DIP Assignment 2

Naila Fatima
201530154

Problem 1: Image Resampling

a) Gaussian pyramid

Parameters 1: Size of filter = 3, Sigma = 0.5, levels = 4

Input Image:

Output Images:

Level 1: (Level 0 is input image)

Level 2:

Level 3:

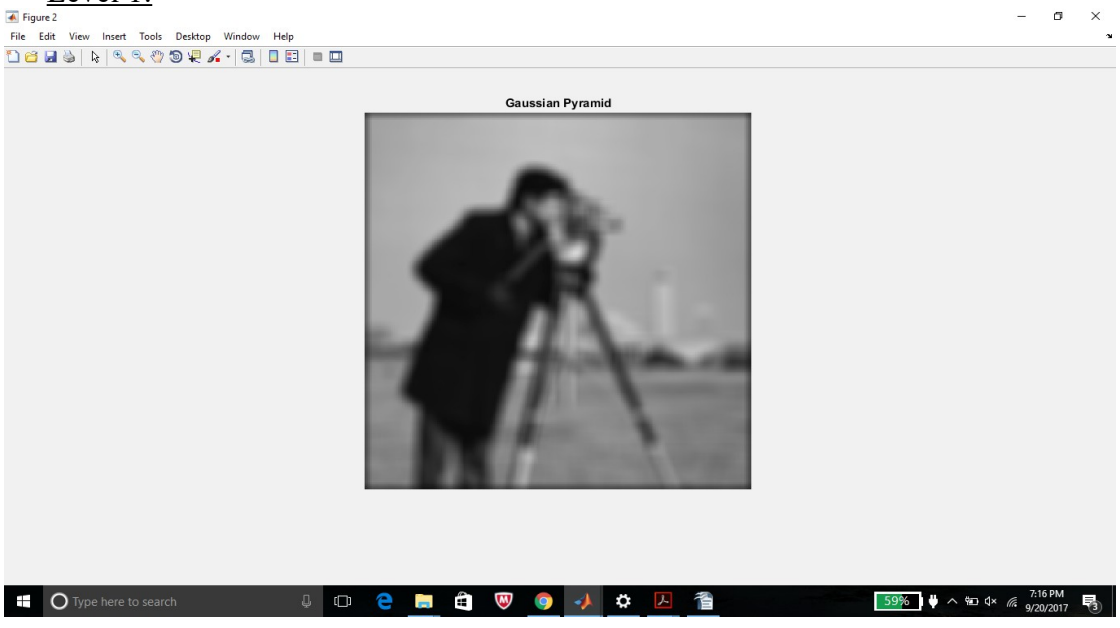
Level 4:

Parameters 2: Size of filter = 10, sigma = 5, number of levels = 5

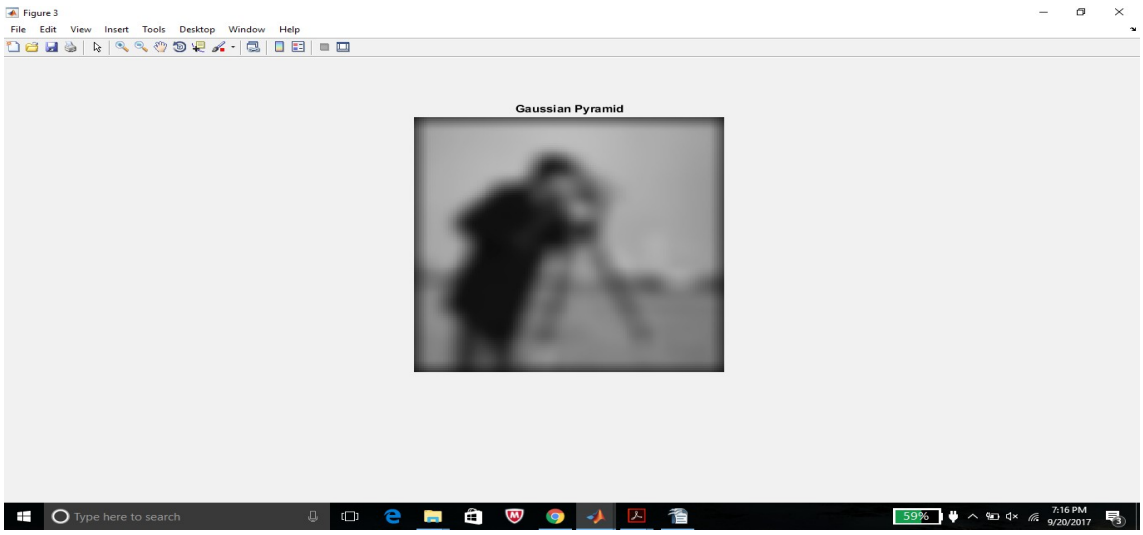
Input Image:

Output Images:

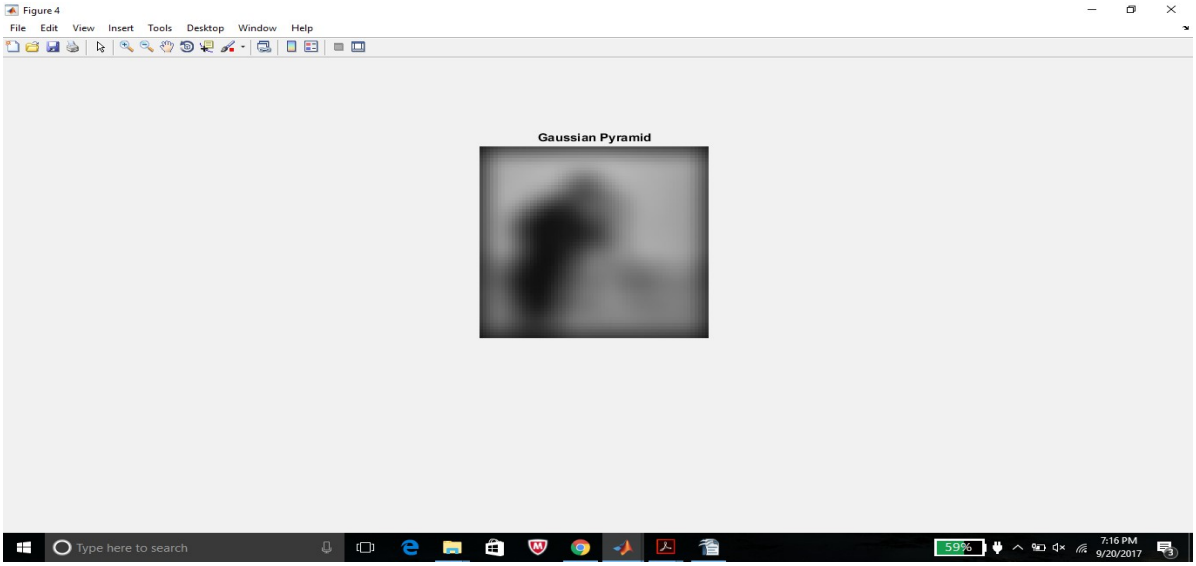
Level 1:



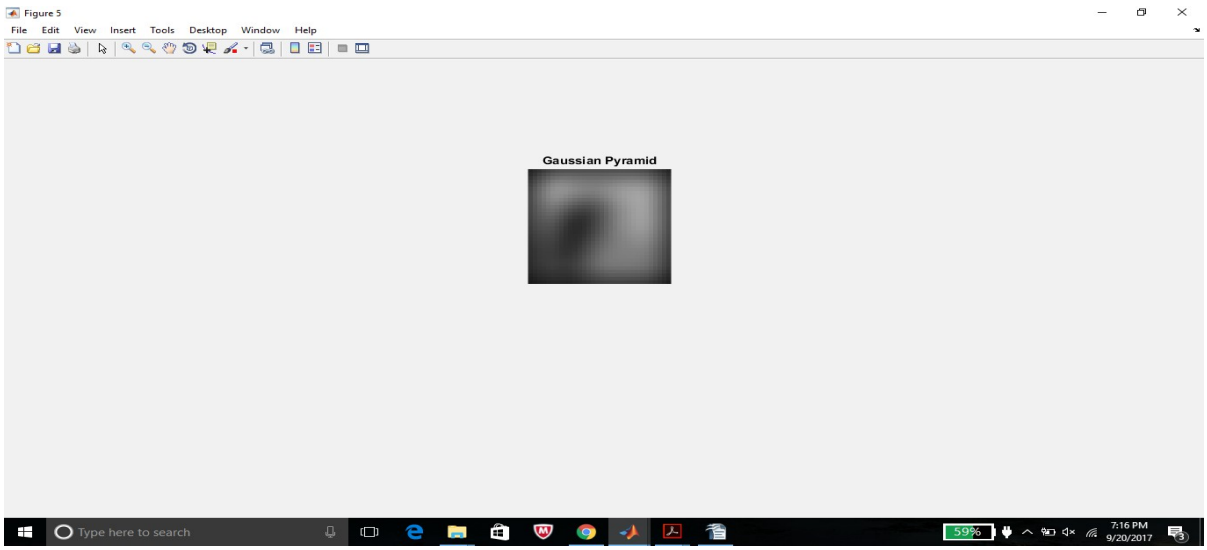
Level 2:



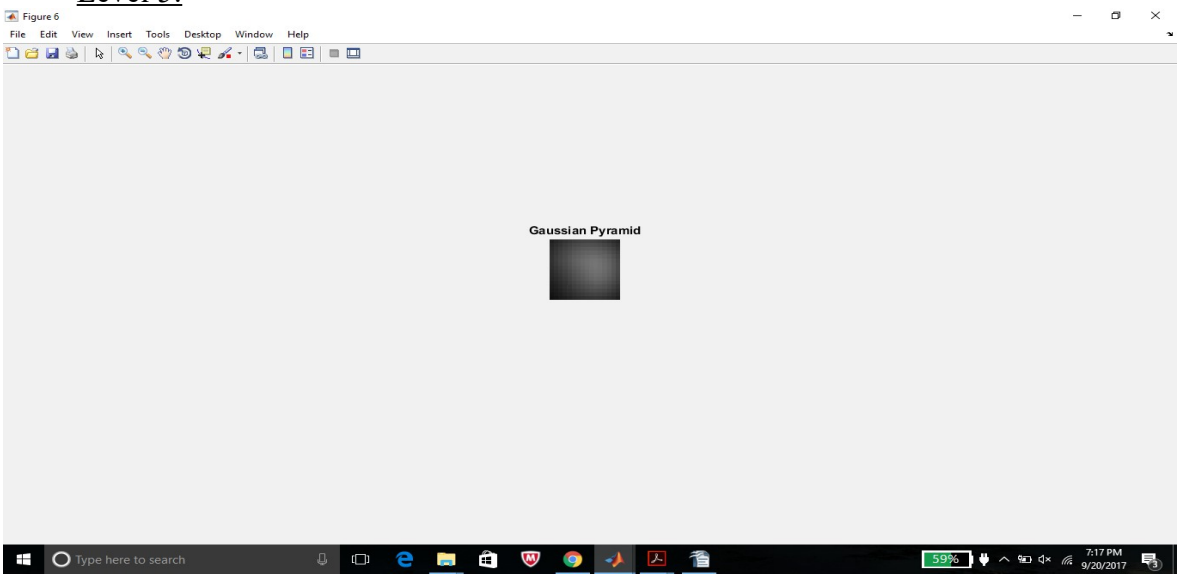
Level 3:



Level 4:

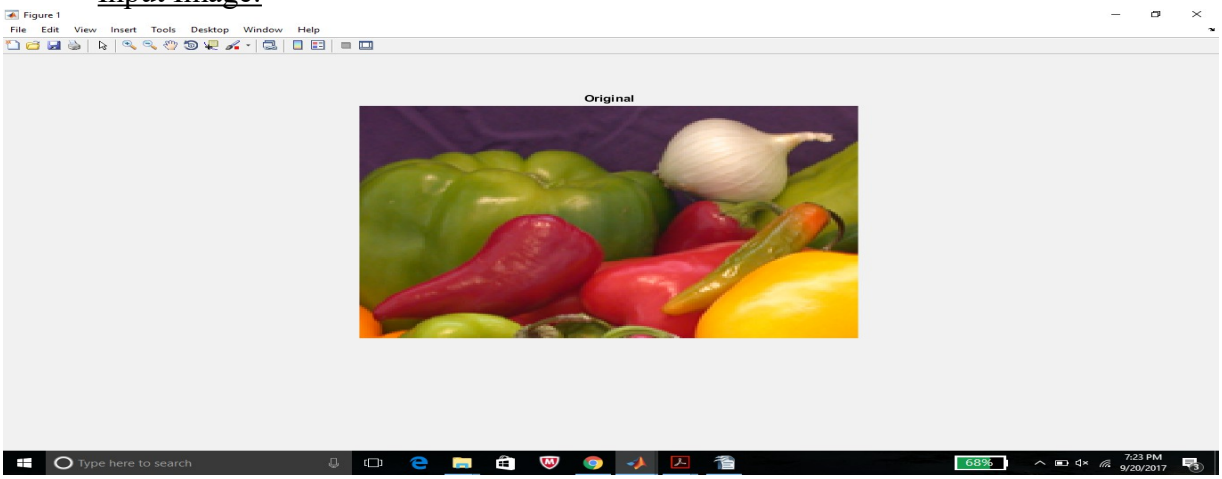


Level 5:



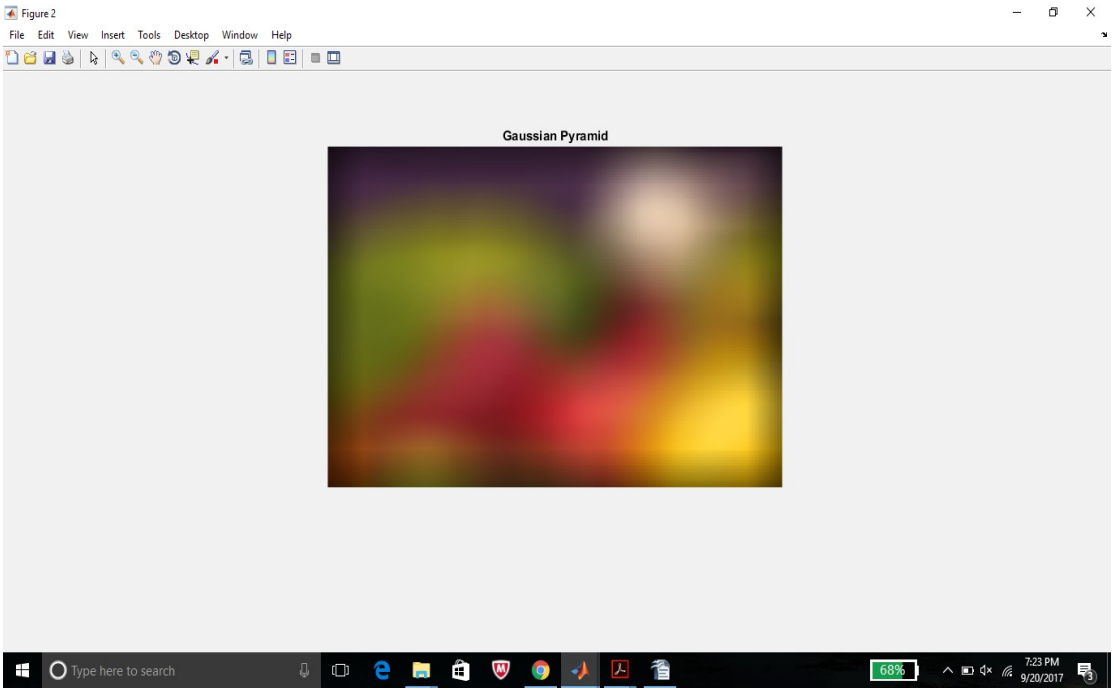
Parameters 3: Size of filter = 30, sigma = 30, levels = 3

Input Image:

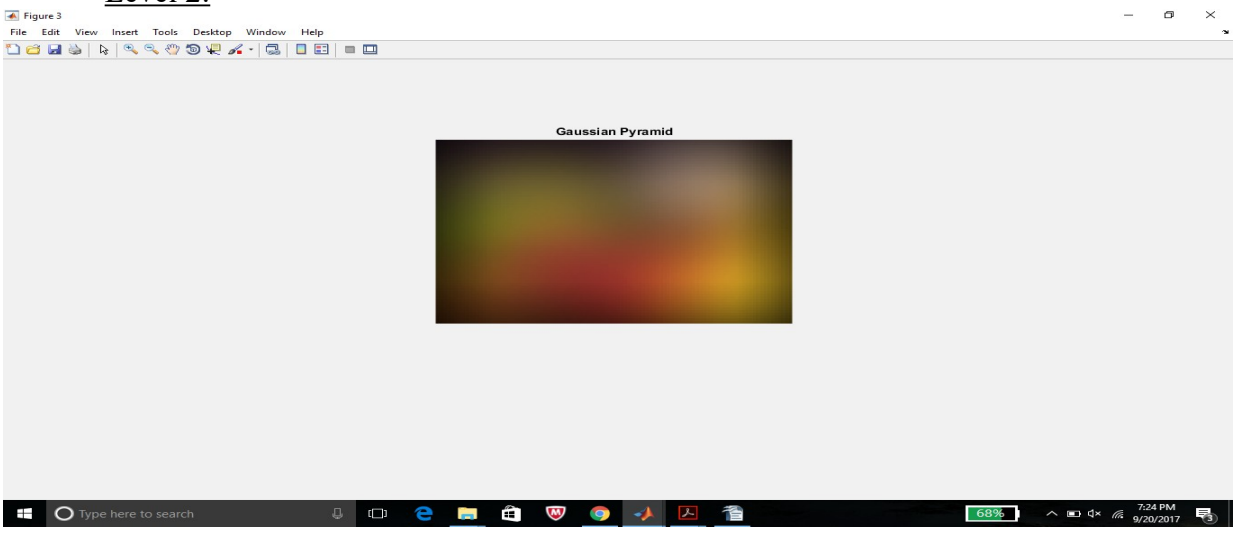


Outputs:

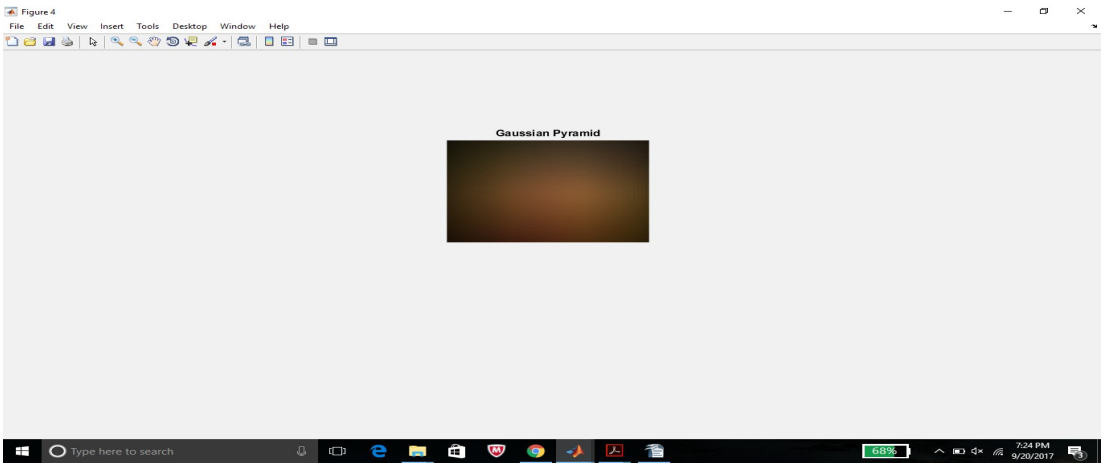
Level 1:



Level 2:



Level 3:



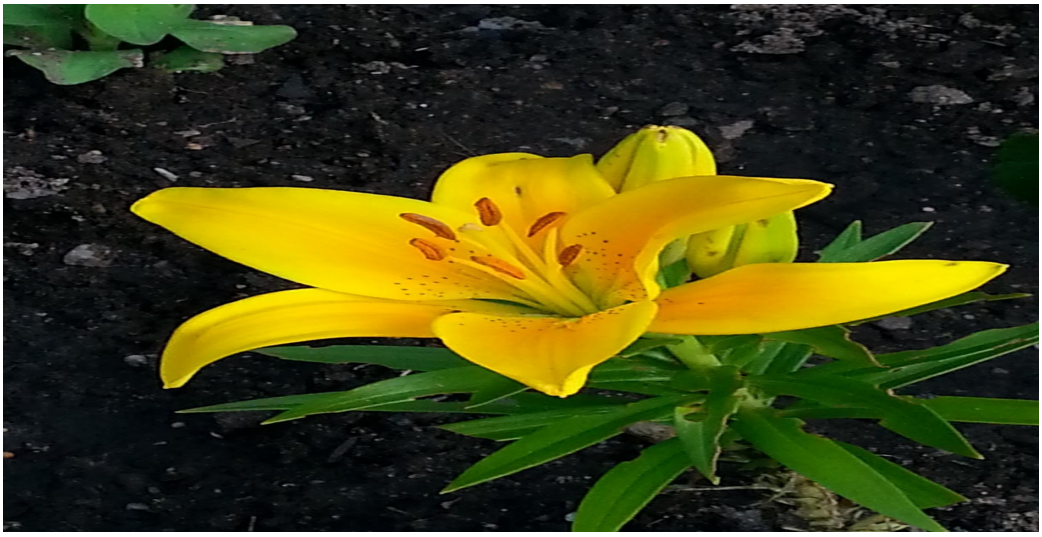
Observations: We can notice that the blurriness of the image at a particular level of the Gaussian pyramid depends on the size of filter, sigma value and the level of pyramid. As the number of levels increase, the images get more blurred. Similarly, as the sigma value and size of filter increase, the blurriness increases again.

Results: To get an ideal blurred image, the sigma and size of filter should be kept relatively small. Sigma between 0.5 and 6 is optimal and the size of filter should be between 3x3 to 15x15 for better outputs as larger filters blur the outputs too much. Also, the number of levels should be kept below 10 for better results.

