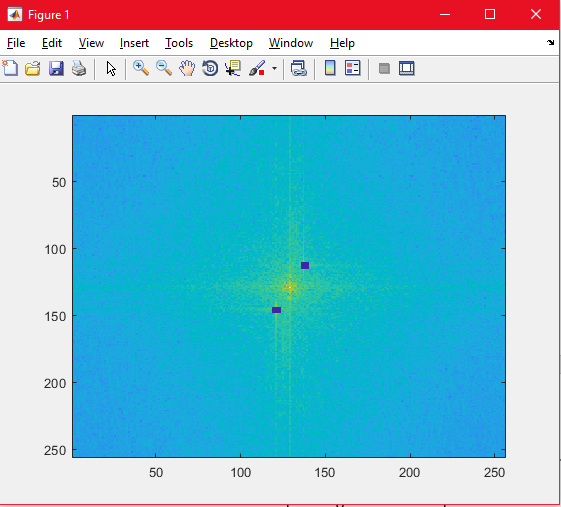
Using ginput(1) which returns 1 selected position of the particular figure, I was able to determine the x and y locations of the components. The bottom left point corresponds to (10, 241) and the top right point corresponds to (249, 18) rounded to the whole numbers for the power spectrum without fftshift.



After doing section d once, I realized that the boxes were wrong. It wasn’t place on the corresponding symmetry peaks. Ginputs read the x and y coordinates like a normal Cartesian Coordinate System where the horizontal axis is normally labelled as X and the vertical axis is labelled as Y. Therefore, I needed to swap the coordinates to (241,10) and (18,249) instead.

1. Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks in the Fourier transform F, not the power spectrum. Recompute the power spectrum and display it as in step (b).

**Result and Comments:**  
Based on the above explanation, the code should be rewritten as follows. Noting the swap of the X and Y axis.

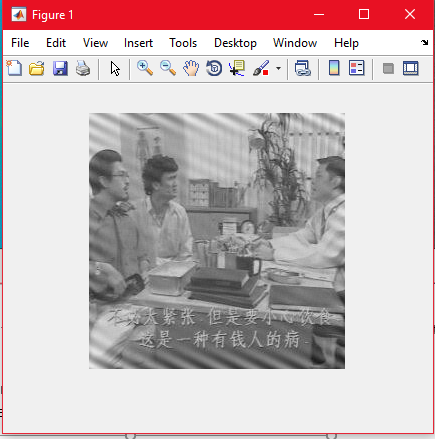


With these, we can see that the two peaks are finally covered. Those covered areas represent the values where it is zero.

1. Compute the inverse Fourier transform using ifft2 and display the resultant image. Comment on the result and how this relates to step (c). Can you suggest any way to improve this?

**Result and Comments:**  
The resultant transformed image has lesser interference. However, we can still see the interreference is not fully removed. In addition, a way to improve this is to create a low pass filter which will filter out the periodic noise.

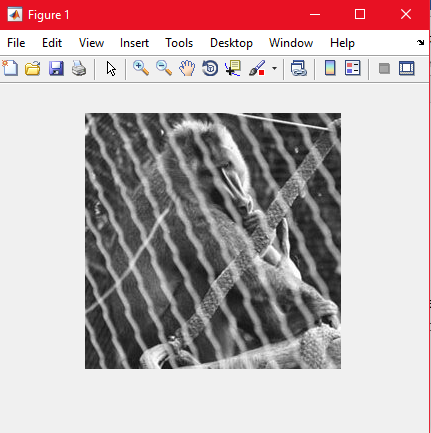




1. Download the image `primate-caged.jpg’ from edveNTUre which shows a primate behind a fence. Can you attempt to “free” the primate by filtering out the fence? You are not likely to achieve a clean result but see how well you can do.