Laplacian Pyramid:

Parameters 1: Size of filter = 3, sigma = 0.5, number of levels = 5

Input Image:

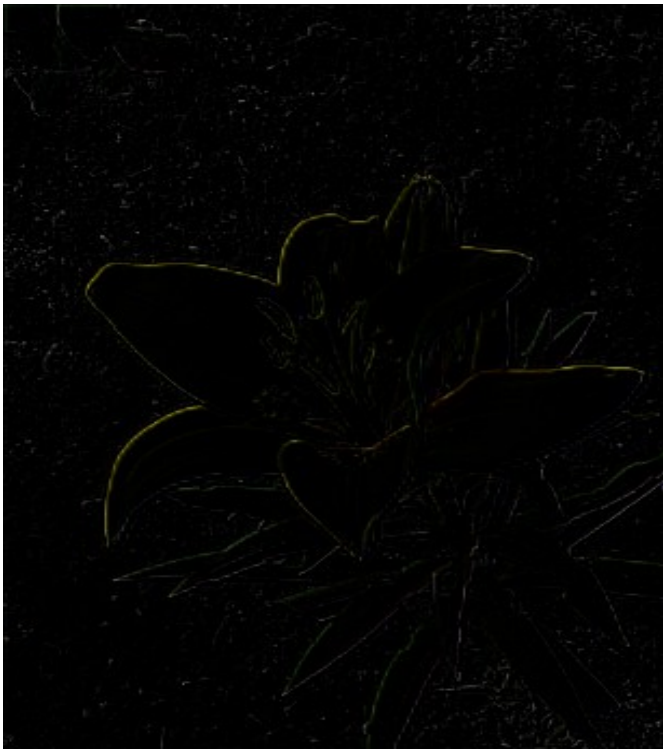


Outputs:

Level 0:

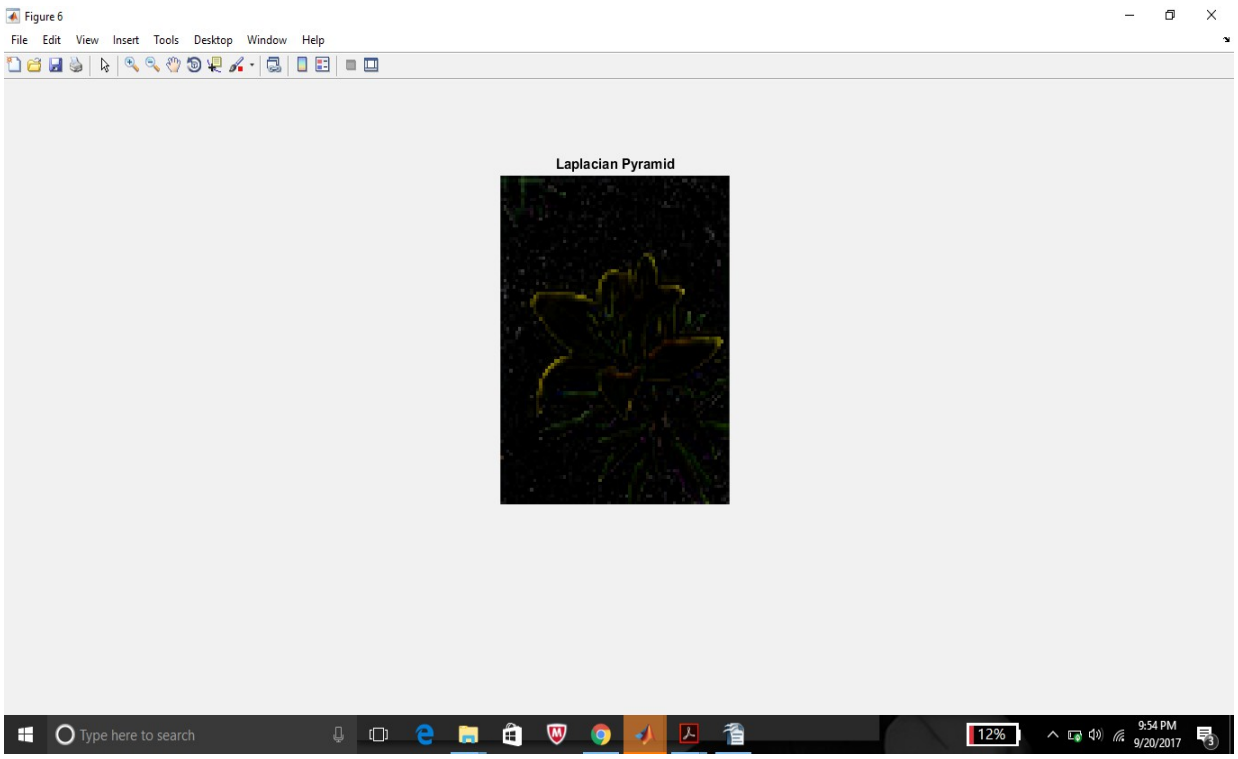
Level 1:

Level 2:

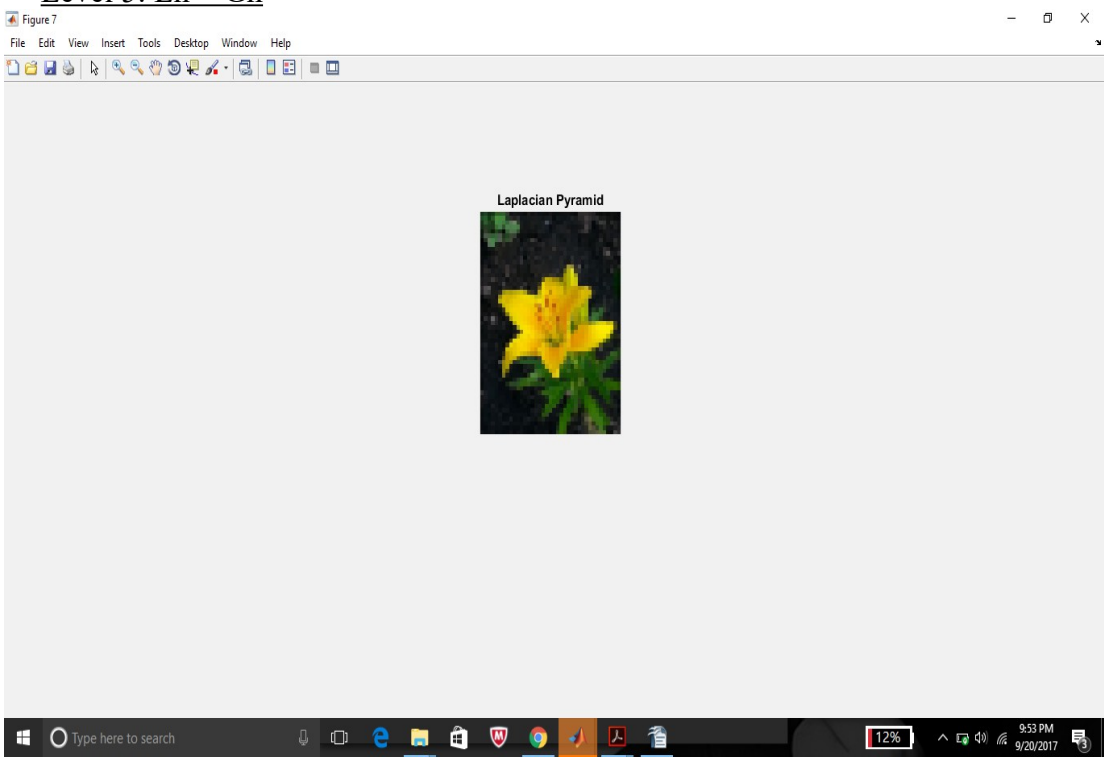


Level 3:

Level 4:

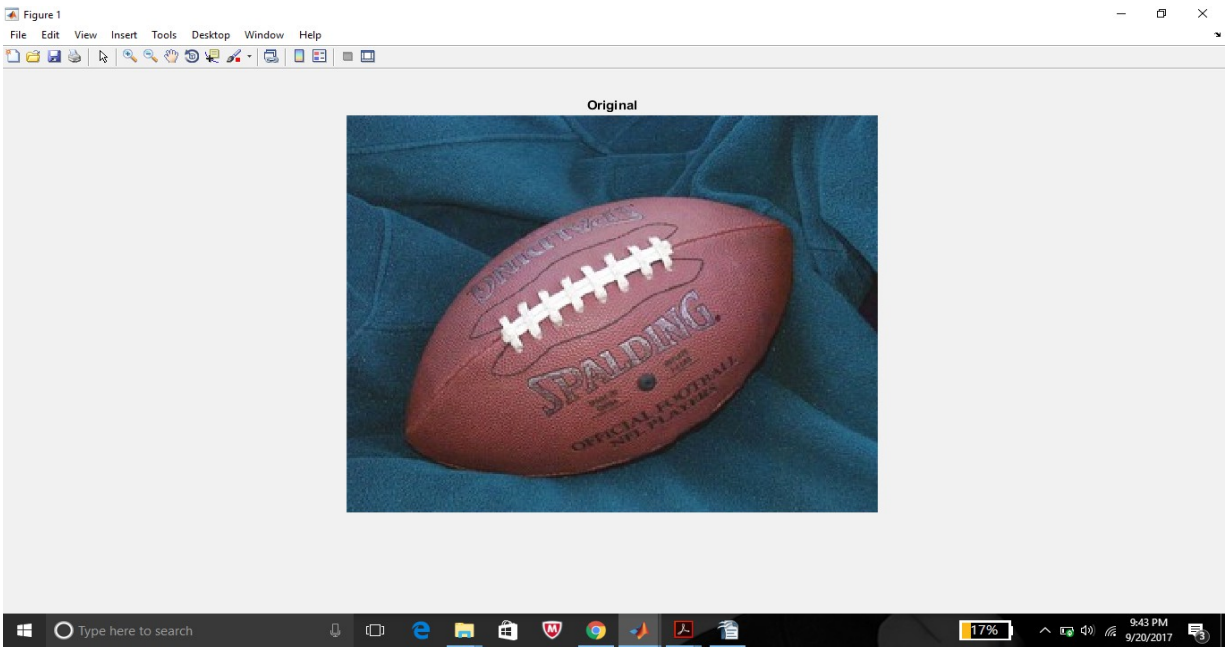


Level 5: Ln = Gn



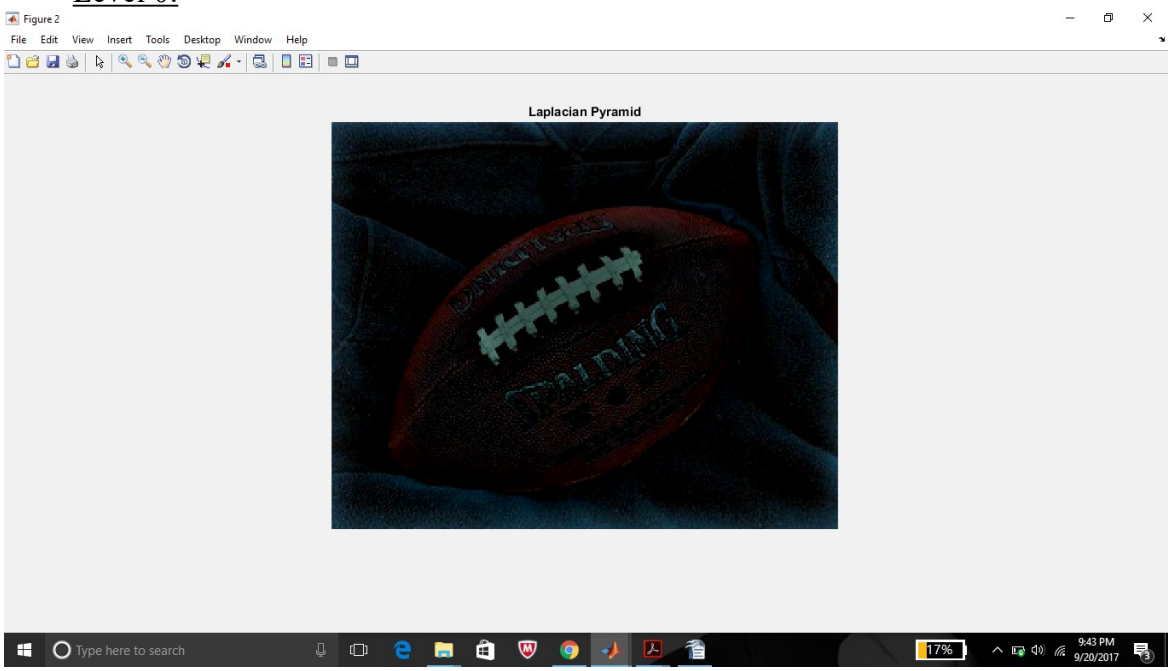
Parameters 2: Size of filter = 40, sigma = 15, levels = 4

Input Image:

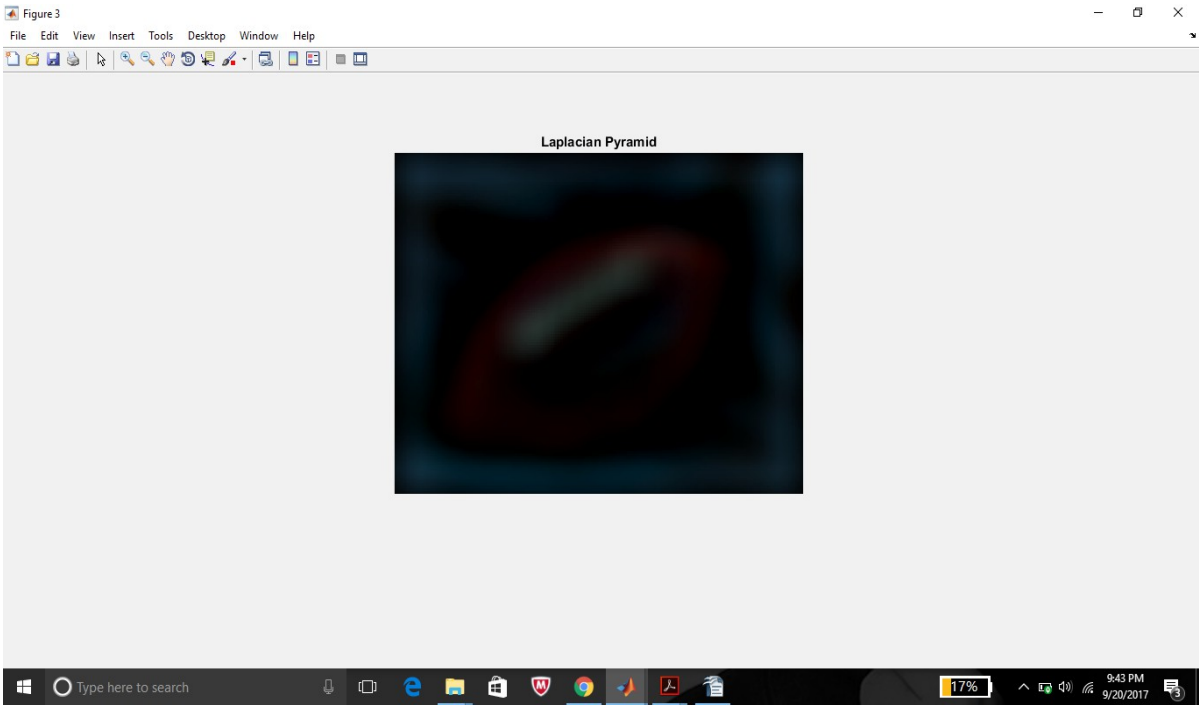


Outputs:

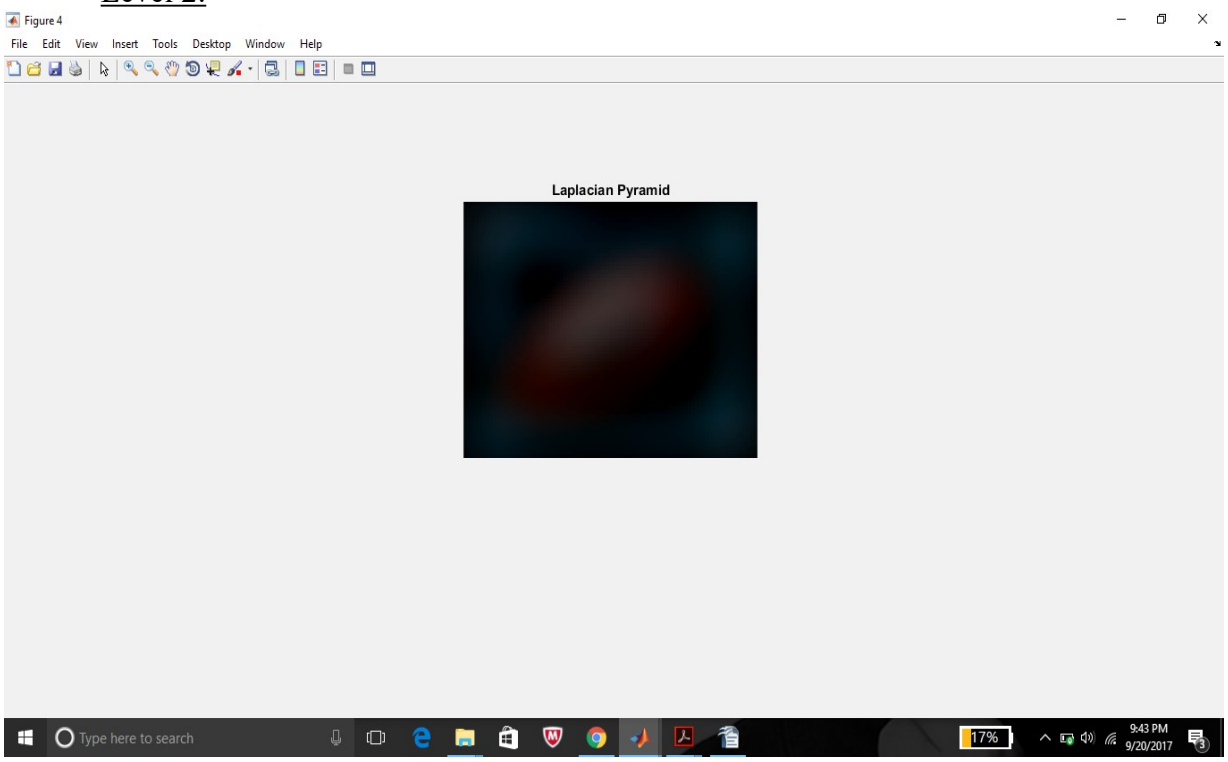
Level 0:



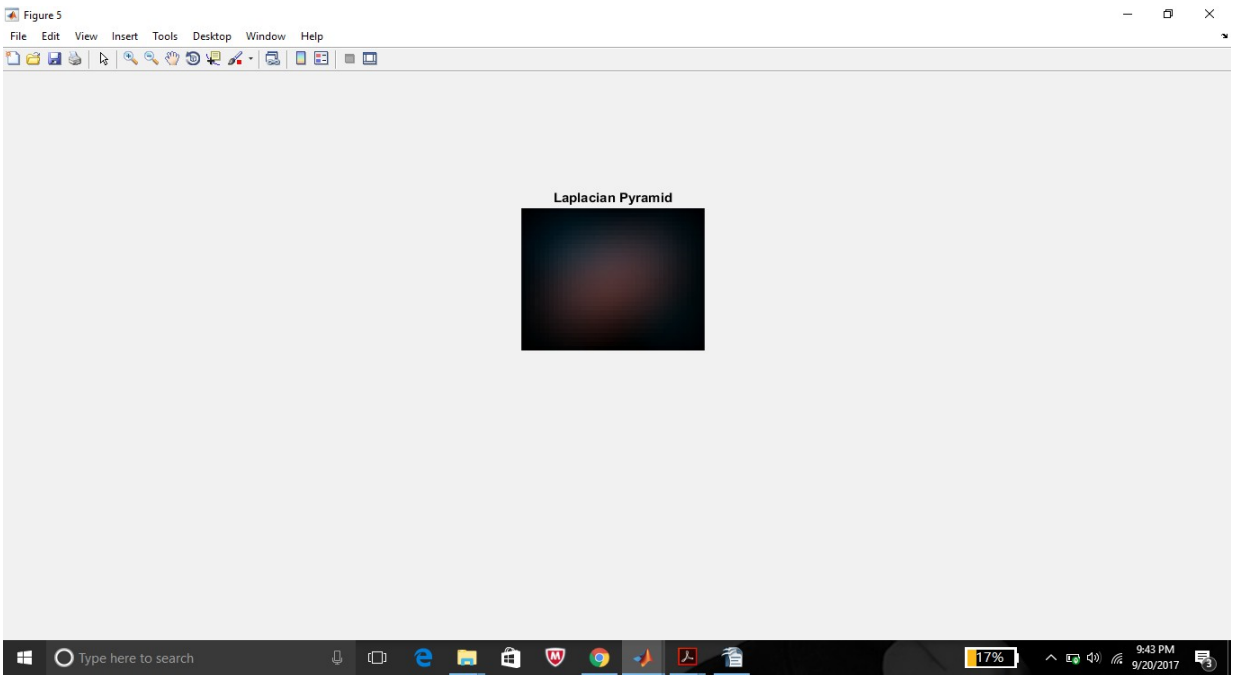
Level 1:



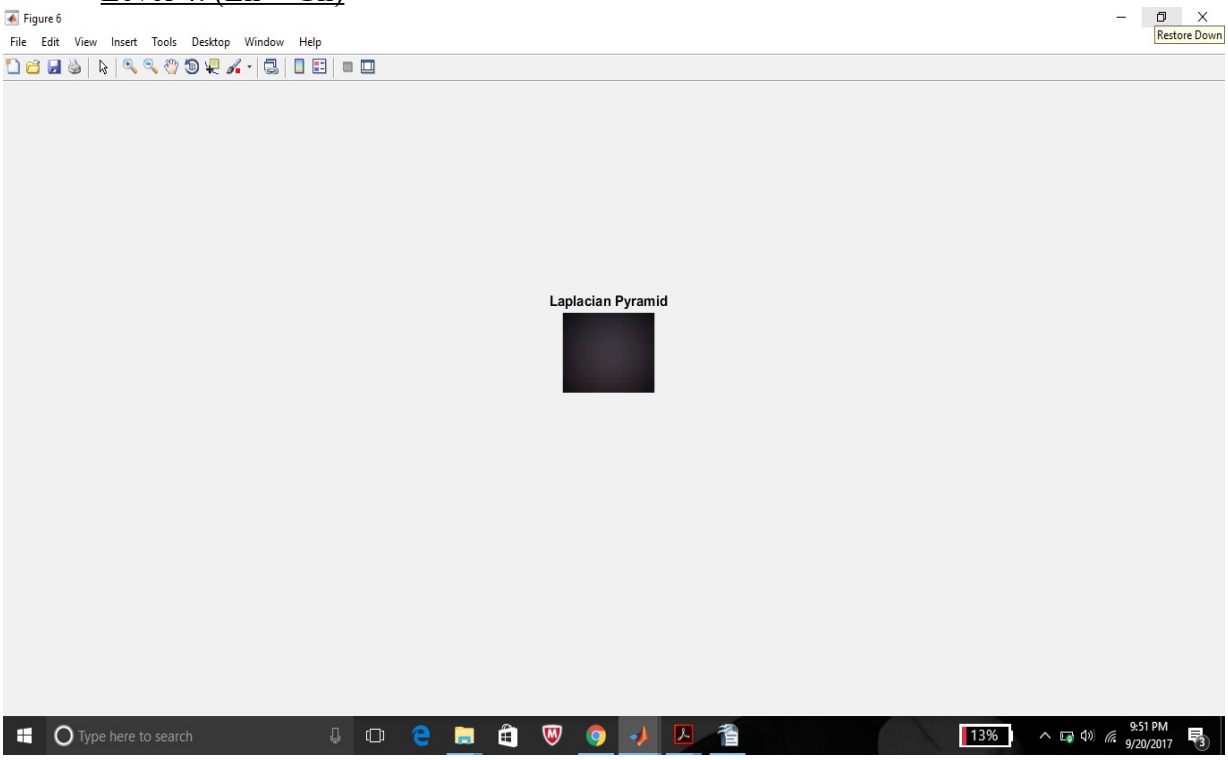
Level 2:



Level 3:



Level 4: (Ln = Gn)

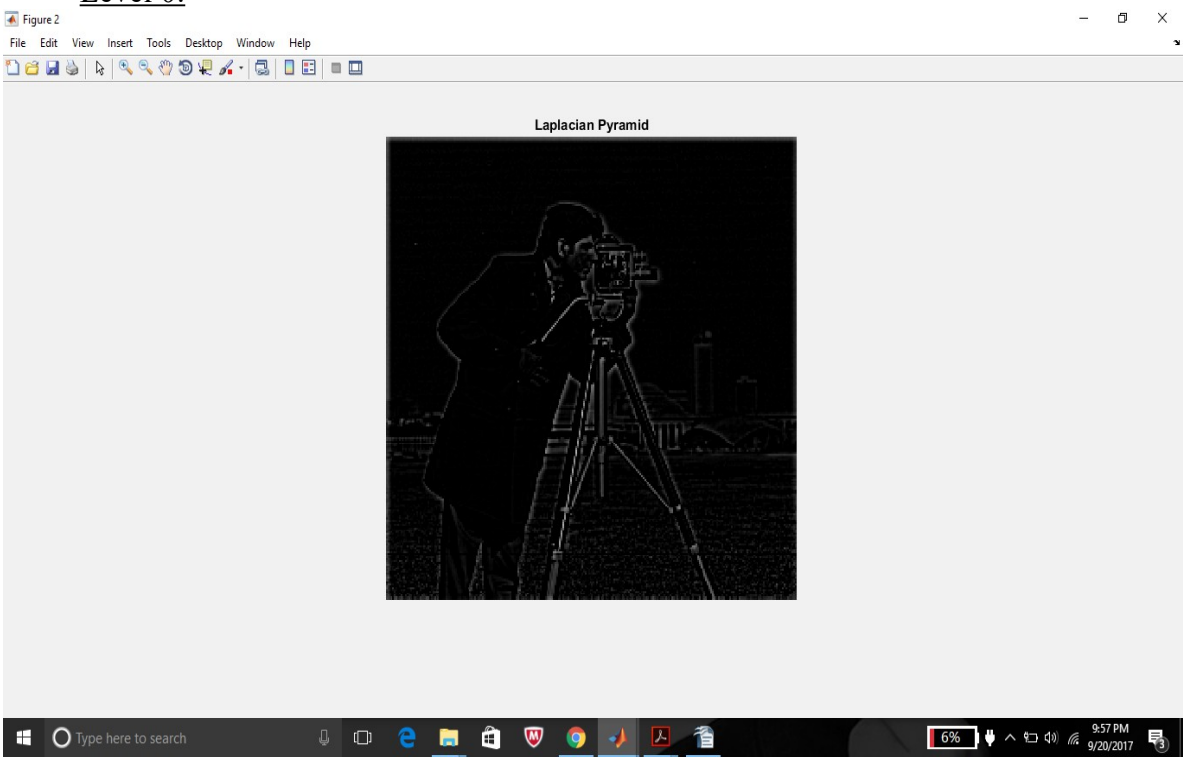


Parameters 3: Size of filter = 5, sigma = 15, number of levels = 3

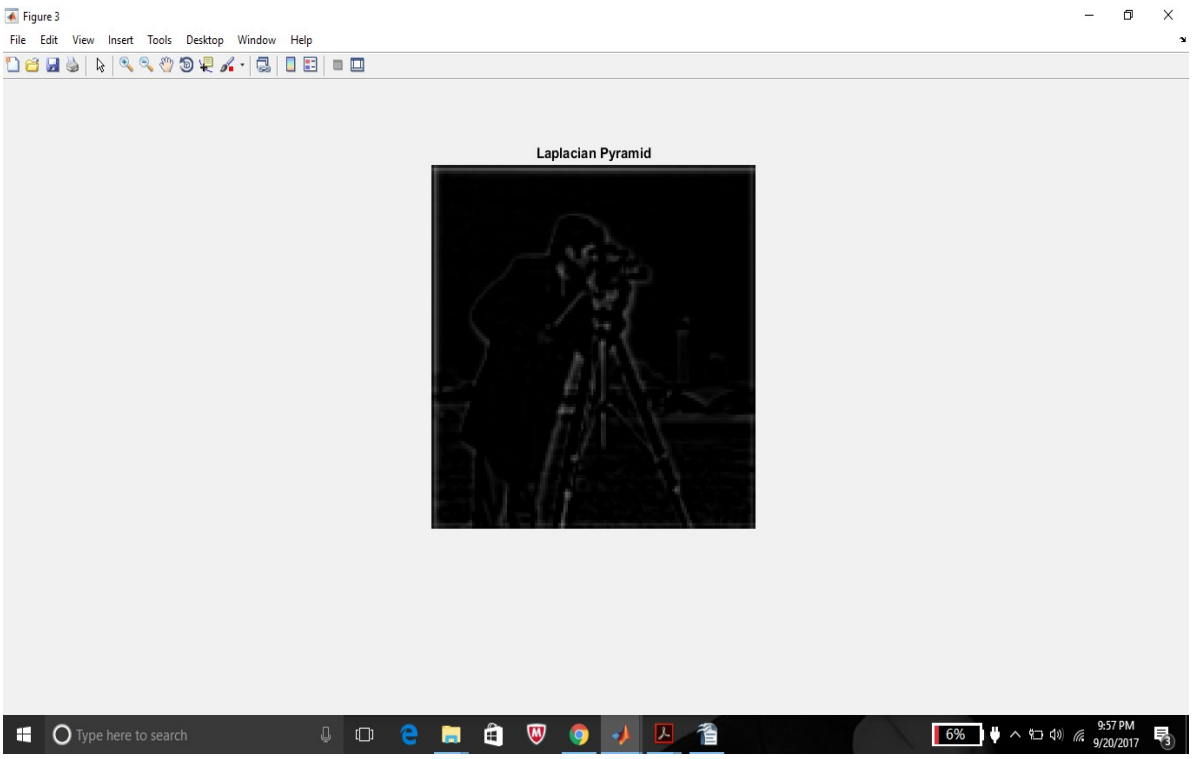
Input Image:

Outputs:

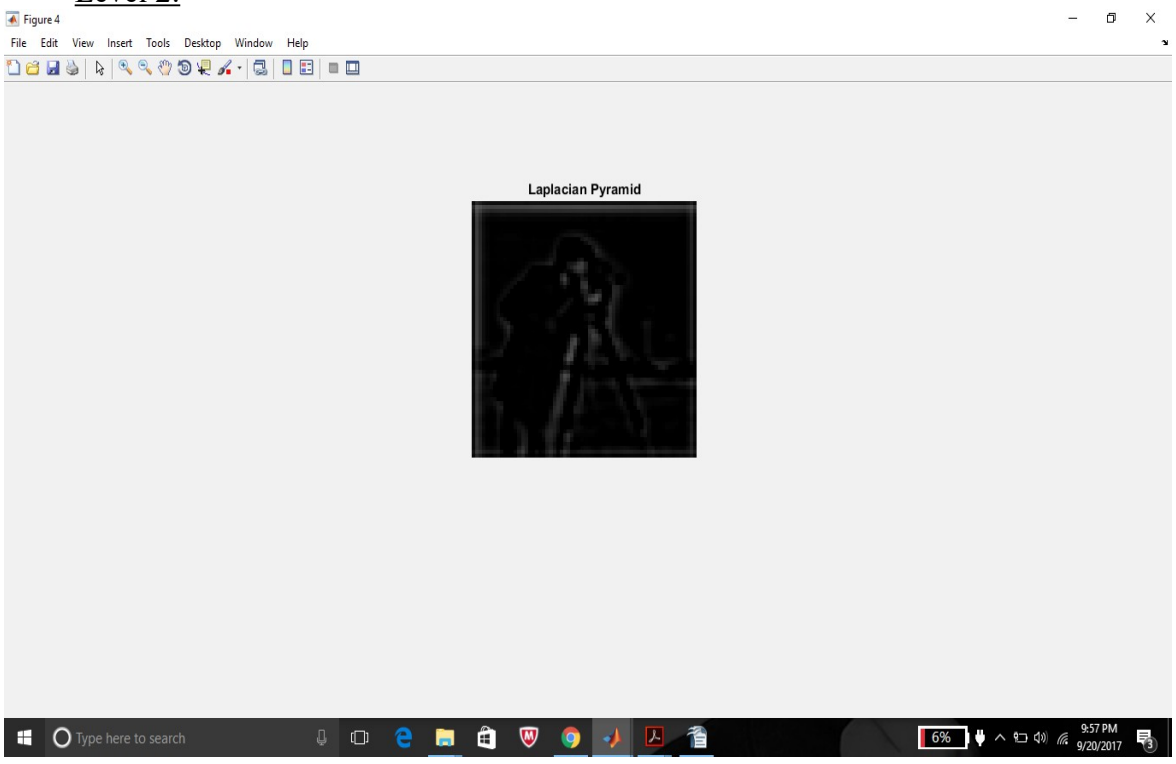
Level 0:



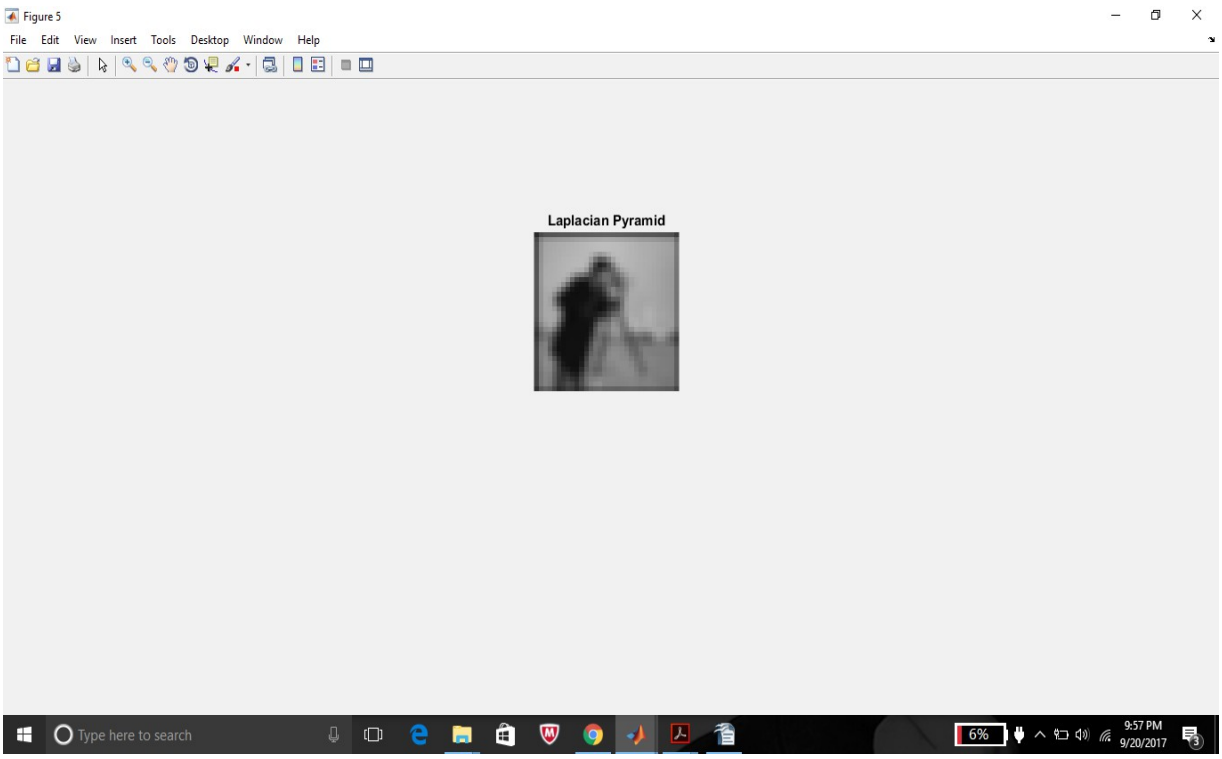
Level 1:



Level 2:



Level 3: Ln = Gn



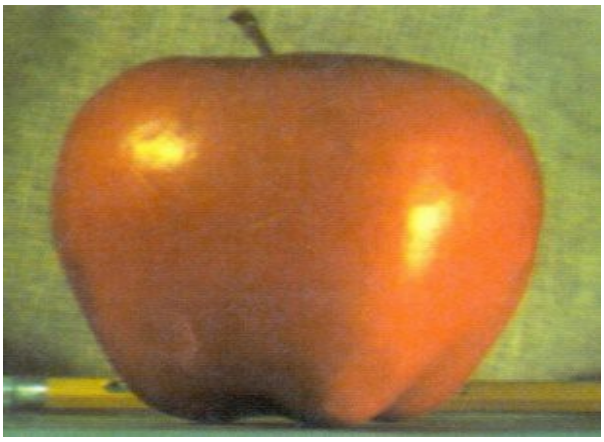
Observations: As sigma and the size of the filter increase, the Laplacian images show more prominent edges due to the higher level of smoothing obtained. Also, as the number of levels increase, the output will have less prominent edges and will be more blurred to the extent that it becomes difficult to understand what is contained in the image.

Results: A relatively high sigma should be used while the size of the filter should not be too large. Also, it is best if we keep the number of levels below 6.

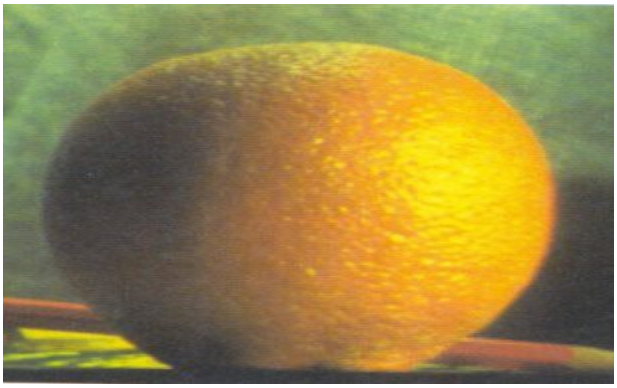
b) Image Blending

Parameters 1: Size of filter = 20, sigma = 50, levels = 4

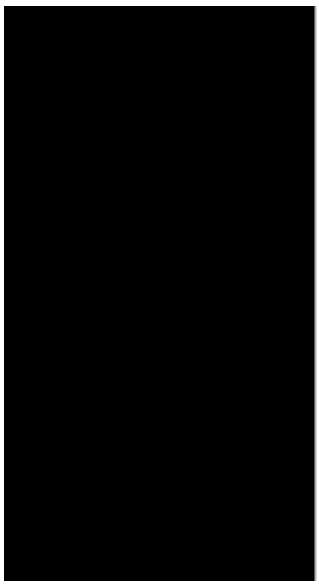
Input Image 1:



Input Image 2:

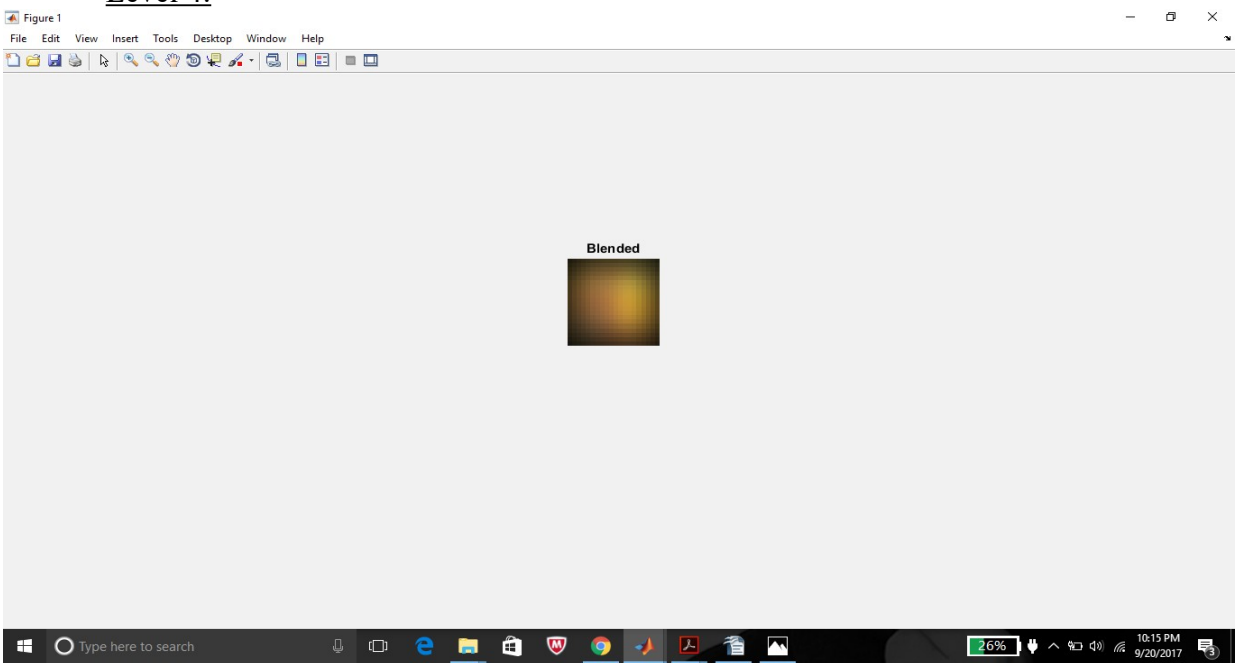


Mask:



Outputs:

Level 4:

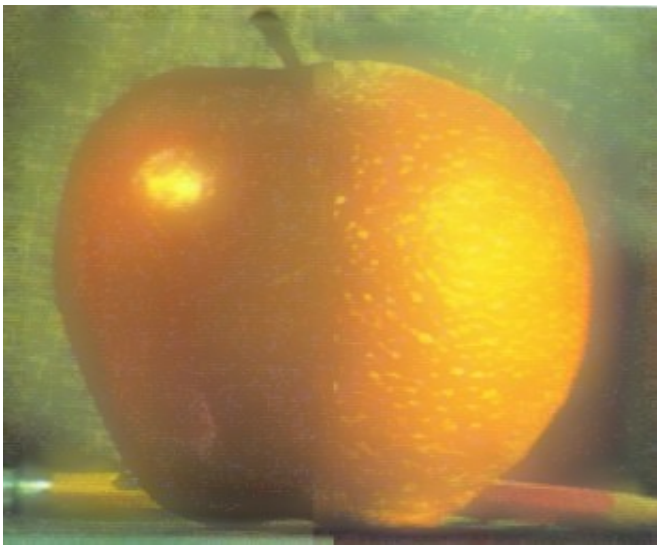


Level 3:

Level 2:

Level 1:

Level 0:

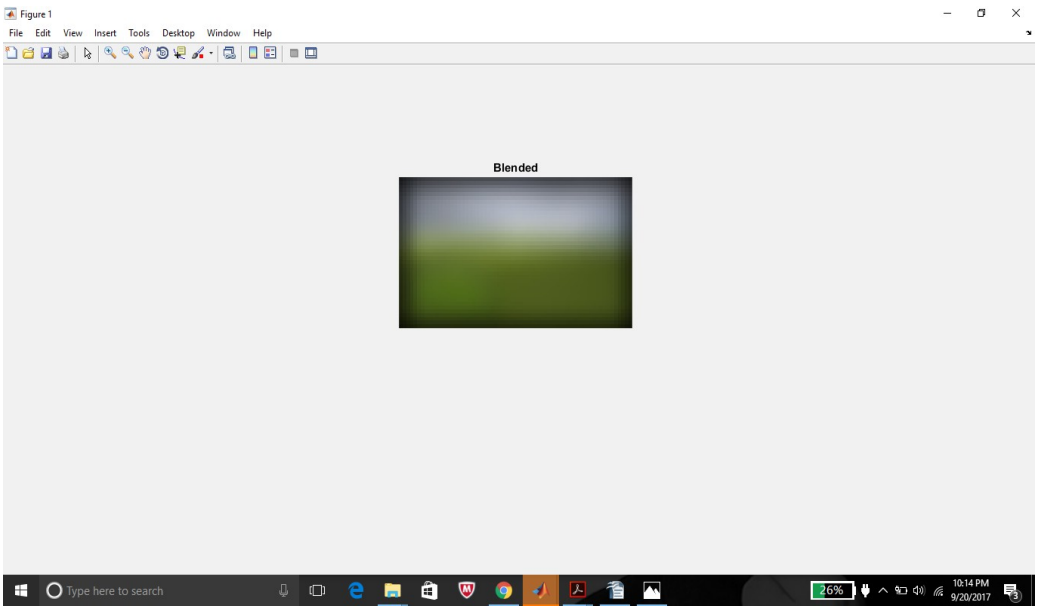


Parameters 2: Size of filter = 20, sigma = 20, number of levels = 4
Input Image 1:

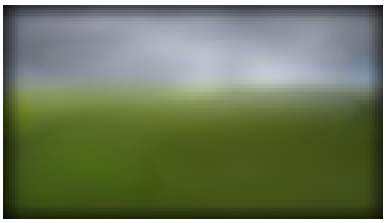
Input Image 2:

Mask:

Outputs:
Level 4:



Level 3:



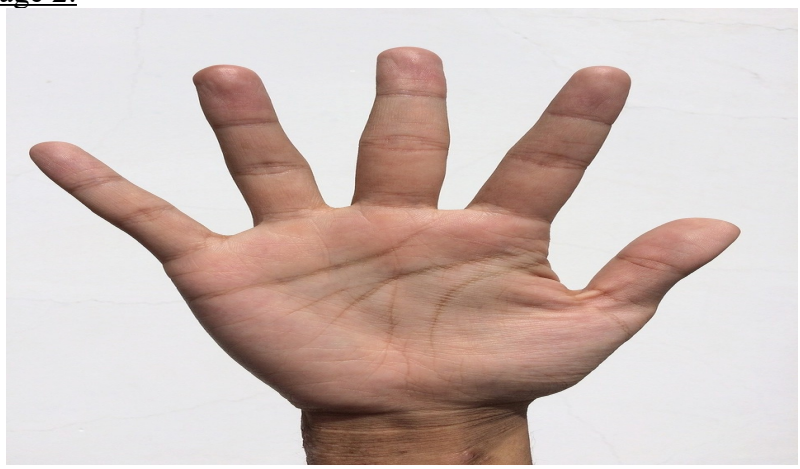
Level 2:

Level 1:

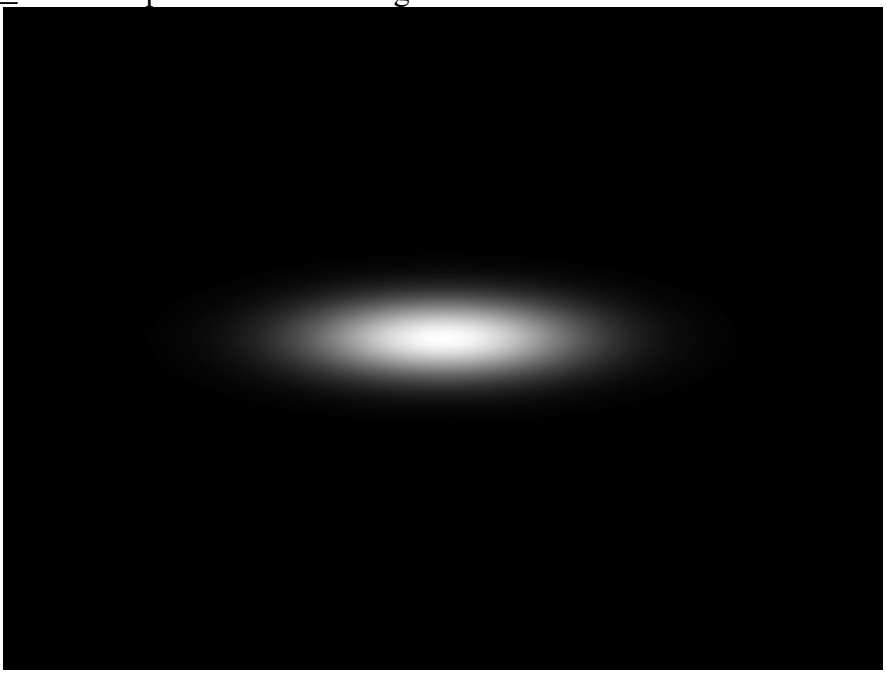
Level 0:

Parameters 3: Size of filter = 10, sigma = 10, number of levels = 3
Input Image 1:

Input Image 2:

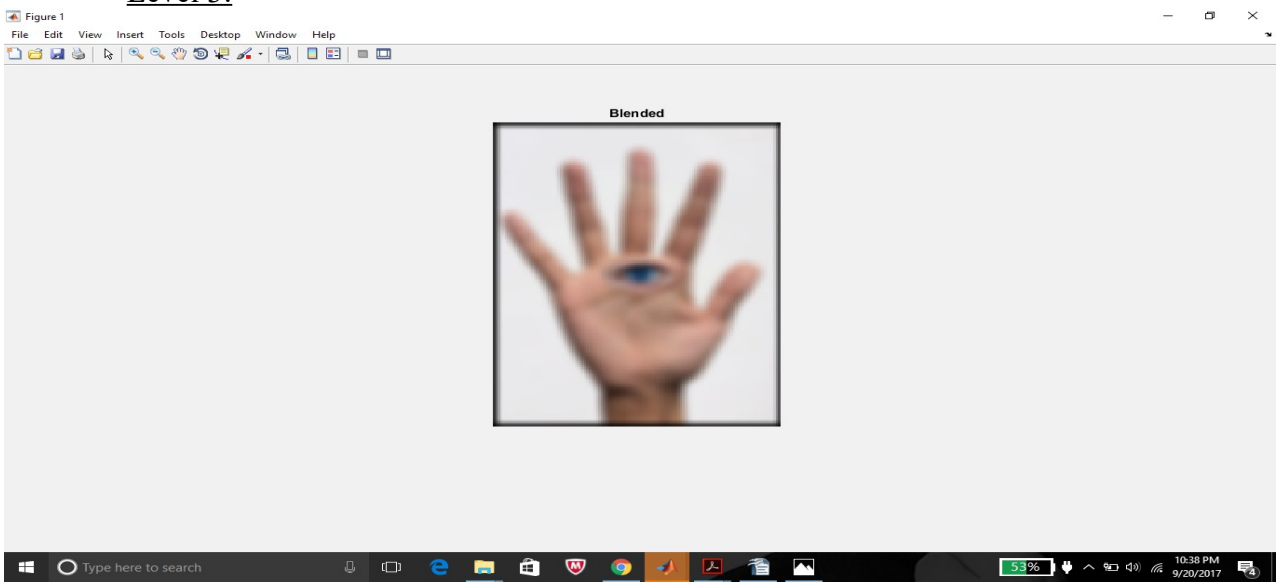


Mask: Used complement of this image as the mask.