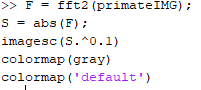
**Result and Comments:**

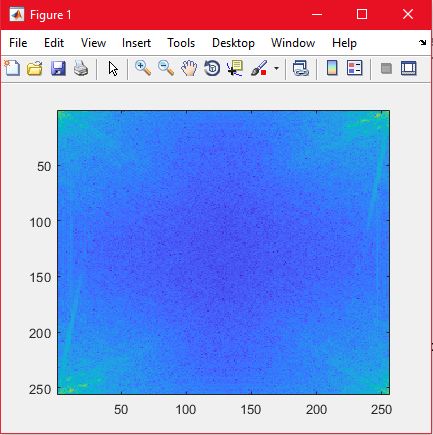
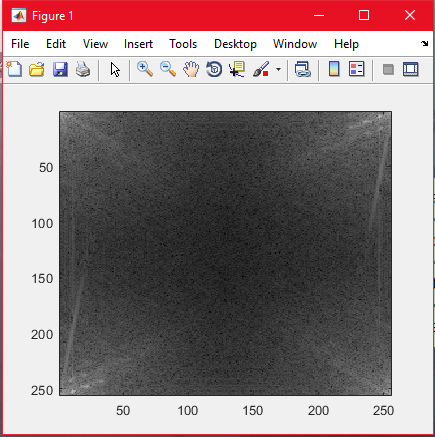




Reading in the primate image and displaying it we can notice that there is a fence that is obstructing the view of the primate.

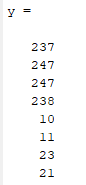
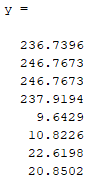
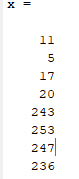
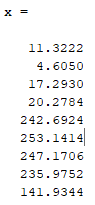
**Step 1: Convert the image into the frequency domain and subsequently the power spectrum to see where the peaks are situated at. I displayed in gray and default colormap for easier viewing.**

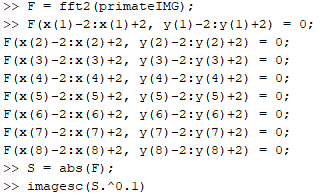
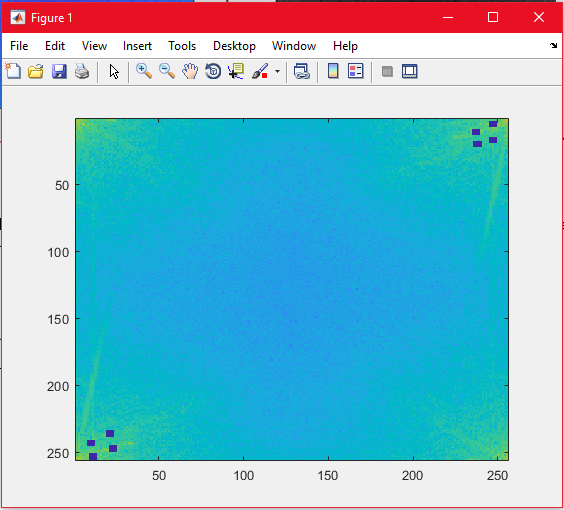




**Step 2: Get all the location of the peak by using the ginput function. Following figure displays the corresponding peaks. Remember, the ginput function reads in as x-y axis of the Cartesian coordinate system therefore there is a need to swap the y and x axis.**

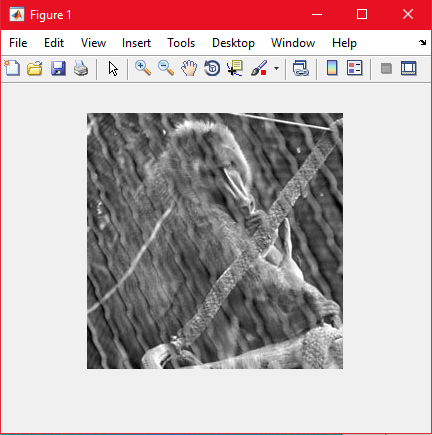
**Step 3: Round up the whole numbers and set the neighboring 5-by-5 pixel of the determined locations in the Fourier domain to be 0.**



After doing the above, the image shows the power spectrum with the corresponding portion that we set to 0.