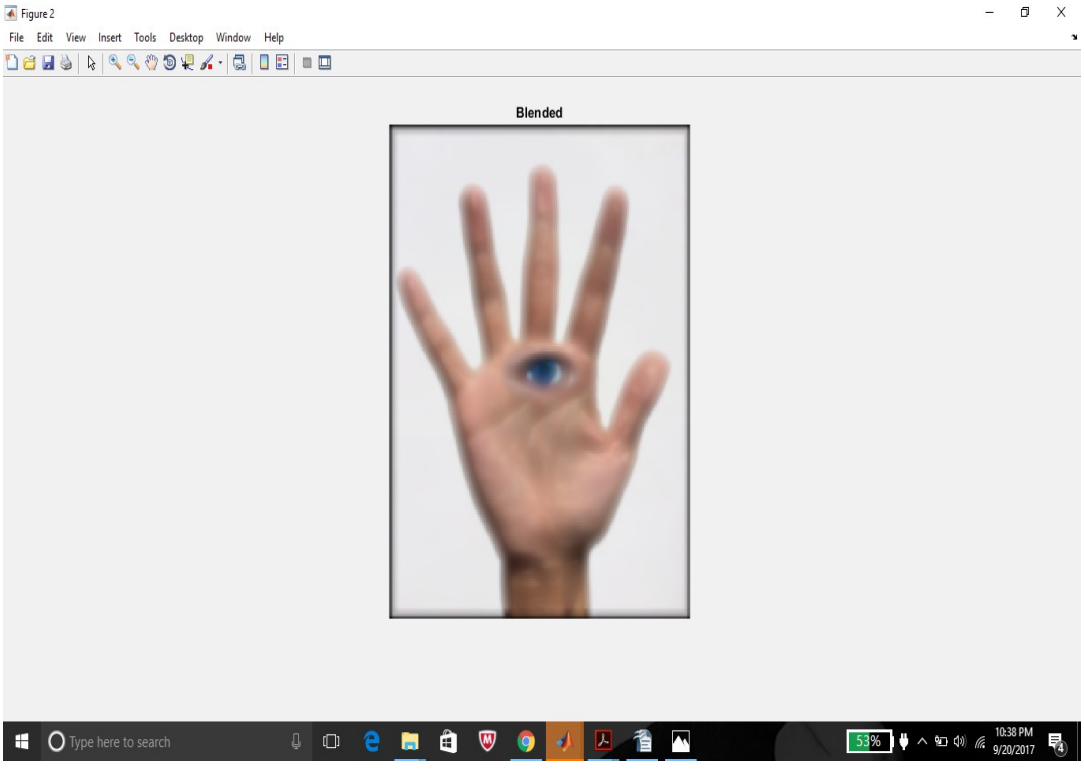


Outputs:

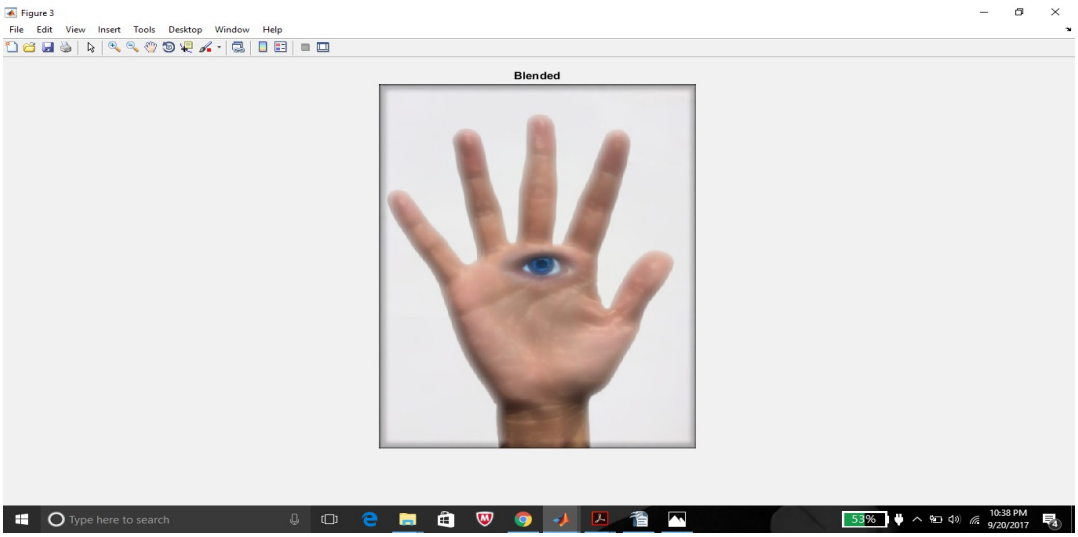
Level 3:



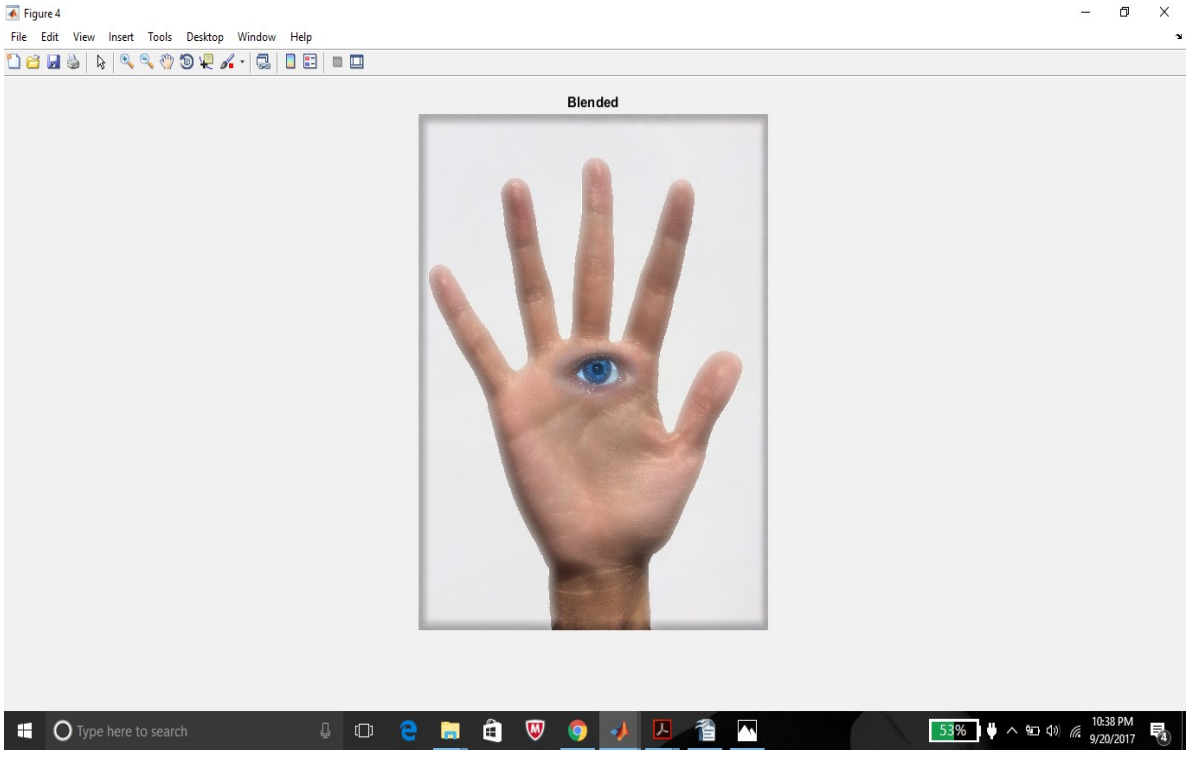
Level 2:



Level 1:



Level 0:



Observations: In order to blend the two images to a larger degree, a larger filter and a higher sigma value should be used. I could not see any large difference if more number of levels were used.

Results: Use an appropriate filter and sigma value. In my opinion, smaller values should be used if they give satisfactory results but if the output does not appear blended, then only should larger values be used. Also, it is better to use less than 10 levels.

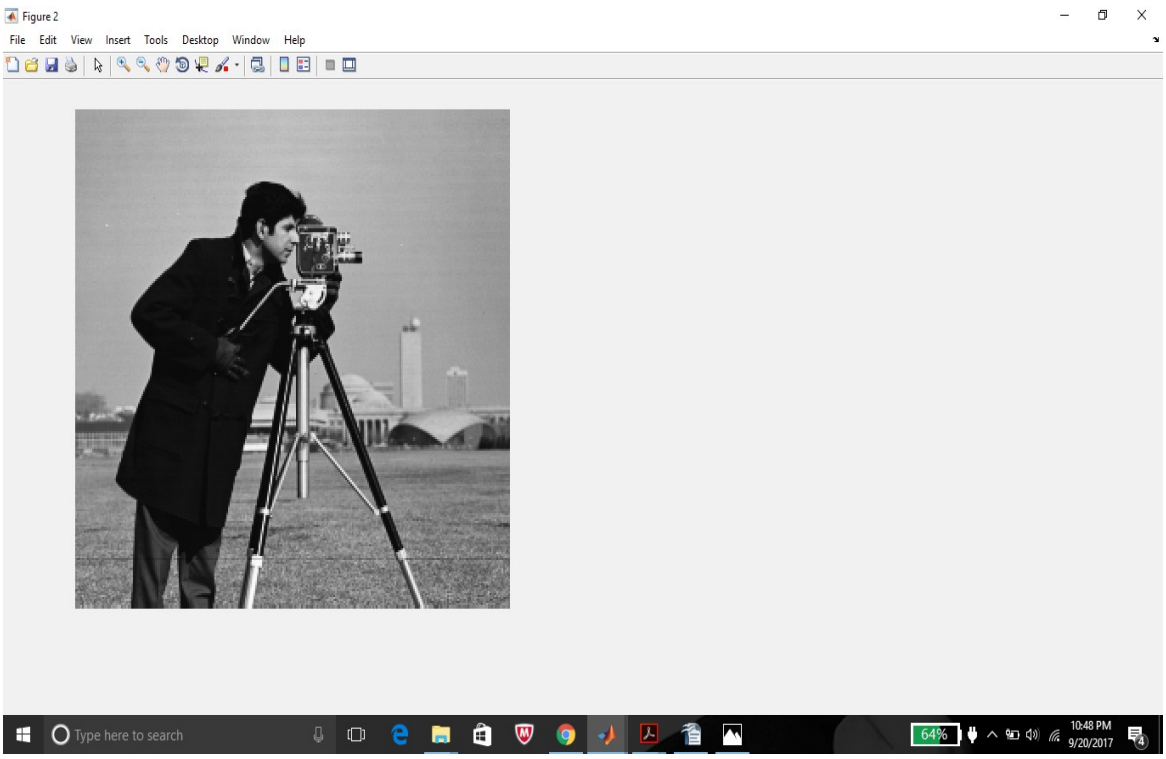
c) Image Upsampling

Nearest Neighbour:

Input Image 1: Number of levels = 3

Output Image 1a: Factor 2

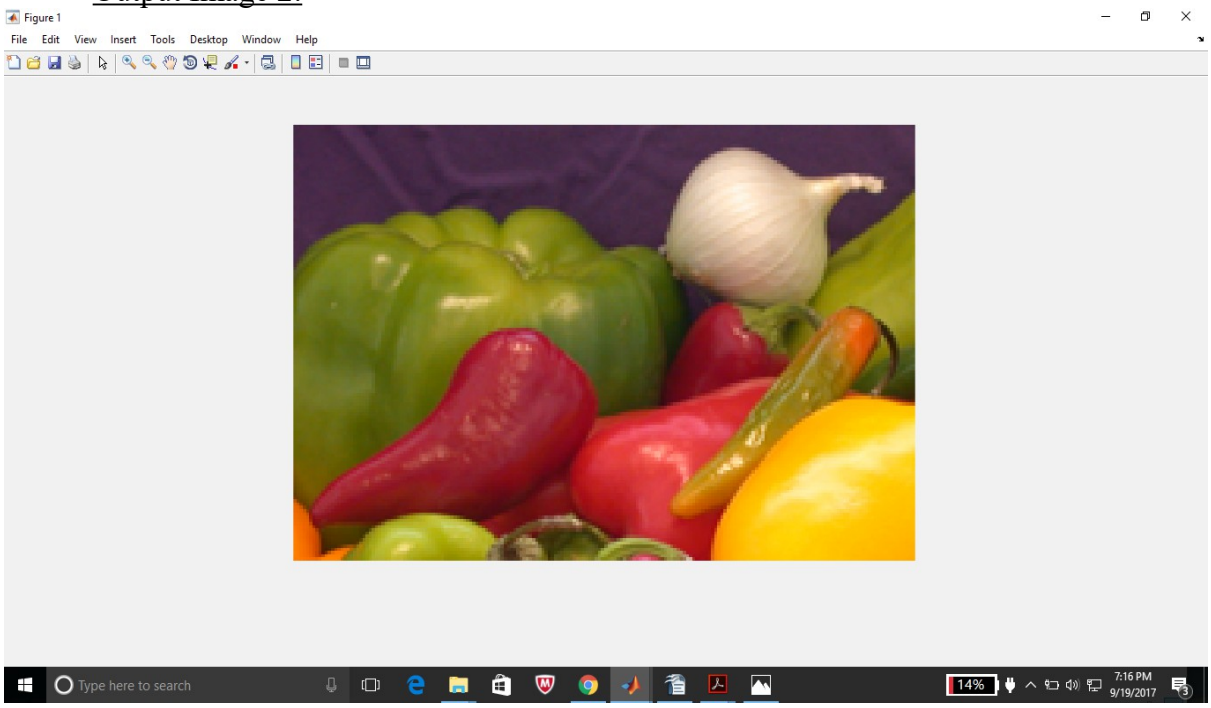
Output Image 1b:



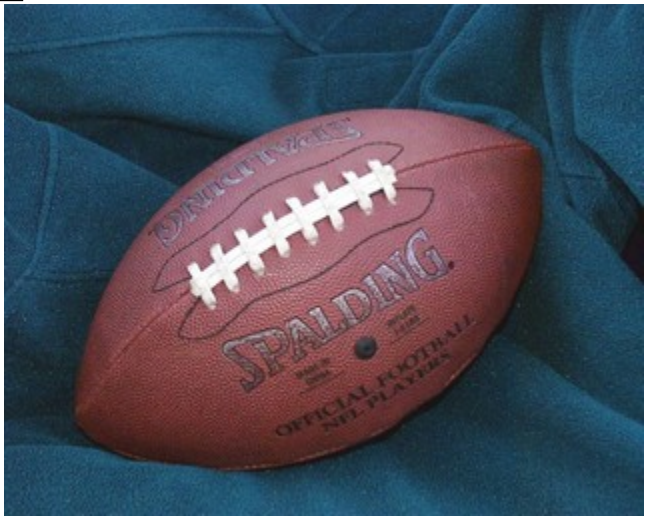
Output Image 1c:

Input Image 2: Number of levels = 1

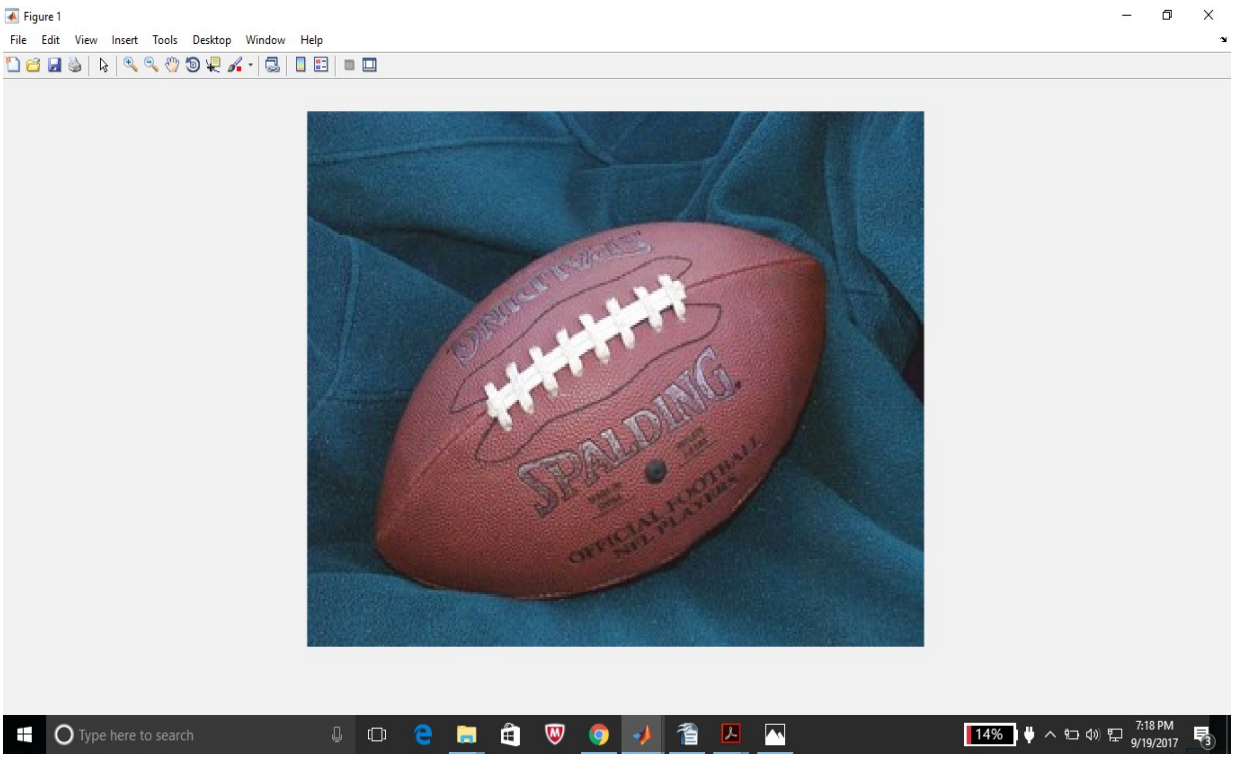
Output Image 2:



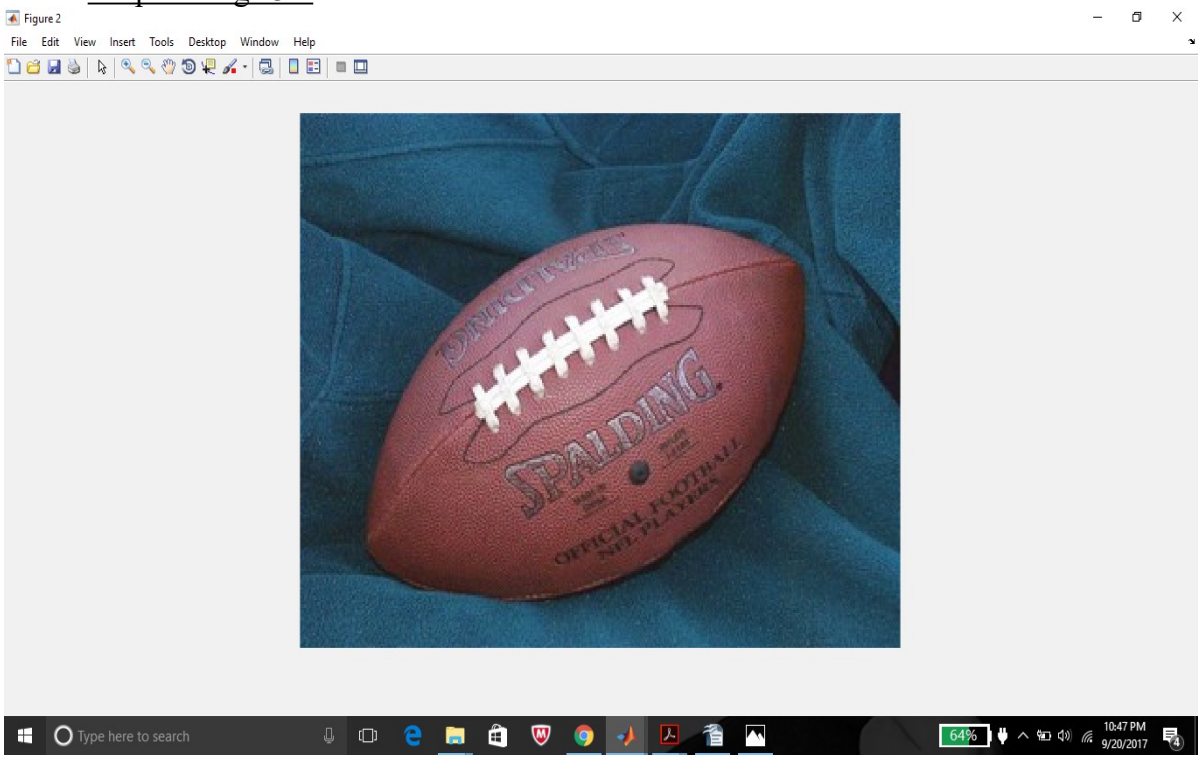
Input Image 3: Number of levels = 2



Output Image 3a:



Output Image 3b:



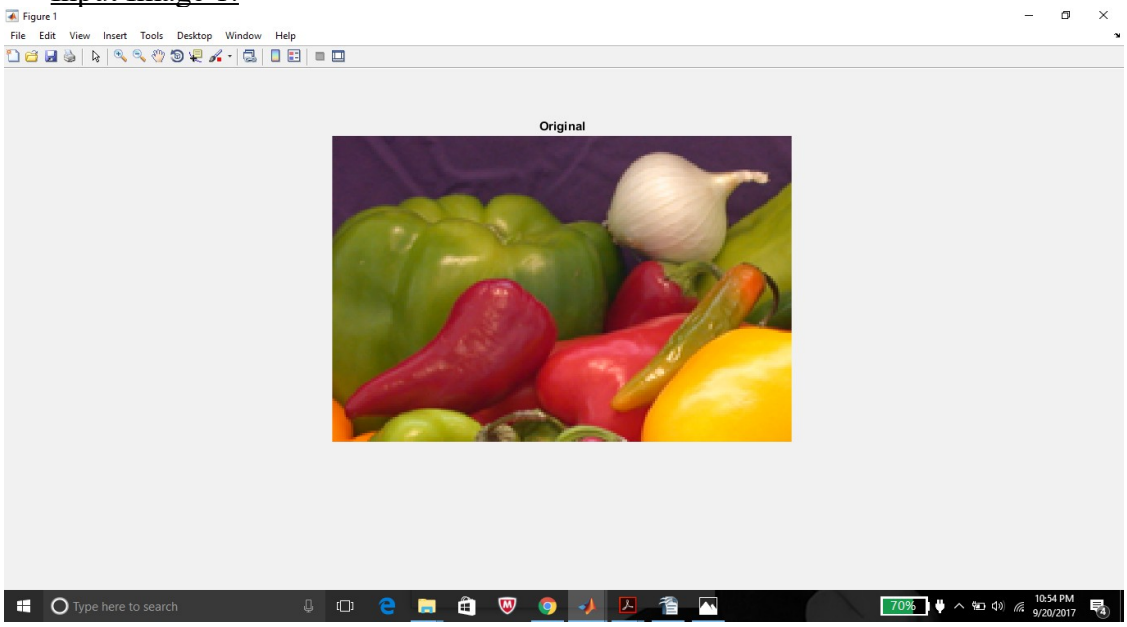
Observations: This interpolation gives fast results but the results are not the best as it just repeats neighbouring pixels. The image quality degrades with increase in *n*.