**Step 4: Convert back to the Spatial Domain using IFFT2 and view the image.**



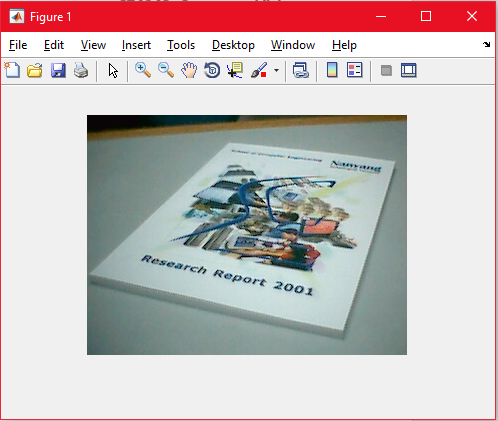


From the results, we can see that the fence is removed however, not entirely. It is hard to achieve clean result as the lines from the cage are not parallel.

## **2.6 Undoing Perspective Distortion of Planar Surface**

1. Download `book.jpg’ from the edveNTUre website as a matrix P and display the image. The image is a slanted view of a book of A4 dimensions, which is 210 mm x 297 mm.

**Result and Comments:**

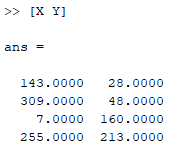
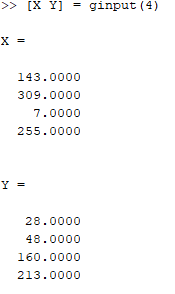


1. The ginput function allows you to interactively click on points in the figure to obtain the image coordinates. Use this to find out the location of 4 corners of the book, remembering the order in which you measure the corners.

>> [X Y] = ginput(4)

Also, specify the vectors x and y to indicate the four corners of your desired image (based on the A4 dimensions), in the same order as above.

**Result and Comments:**  
The order of the **corners for the book** was measured from the top left corner of the book, top right corner, bottom left corner and lastly the bottom right corner.



For the image vector x and y, the arrays are in this format as it follows the A4 size paper dimensions as well as the how ginput was taken.

Top Left-Hand Corner: (0,0),   
Top Right-Hand Corner: (210,0),   
Bottom Left-Hand Corner: (0, 297),   
Bottom Right-Hand Corner: (210, 297).



1. Set up the matrices required to estimate the projective transformation based on the equation (\*) above.

>> u = A \ v;

The above computes u = (A)e-1 v, and you can convert the projective transformation parameters to the normal matrix form by

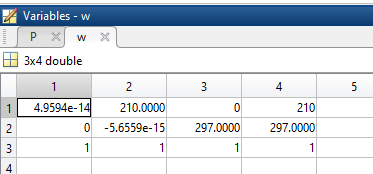
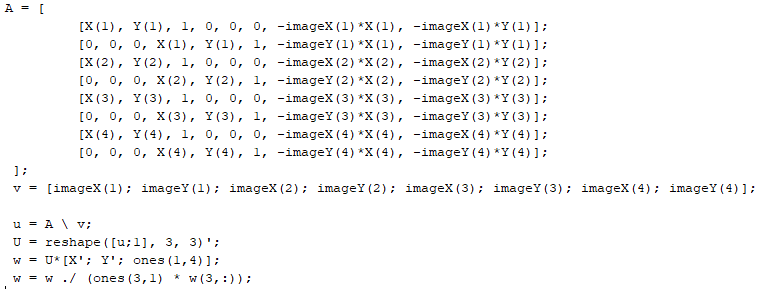
>> U = reshape([u;1], 3, 3)';

Write down this matrix. Verify that this is correct by transforming the original coordinates

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

Does the transformation give you back the 4 corners of the desired image?

**Result and Comments:**



Yes. The transformation does give back the 4 corners of the desired image. The values are like that of those that have been declared in part b vectors x and y.

1. Warp the image via

>> T = maketform(‘projective’, U’);

>> P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);

**Result and Comments:**



1. Display the image. Is this what you expect? Comment on the quality of the transformation and suggest reasons.

**Result and Comments:**

No. I did not expect such a good outcome. I am amazed by the output. The quality is good considering we did not take the picture directly on the book itself.

However, note that the top of the image is blurred. The top wordings for the transformed image (right) is blurred, however this can also be seen from the original image (left) as the words are hardly readable. This may also be caused by the resolution of the original image which is not very clear. But overall, it’s performance is astounding.