Results: For the images shown, the results seem to be satisfactory. Also, considering the speed of computation, this is the fastest implementation of upsampling.

Linear:

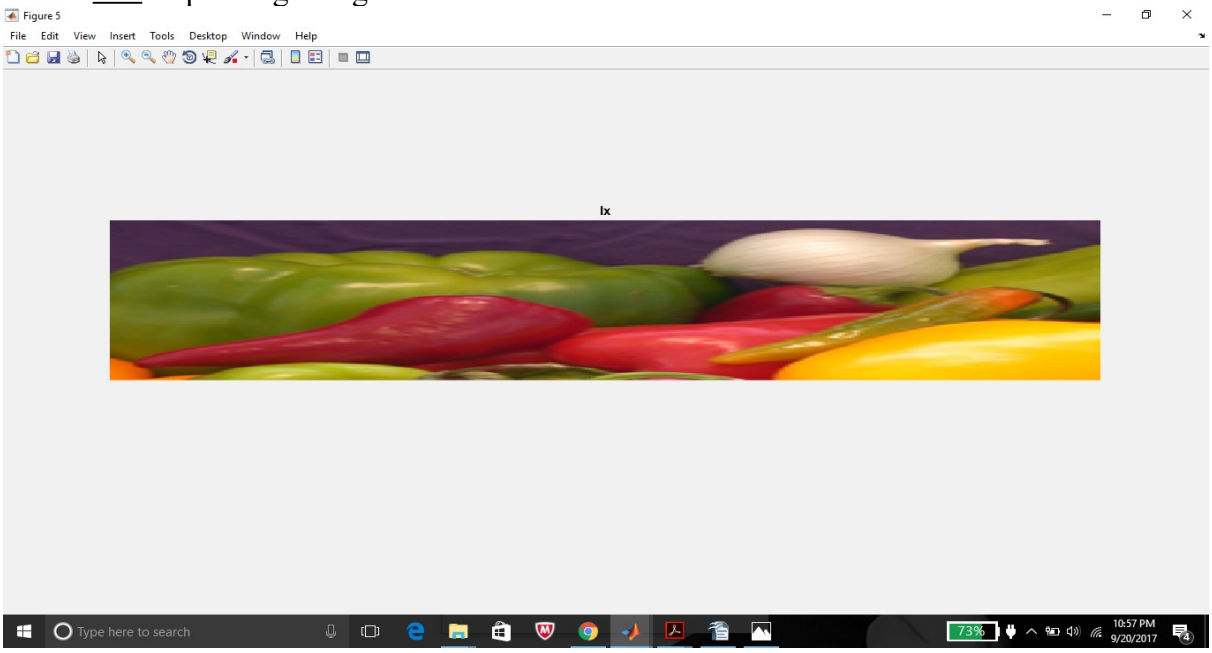
Parameters 1: Number of levels = 2

Input Image 1:

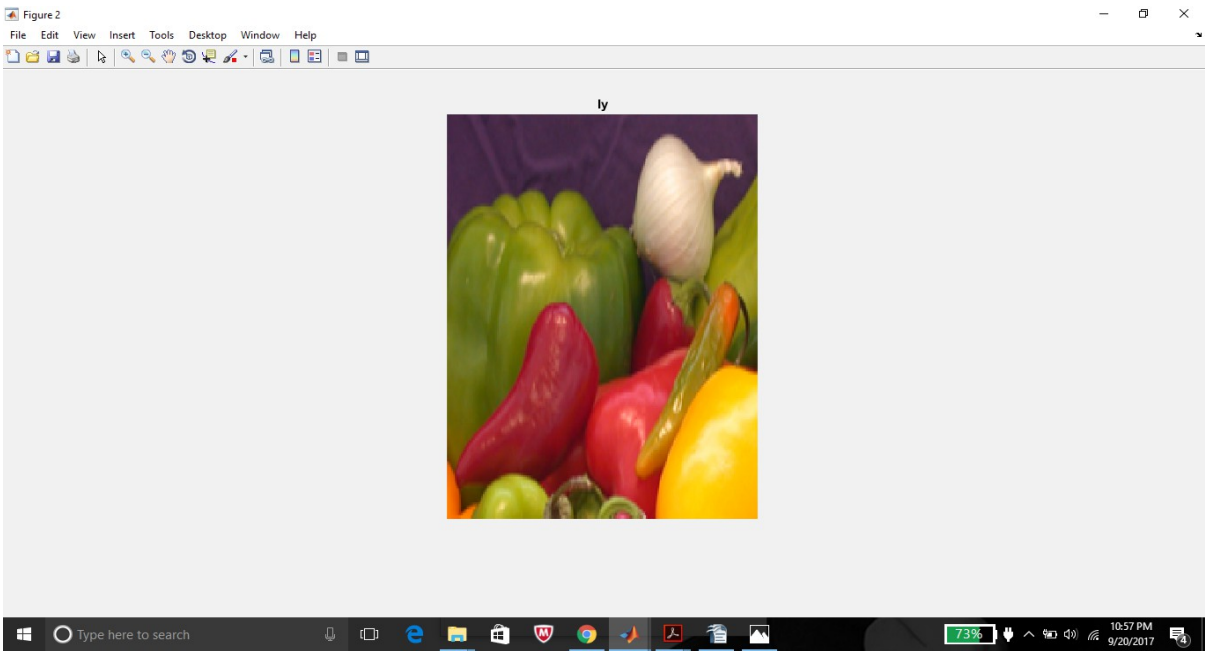


Output Images:

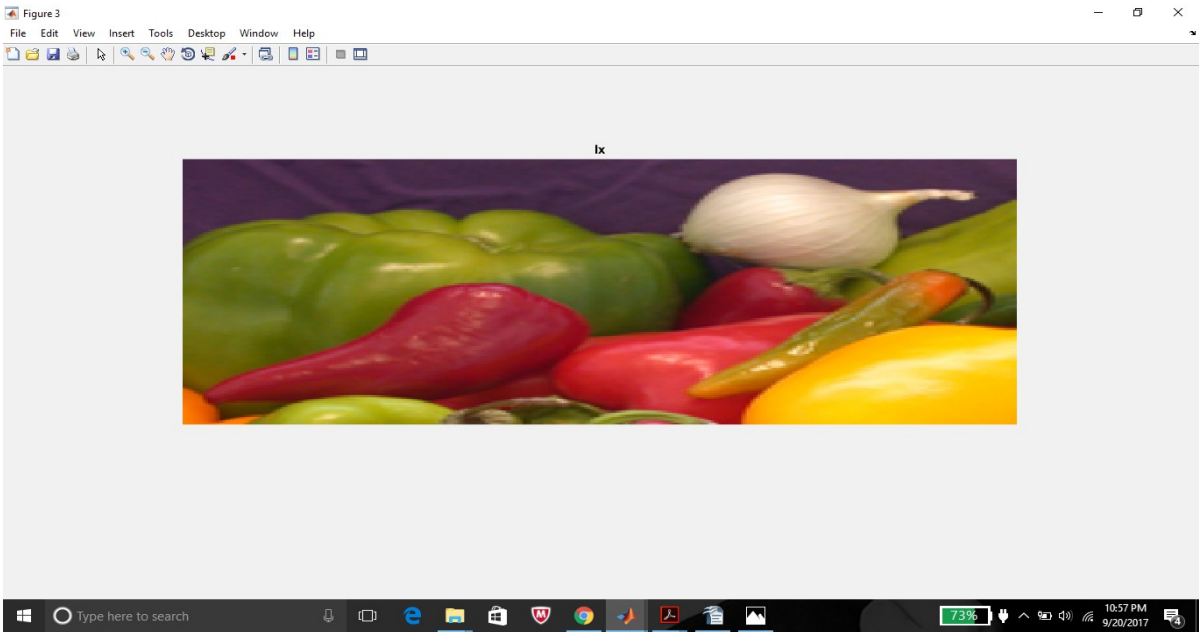
Ixa: Expanding along x direction



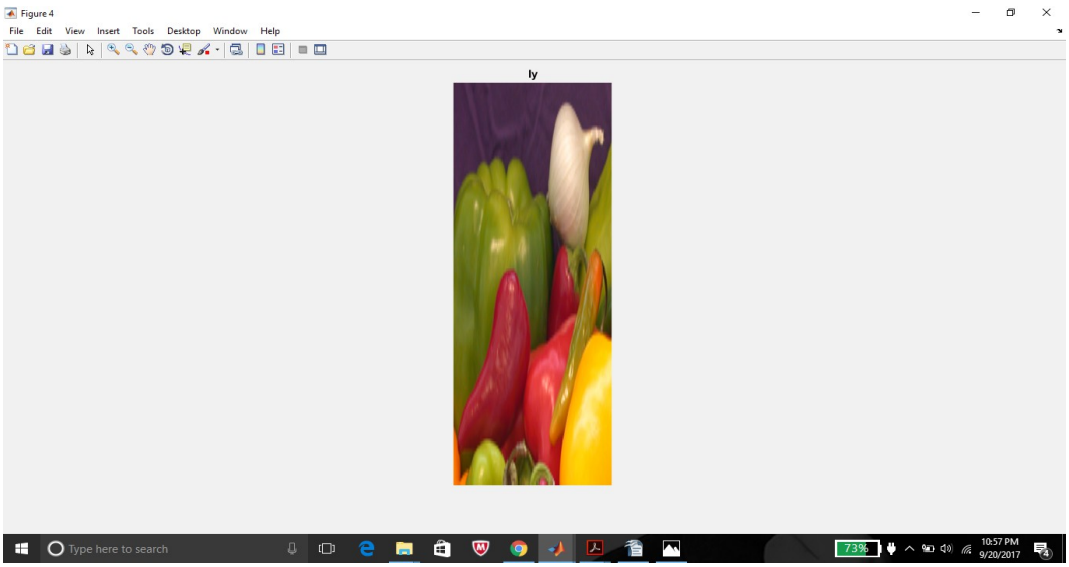
Iya: Expanding along y direction



Ixb : Expanding Ixa along x direction



Iyb: Expanding Iya along y direction

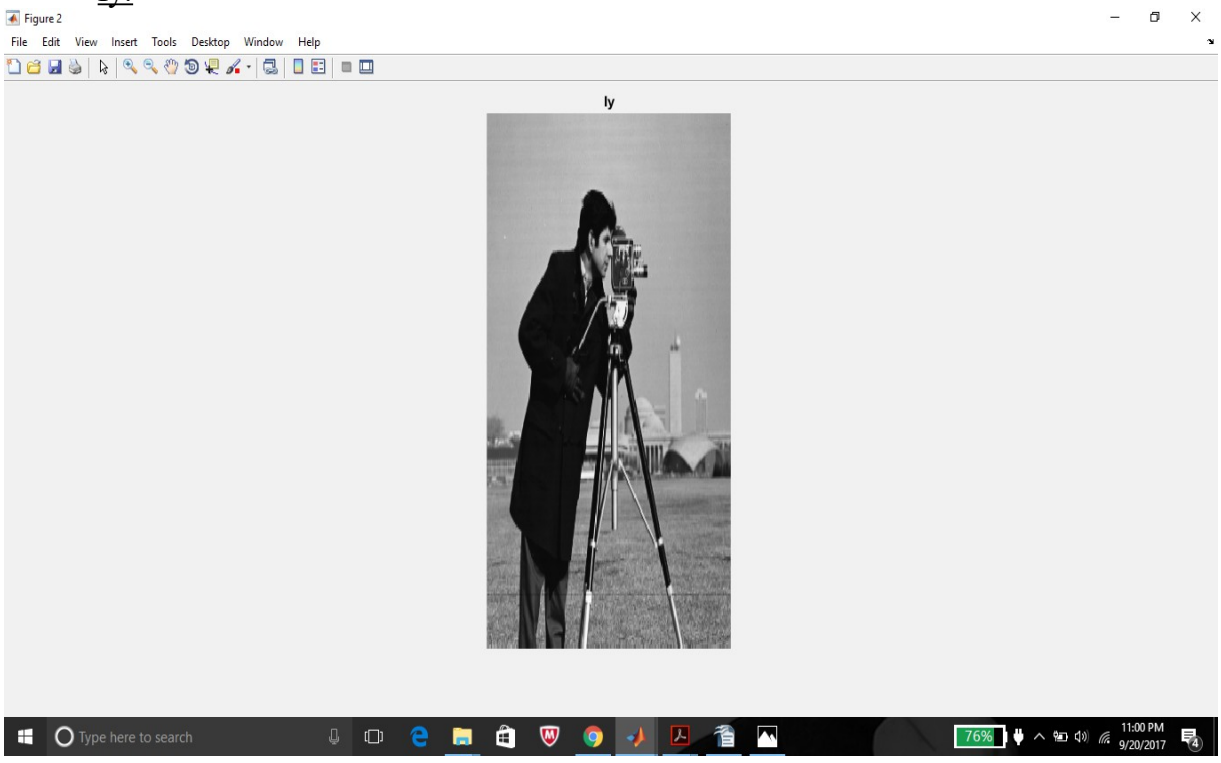


Parameters 2: Number of levels = 1

Input Image:

Ix:

Iy:



Parameters 3: Number of levels = 1

Input Image:

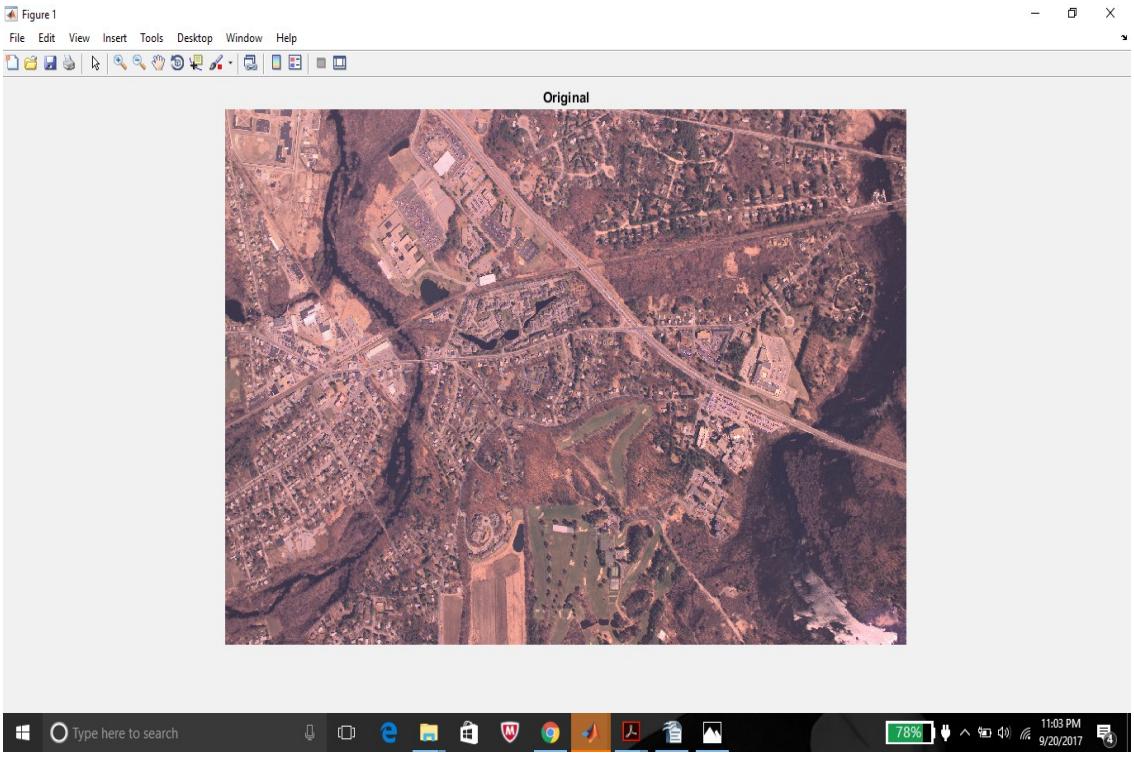


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Type here to search
78%
11:03 PM
9/20/2017

Ix:

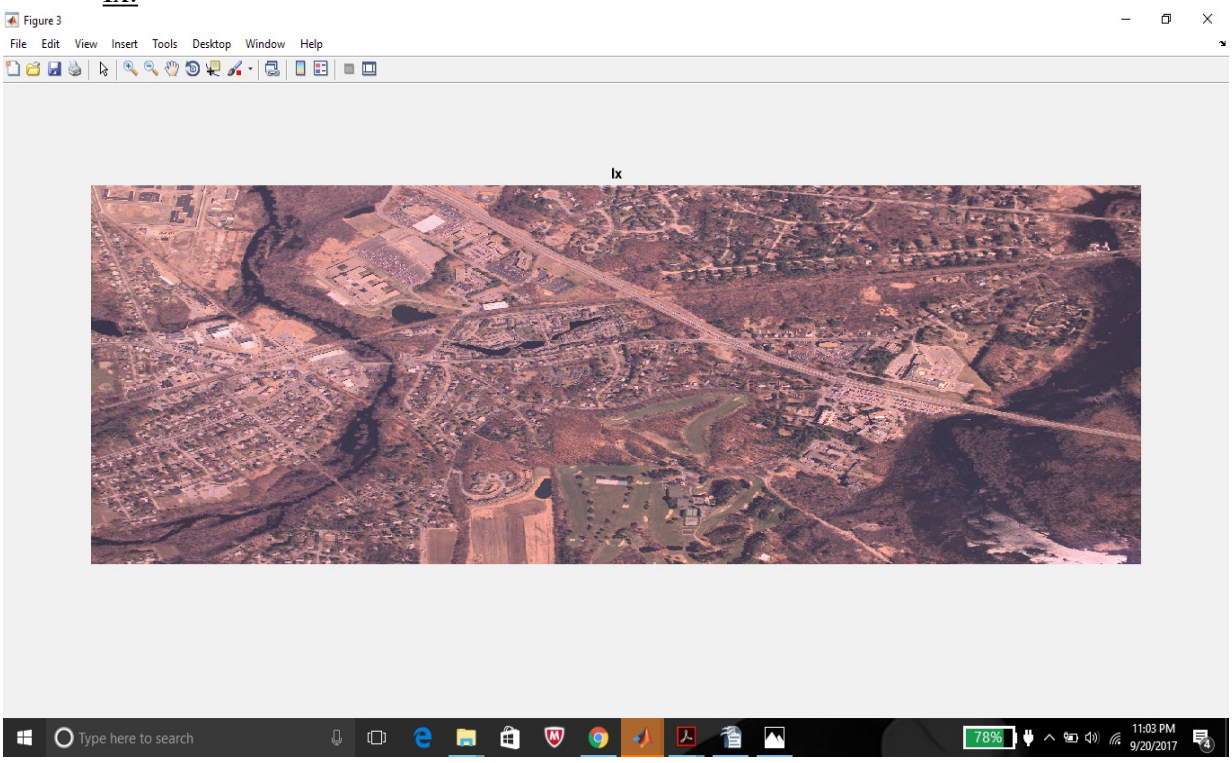
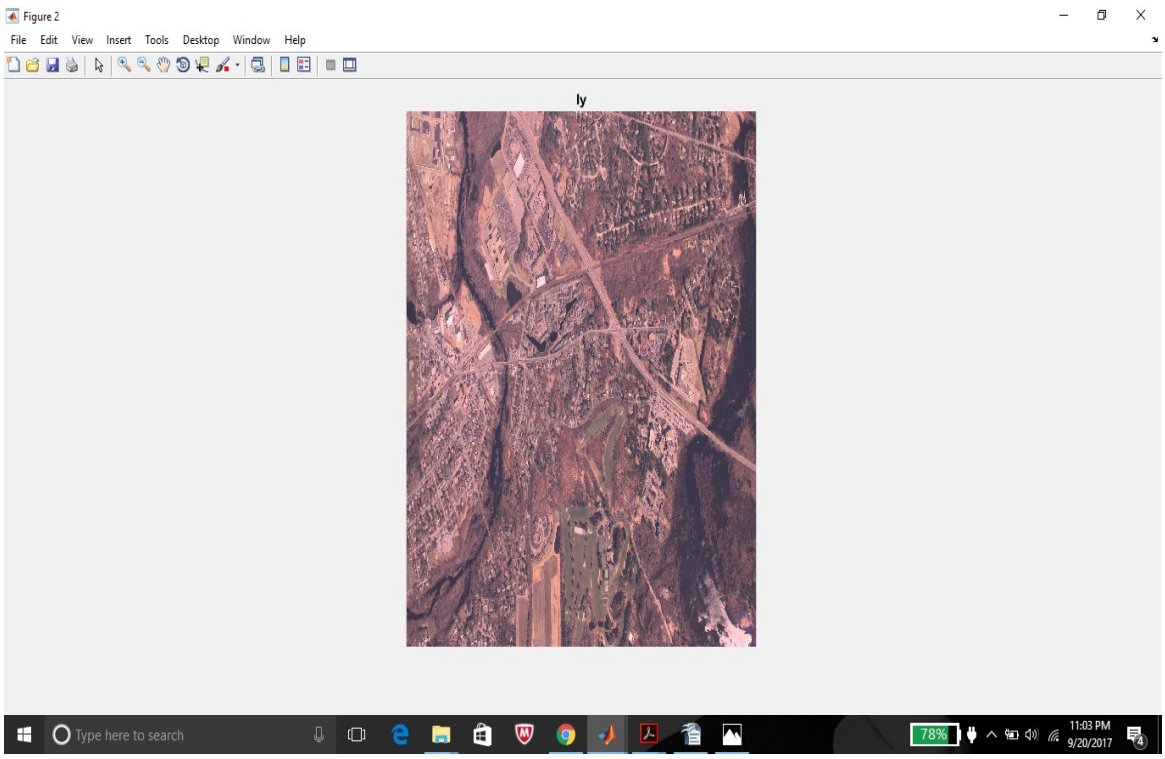


Figure 3
File Edit View Insert Tools Desktop Window Help
Ix
Type here to search
78%
11:03 PM
9/20/2017

Iy:



Observations: We can see that as we increase *n* (number of times the image is upsampled), the output degrades in quality. However, compared to nearest neighbour interpolation, this gives slightly improved results. Note that either the height or the width is scaled in this case since if they are both simultaneously scaled, the implementation becomes that of bilinear interpolation.

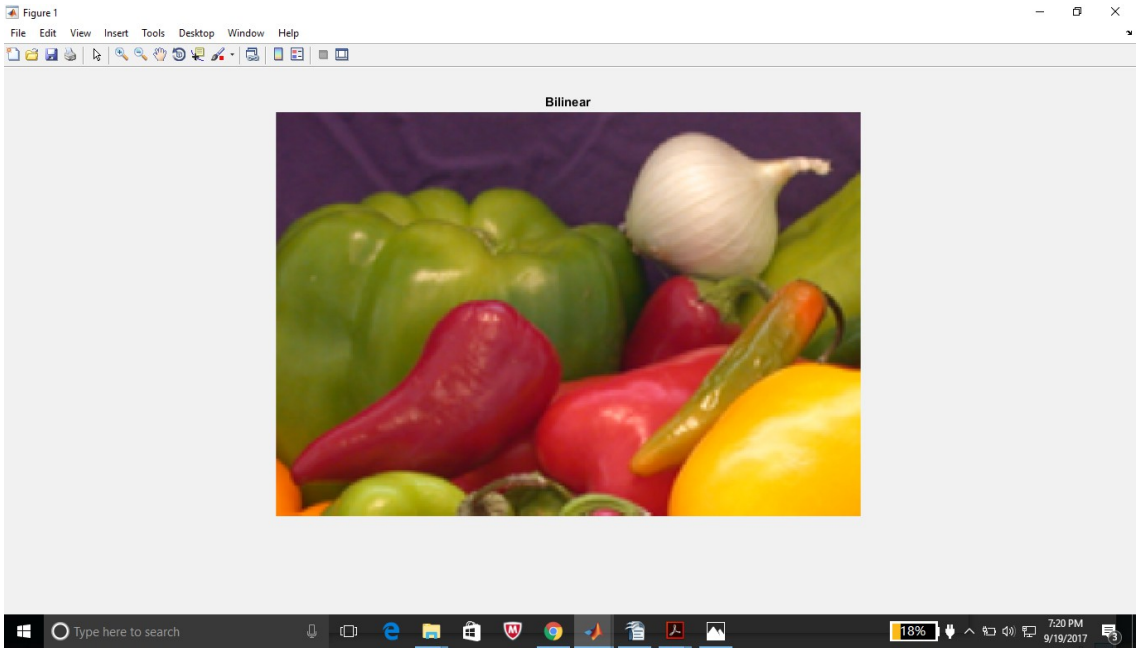
Results: If we keep the factor and *n* small, this gives satisfactory results. This seems to give odd results, though, as the image is stretched only in one direction.

Bilinear:

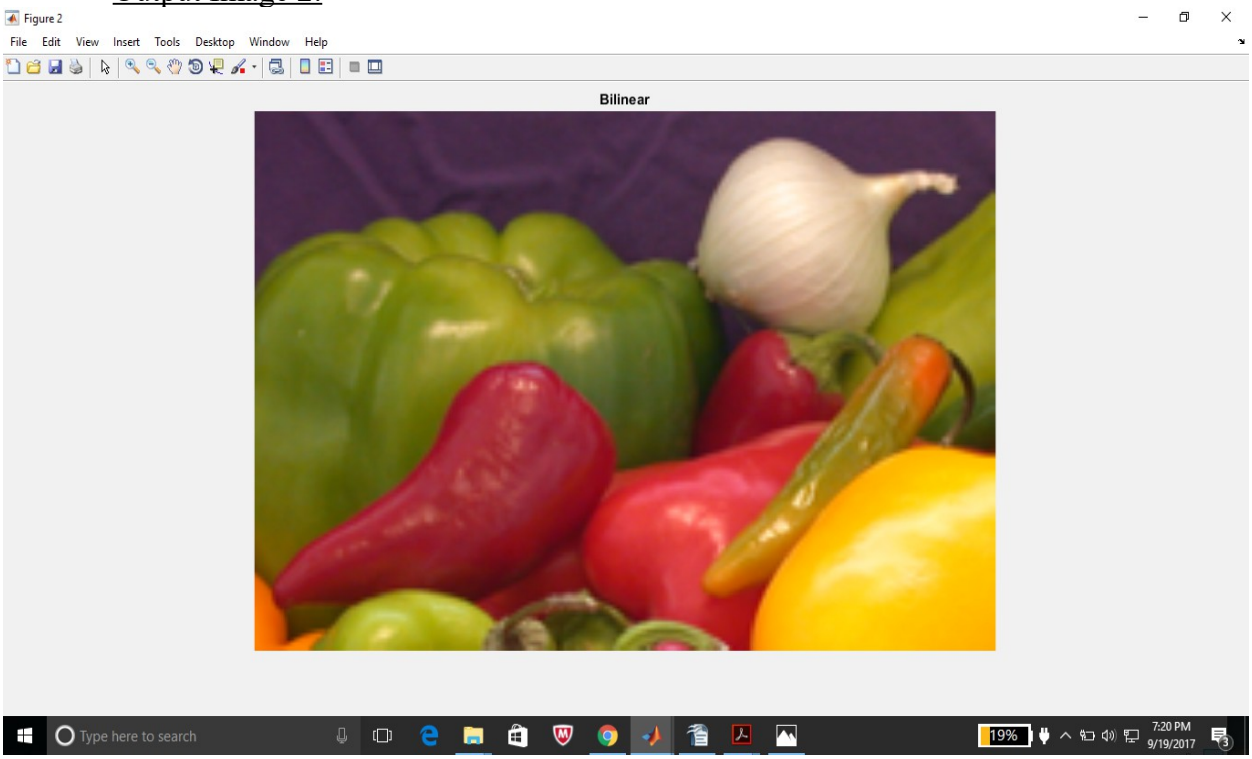
Parameters 1: Levels = 2

Input Image 1:

Output Image 1:



Output Image 2:



Parameters 2: Levels = 3

Input Image:

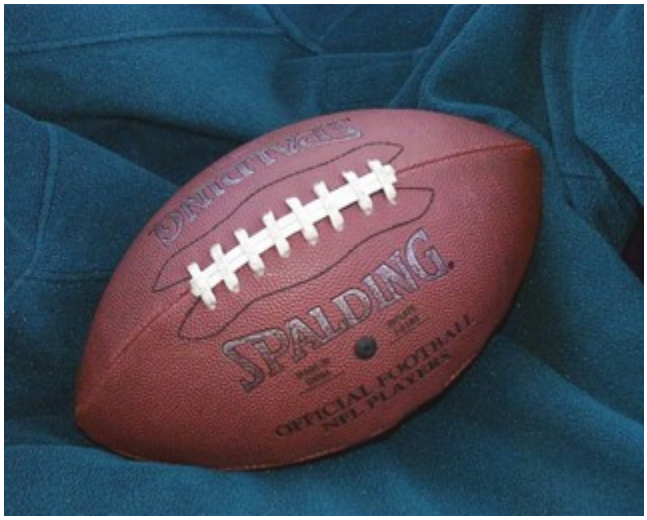
Output Image 1:

Output Image 2:

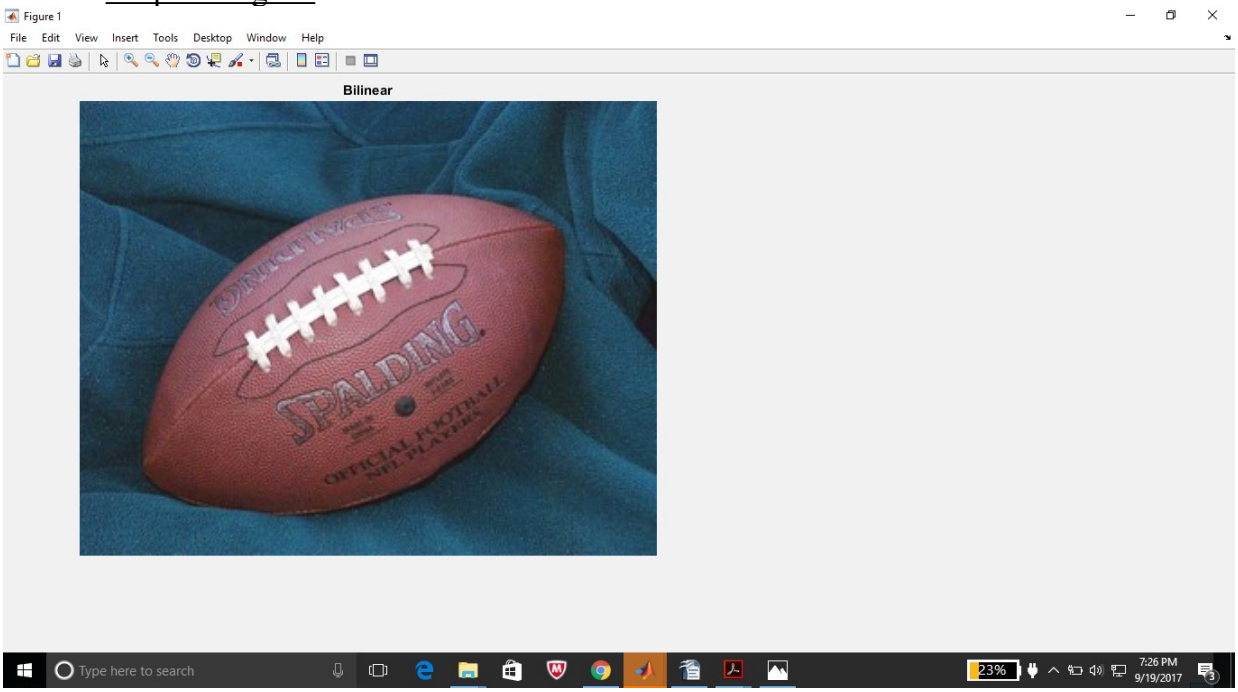
Output Image 3:

Parameters 3: Levels = 4

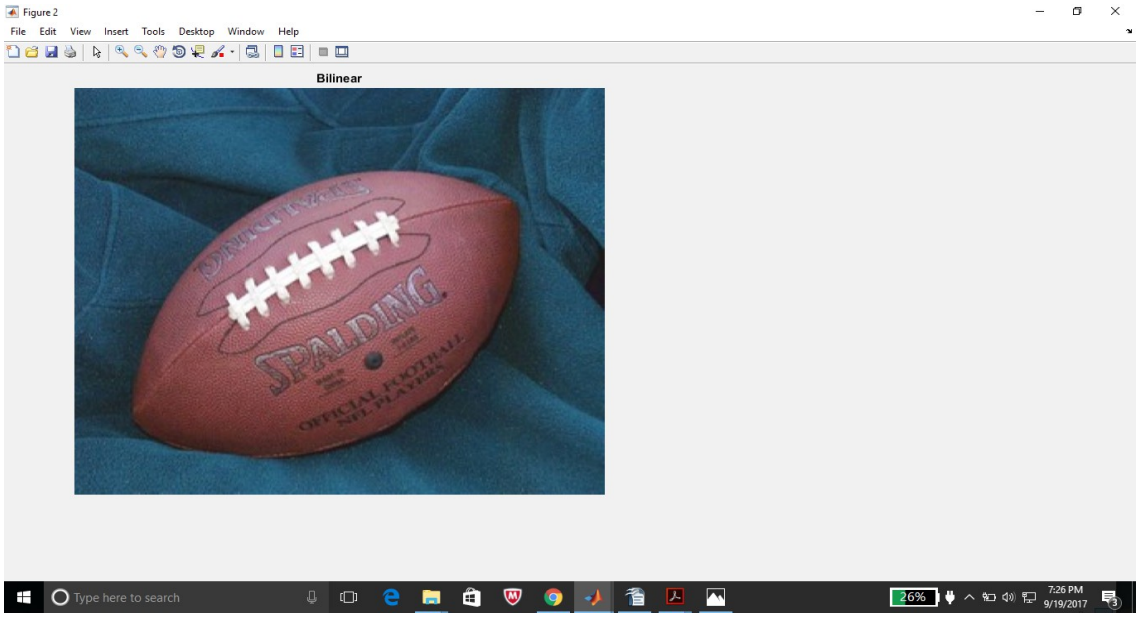
Input Image :



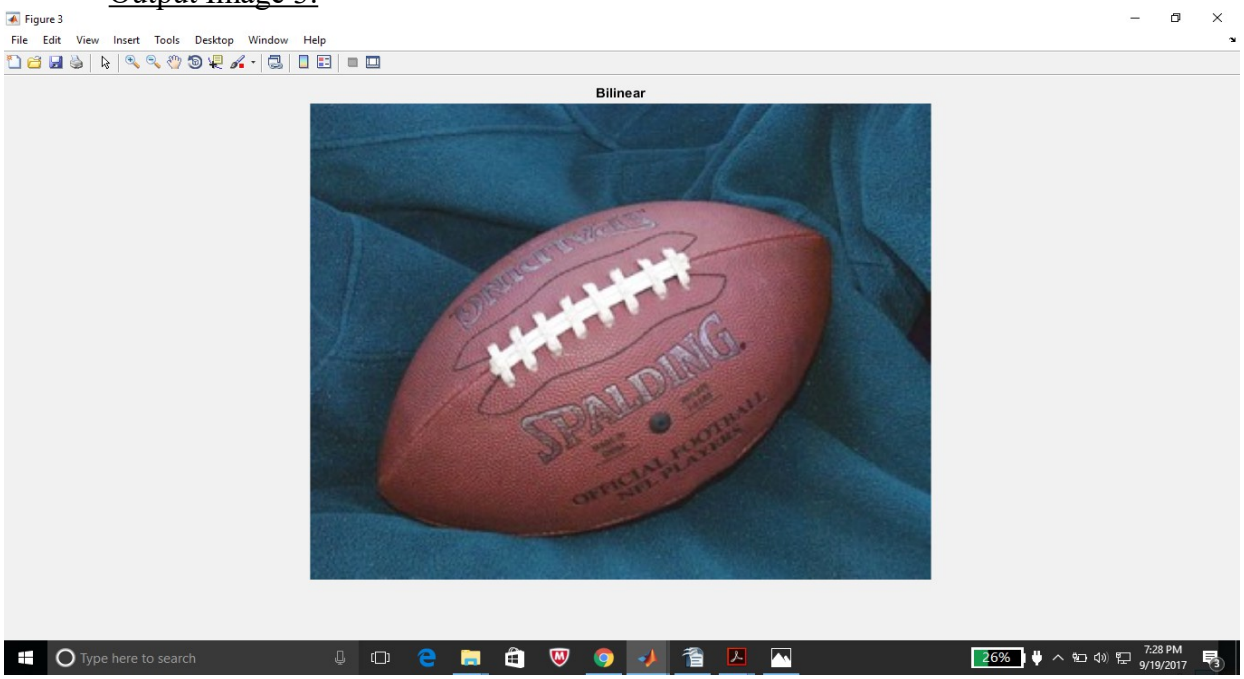
Output Image 1:



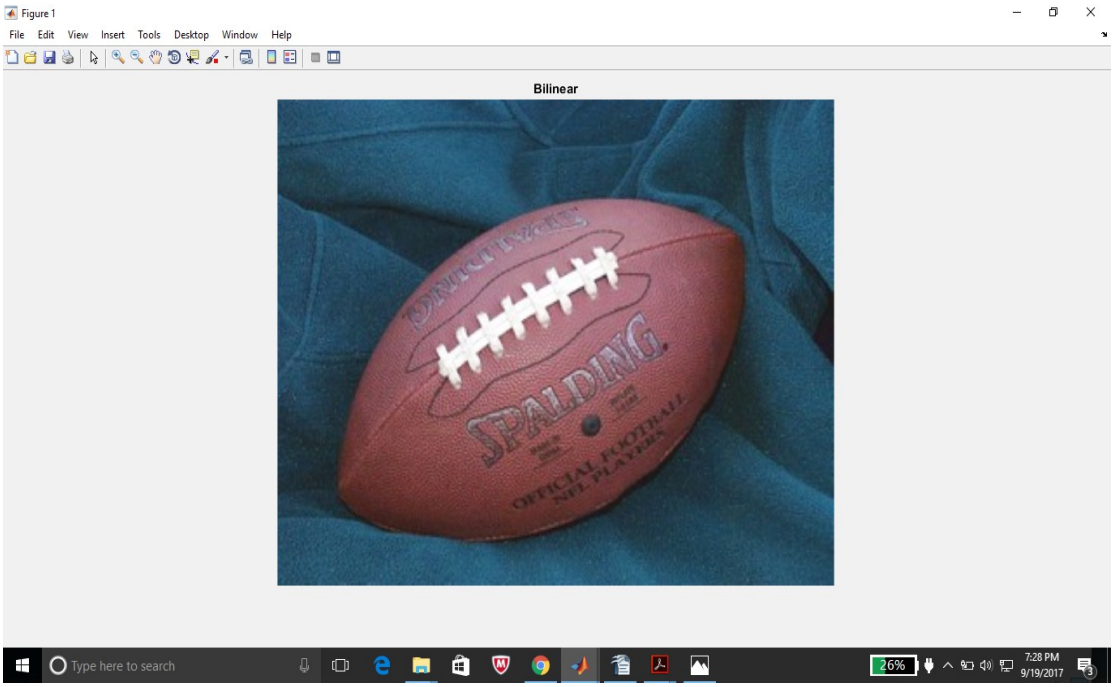
Output Image 2:



Output Image 3:



Output Image 4:



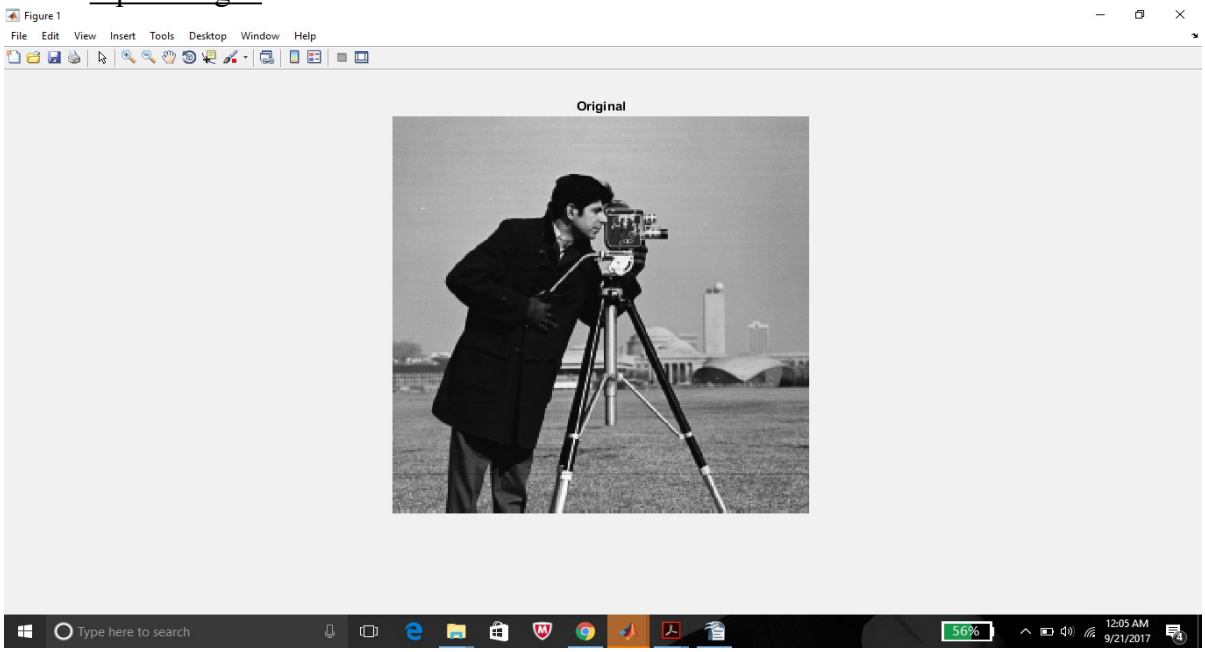
Observations: The output quality degrades slightly when *n* and factor by which upsampling occurs are increased. This gives better results than nearest neighbour interpolation. This is equivalent to doing linear interpolation in one dimension and then the other. This takes slightly more time than nearest neighbours.

Results: If we keep *n* and factor small, this method gives good results and seems to be a good tradeoff in quality of image and time taken. Note that nearest neighbour gives poorest quality in fastest time whereas bicubic gives best quality in a lot of time. Bilinear interpolation seems to be the best choice as the quality is satisfactory for the time taken.

Bicubic:

Parameters 1: n = 2, factor = 2

Input Image :

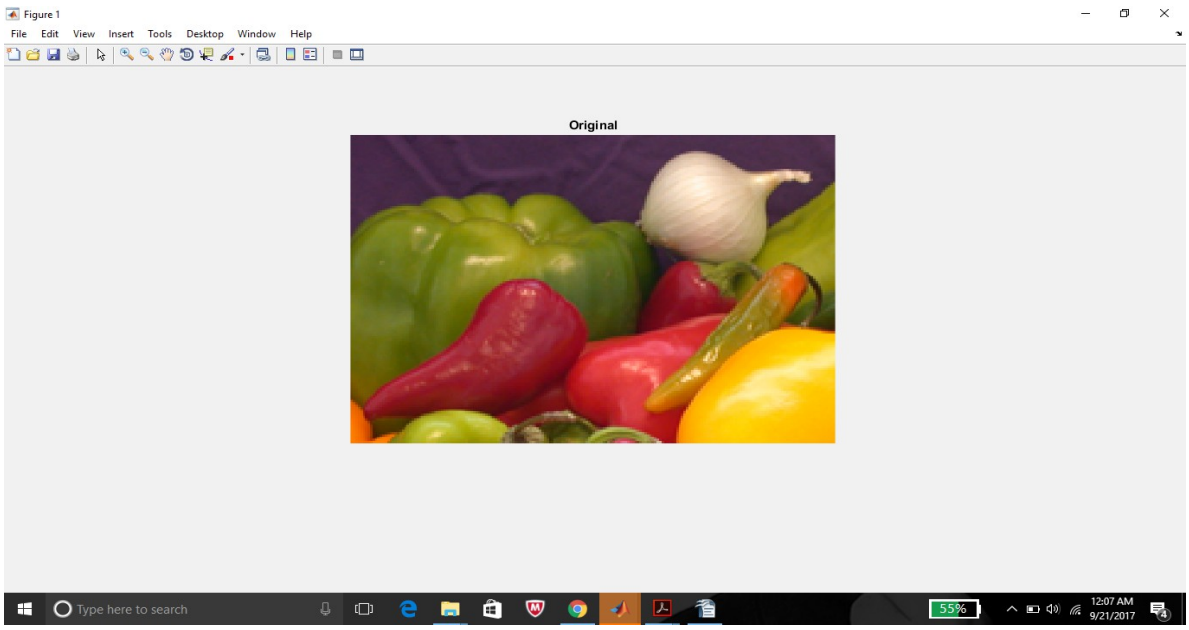


Output 1a:

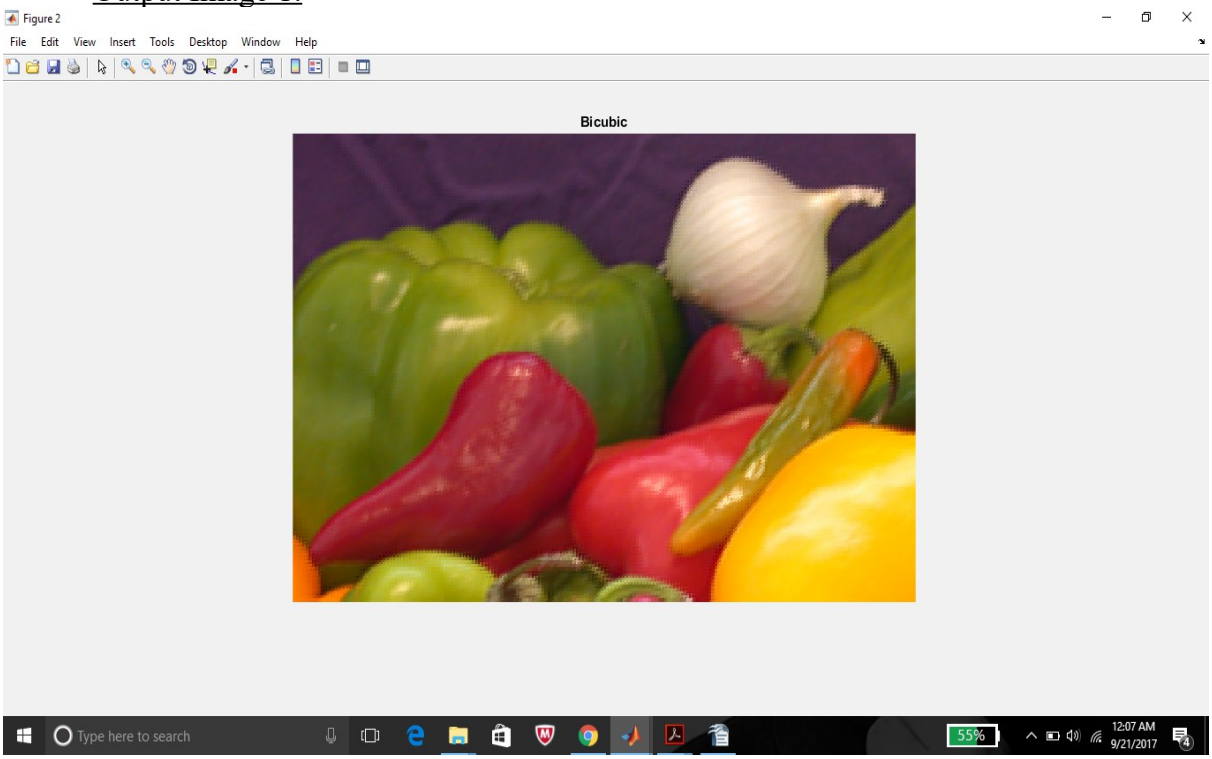
Output 1b:

Parameters 2: n = 3, factor = 2

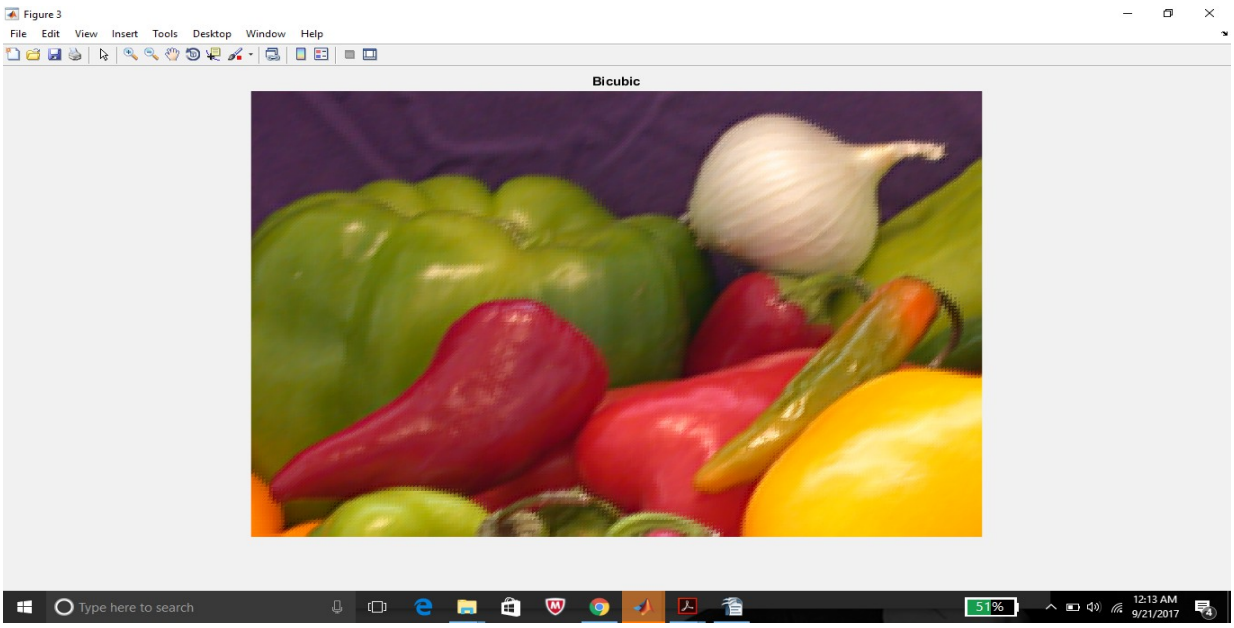
Input Image:



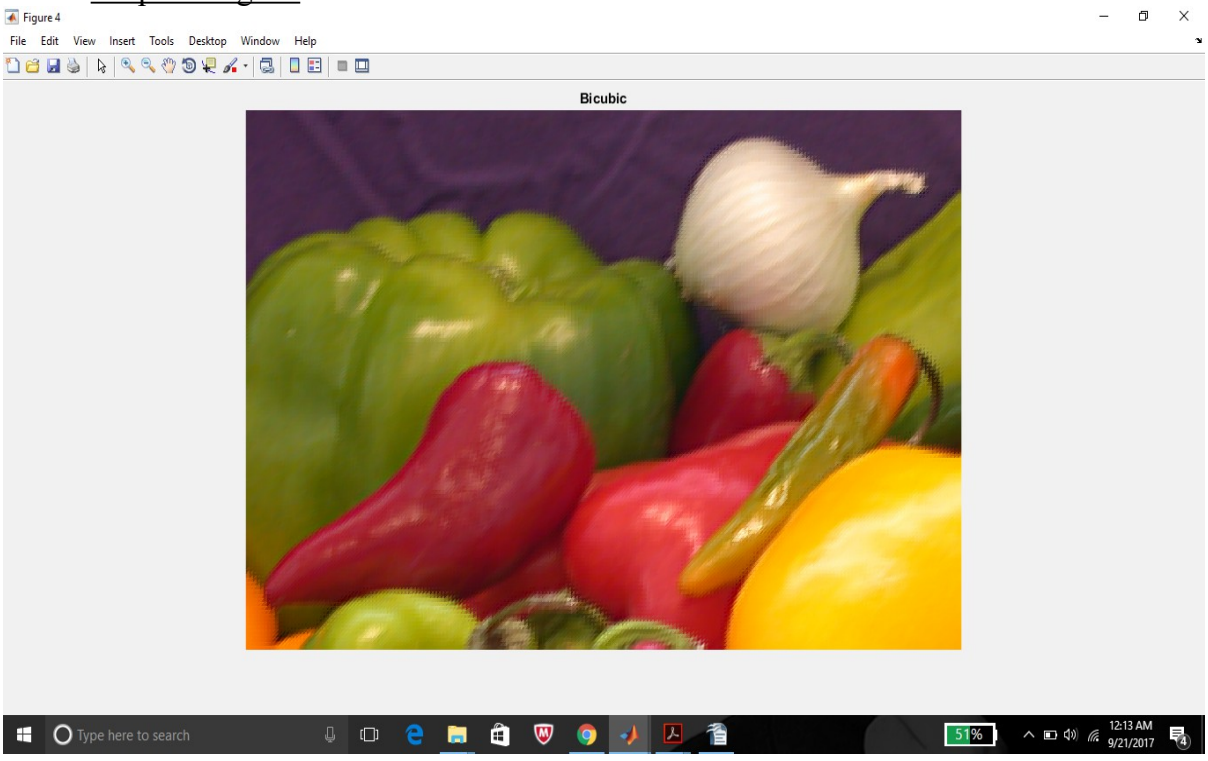
Output Image 1:



Output Image 2:

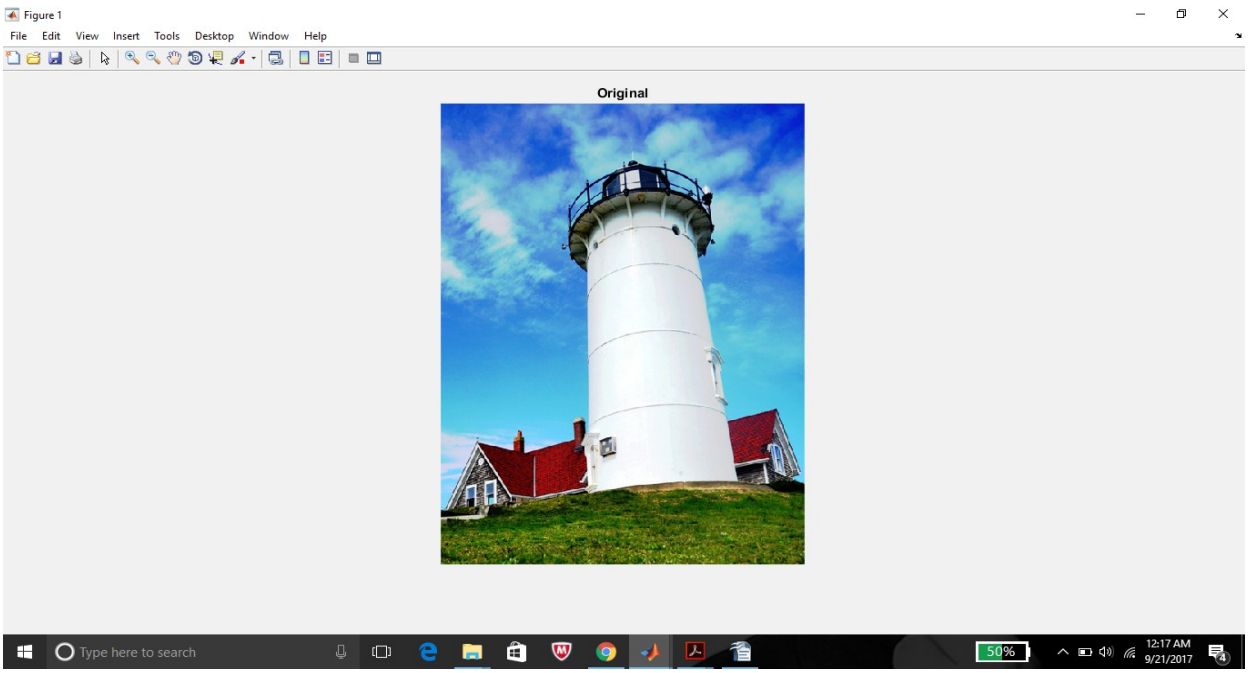


Output Image 3:

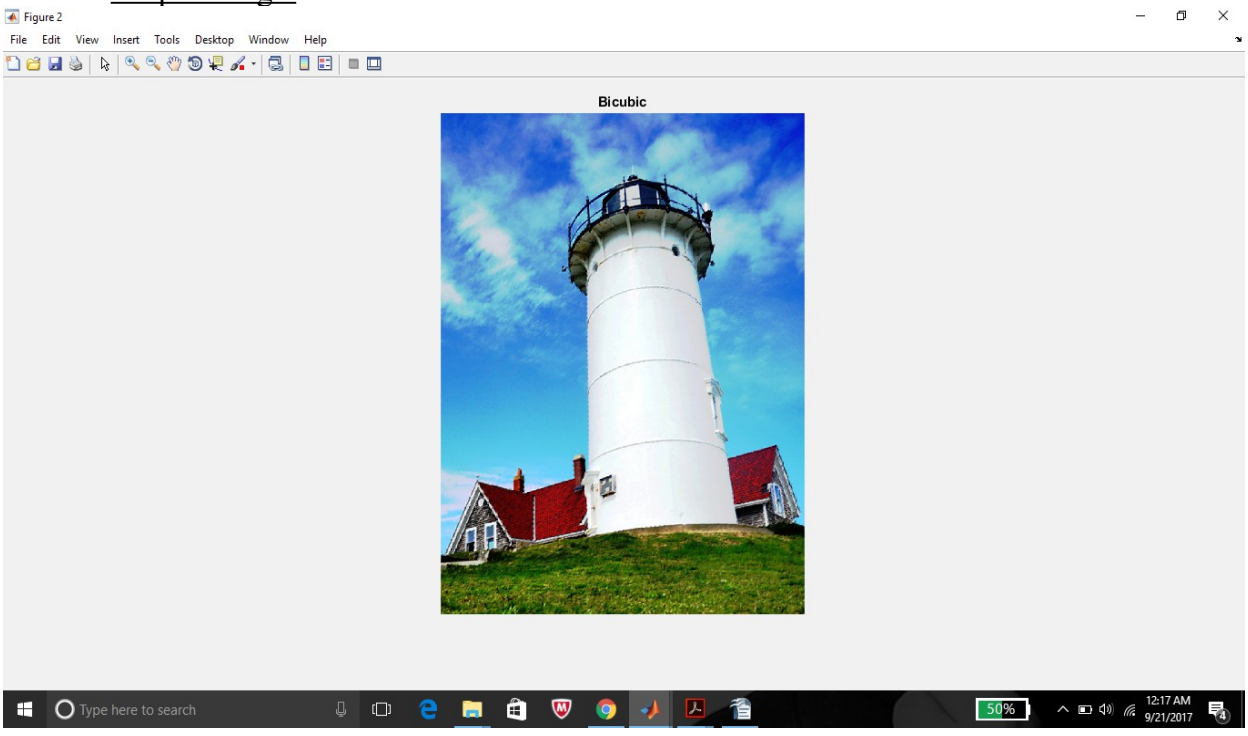


Parameters 3: n = 1, factor = 2

Input Image:



Output Image:



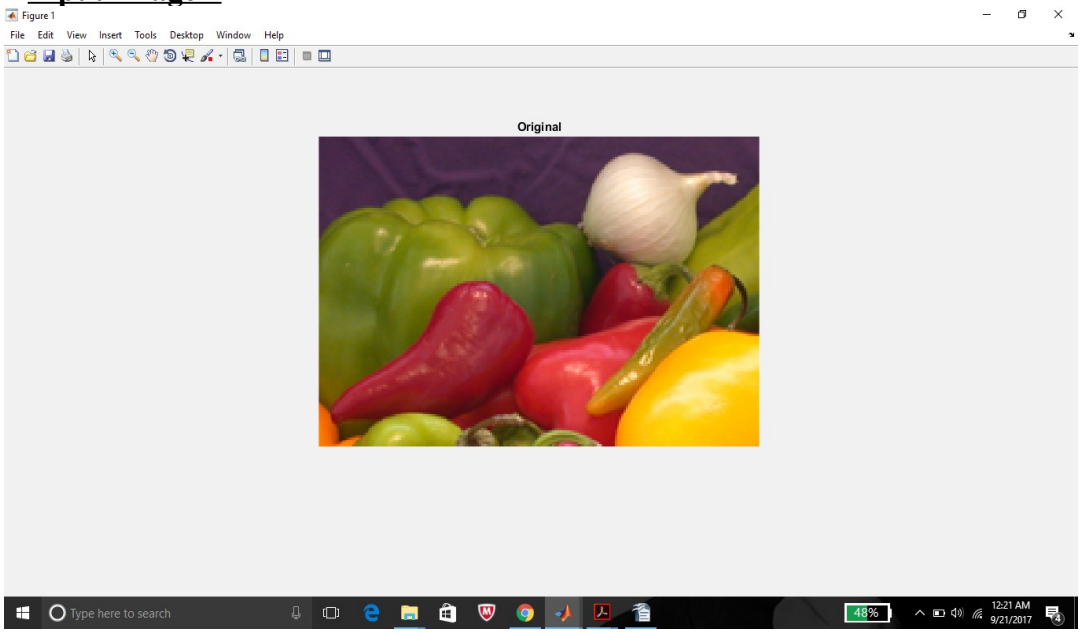
Observations: The quality of the output is very good although a lot of time is taken (which is why I have kept the number of levels low). It is slightly faster to work with grayscale images as compared to RGB since the operations have to be performed only once (as compared to thrice in RGB).

Results: The output is good but since a lot of time is taken, I feel that bilinear interpolation is a better option. This may be a good choice, however, for smaller images since it takes lesser time to get the output. Also, *n* should be kept small.

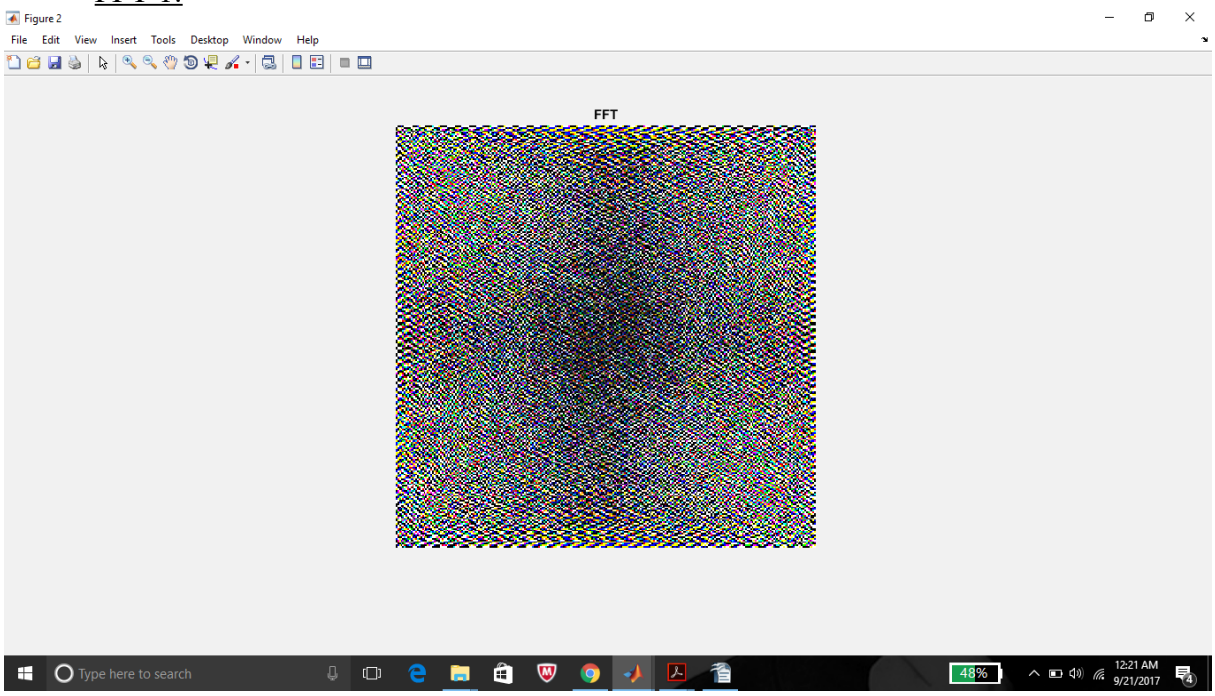
Problem 2:

a) Recursive Fast Fourier Transform:

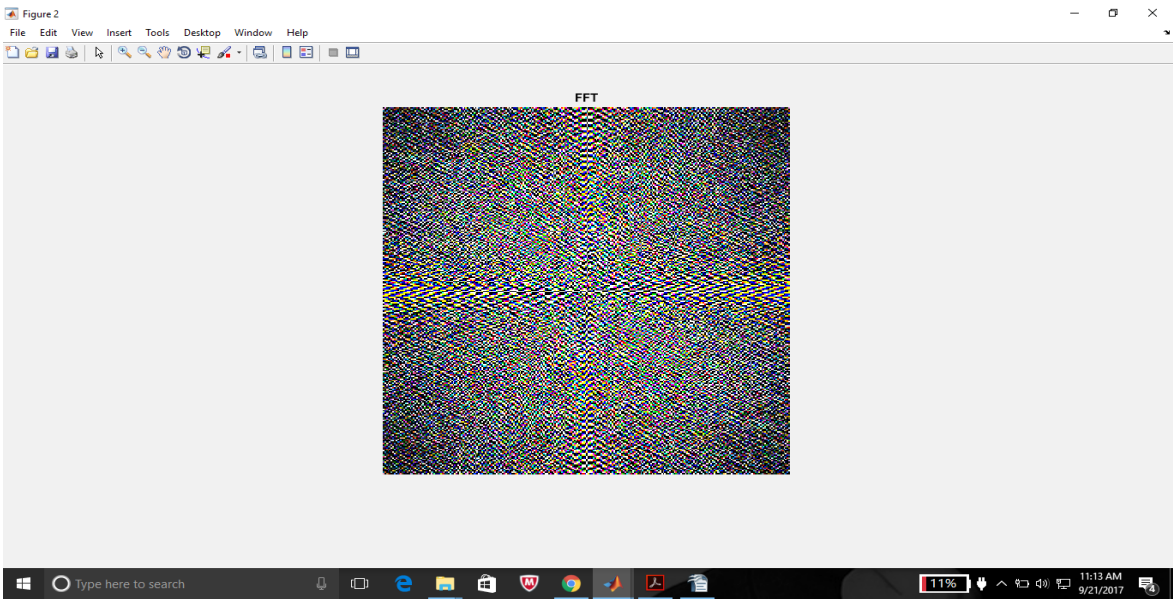
Input Image 1



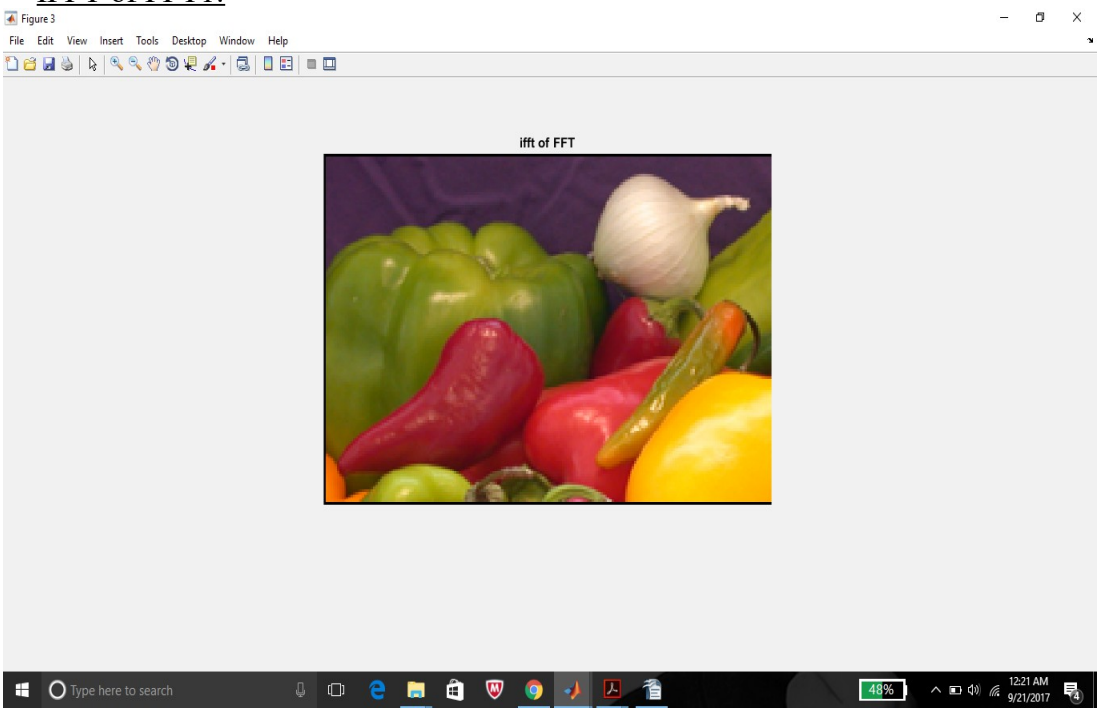
FFT 1:



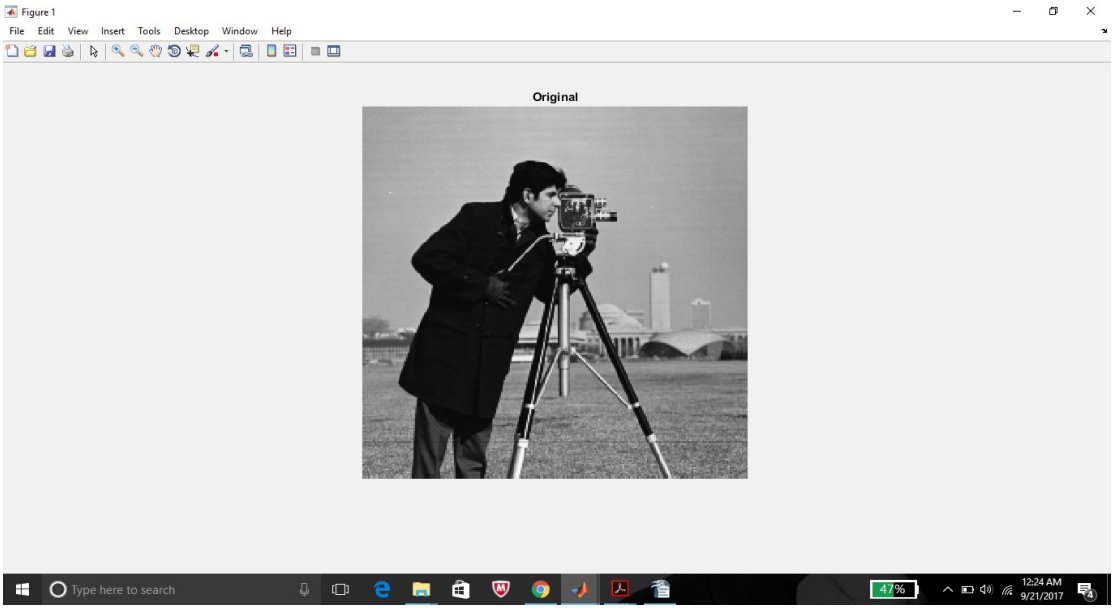
Centered FFT1: I have kept a variable center which gives centered FFT when set to 1. Default value is 0.



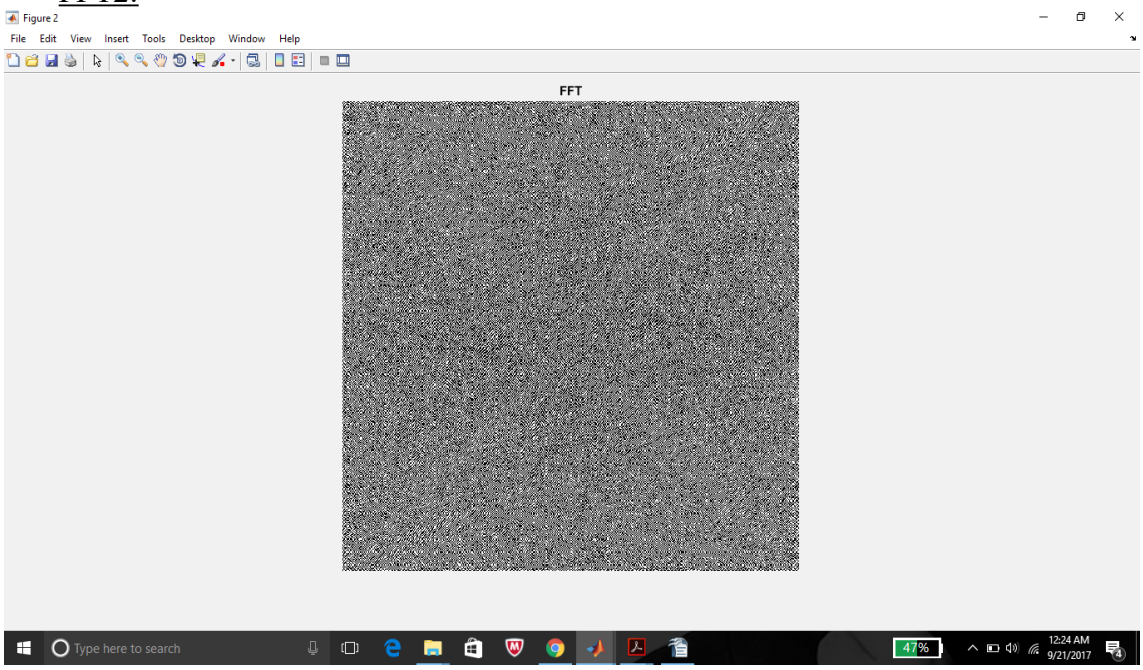
IFFT of FFT1:



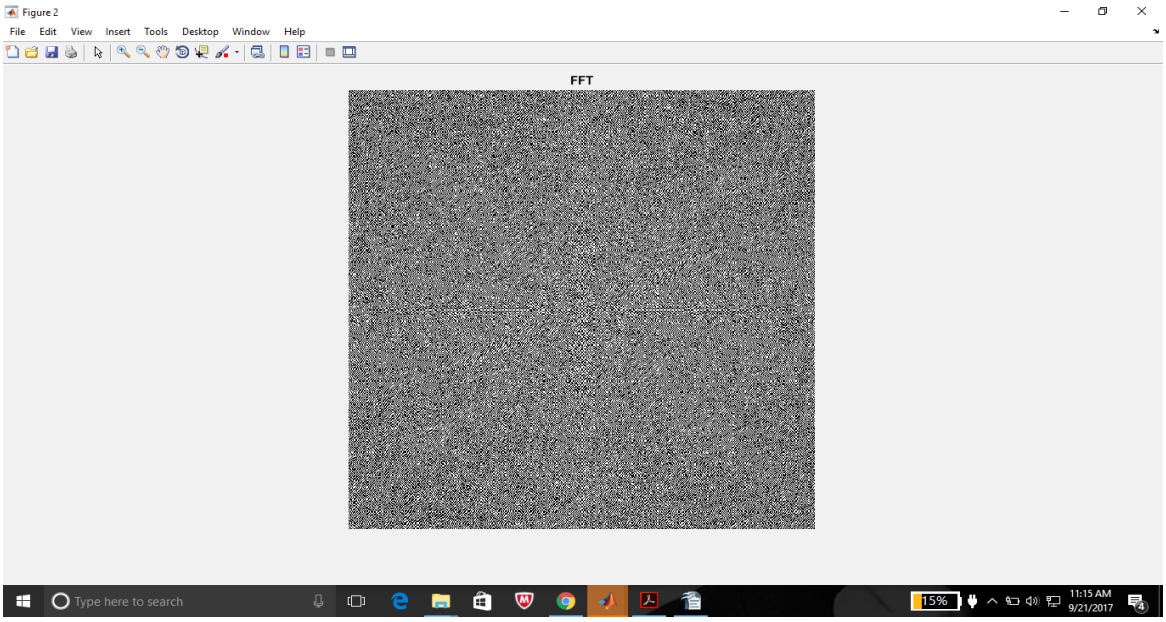
Input Image 2:



FFT2:

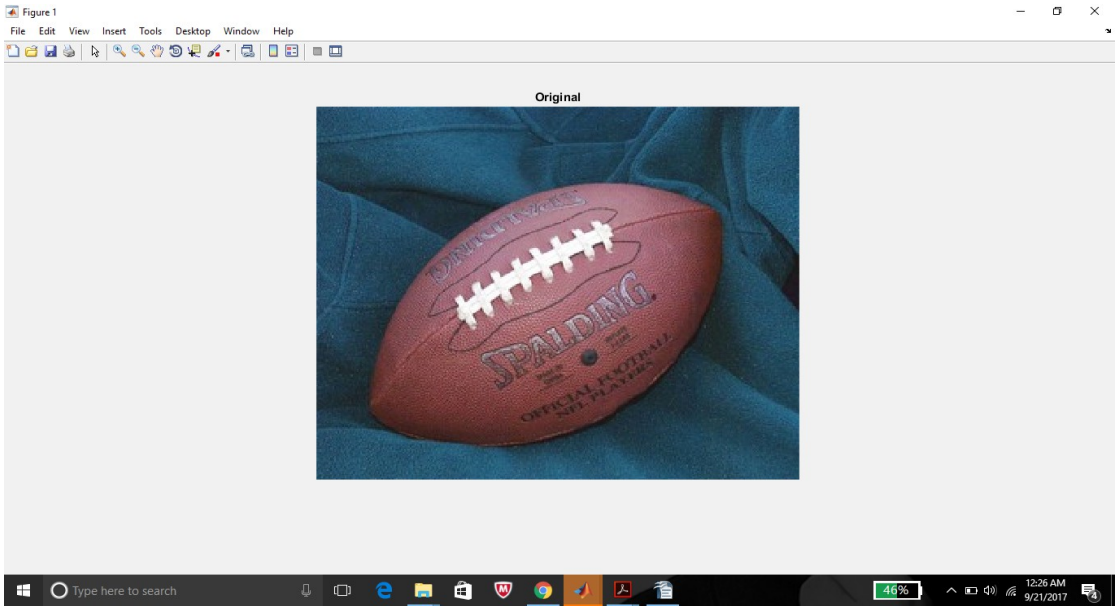


Centered FFT2:

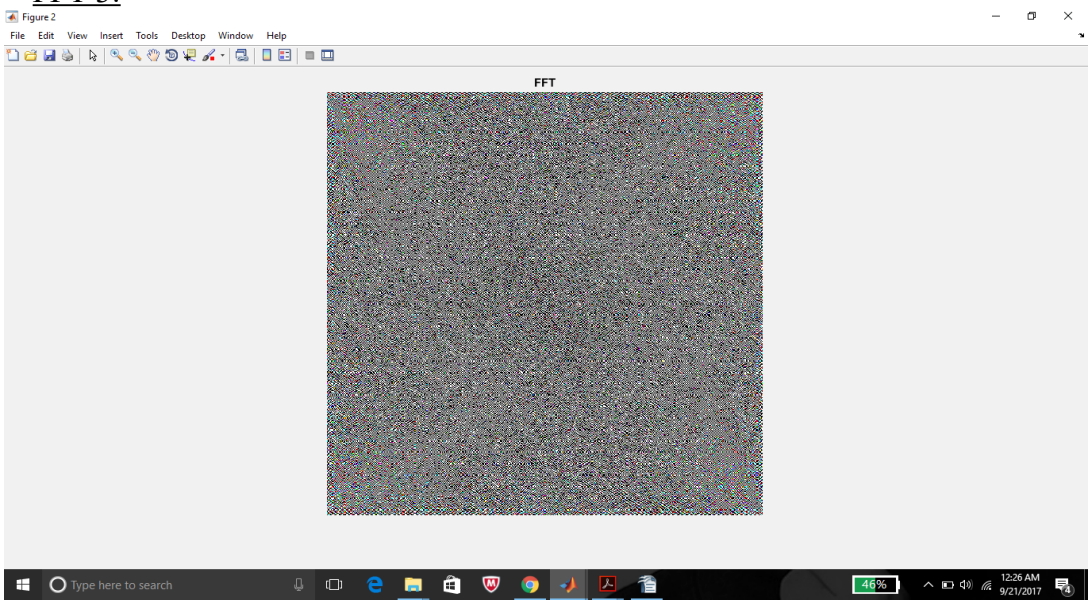


IFFT of FFT2:

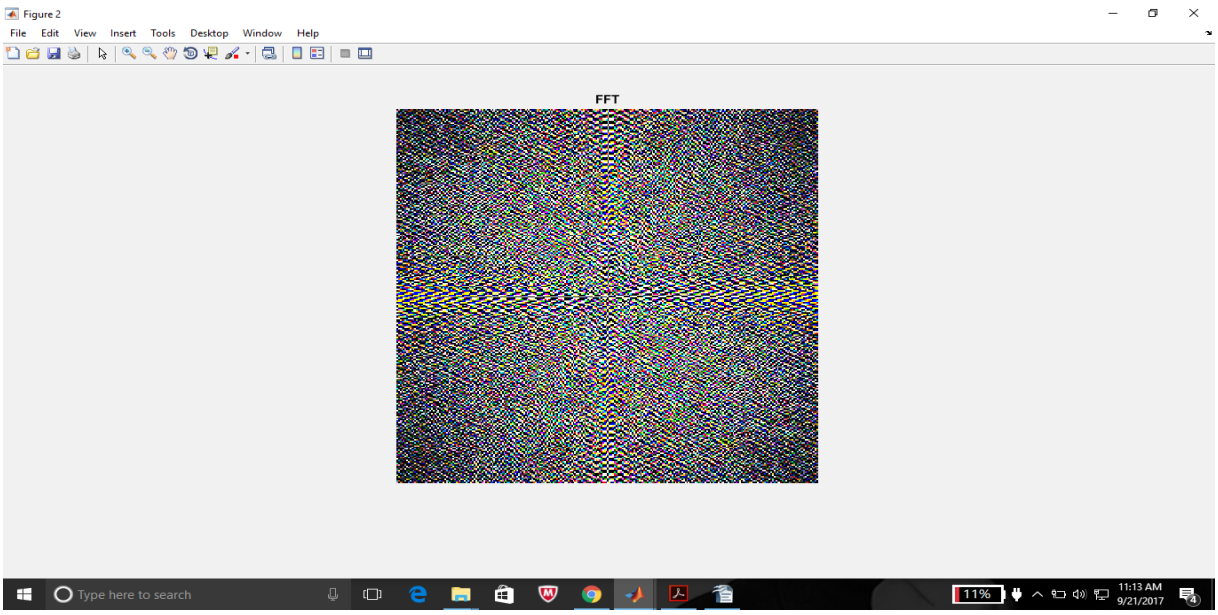
Input Image 3:



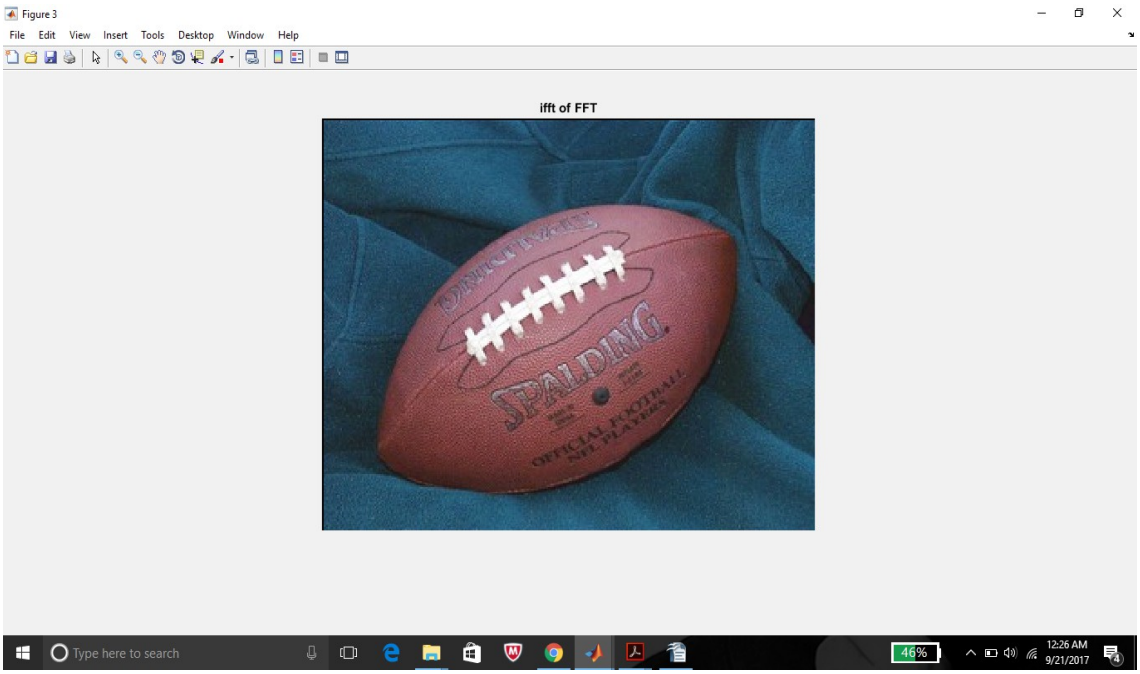
FFT 3:



Centered FFT3:



IFFT of FFT3:



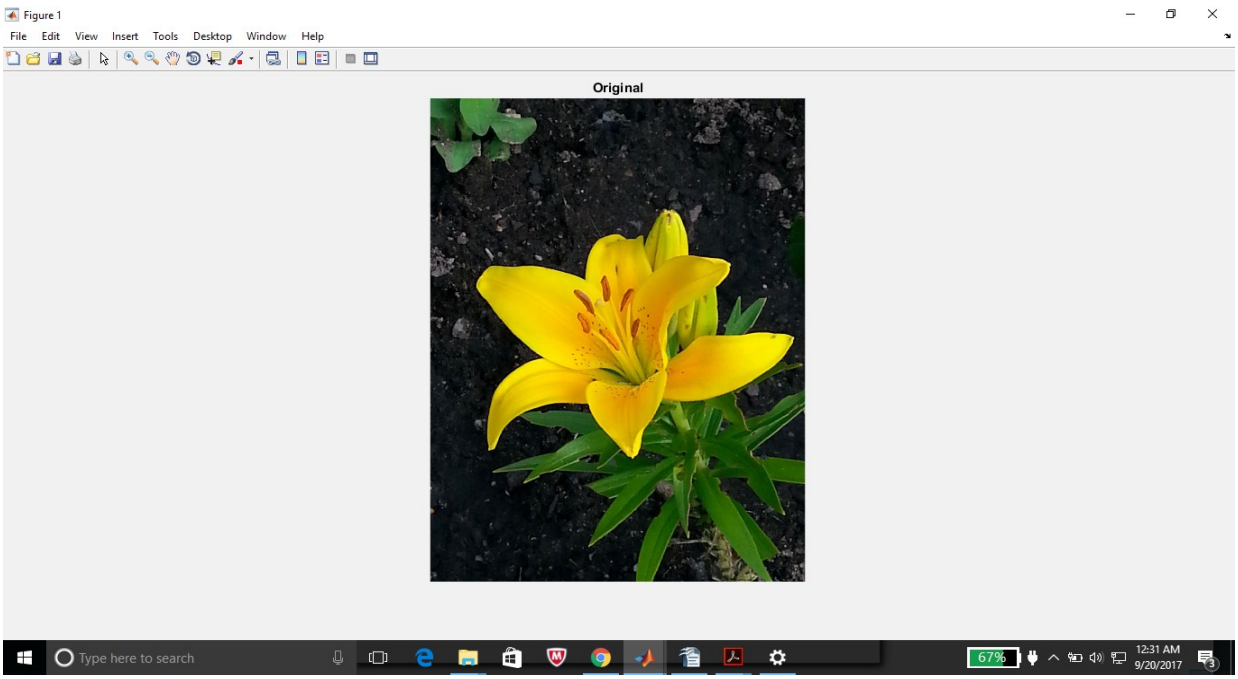
Observations: The time taken to compute the FFT is of order nlogn which is why a little delay is present. We can see that the ifft of the Fourier transform is nearly identical to the input image. Note that I computed FFT for all channels separately in case of color images and that the FFT is radix-2 recursive.

Results: The Fourier transform is obtained in little time.

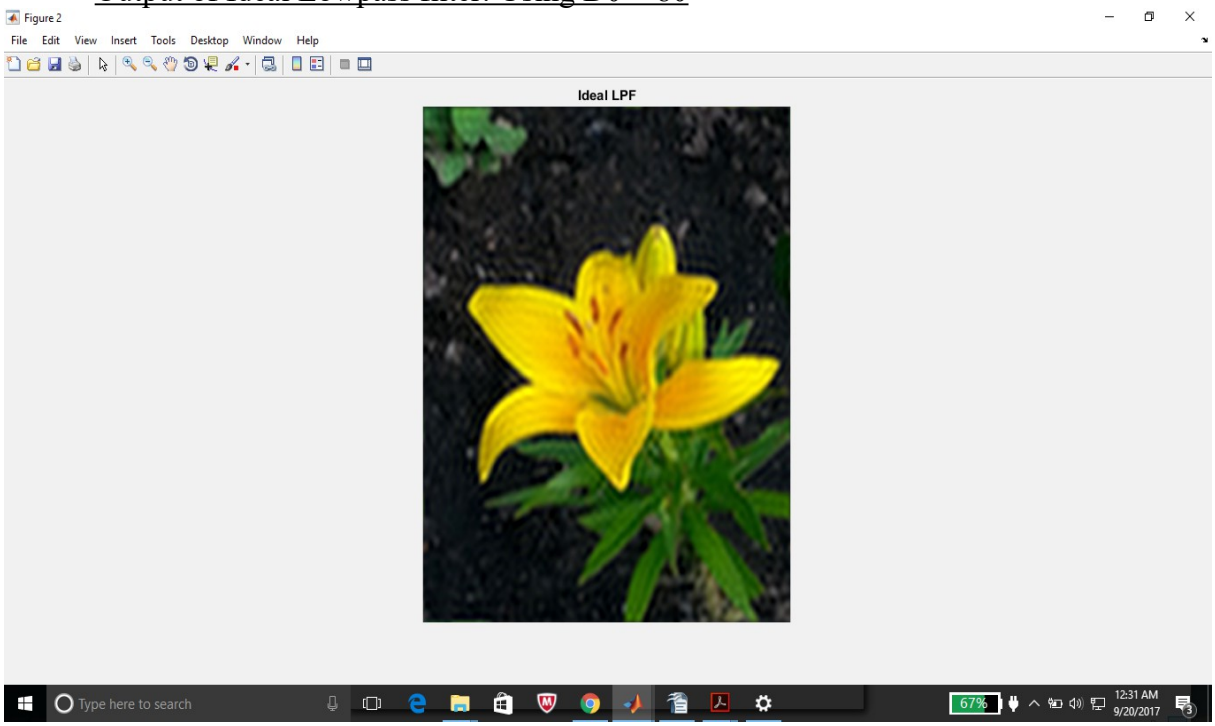
b) Filters

Parameters 1:

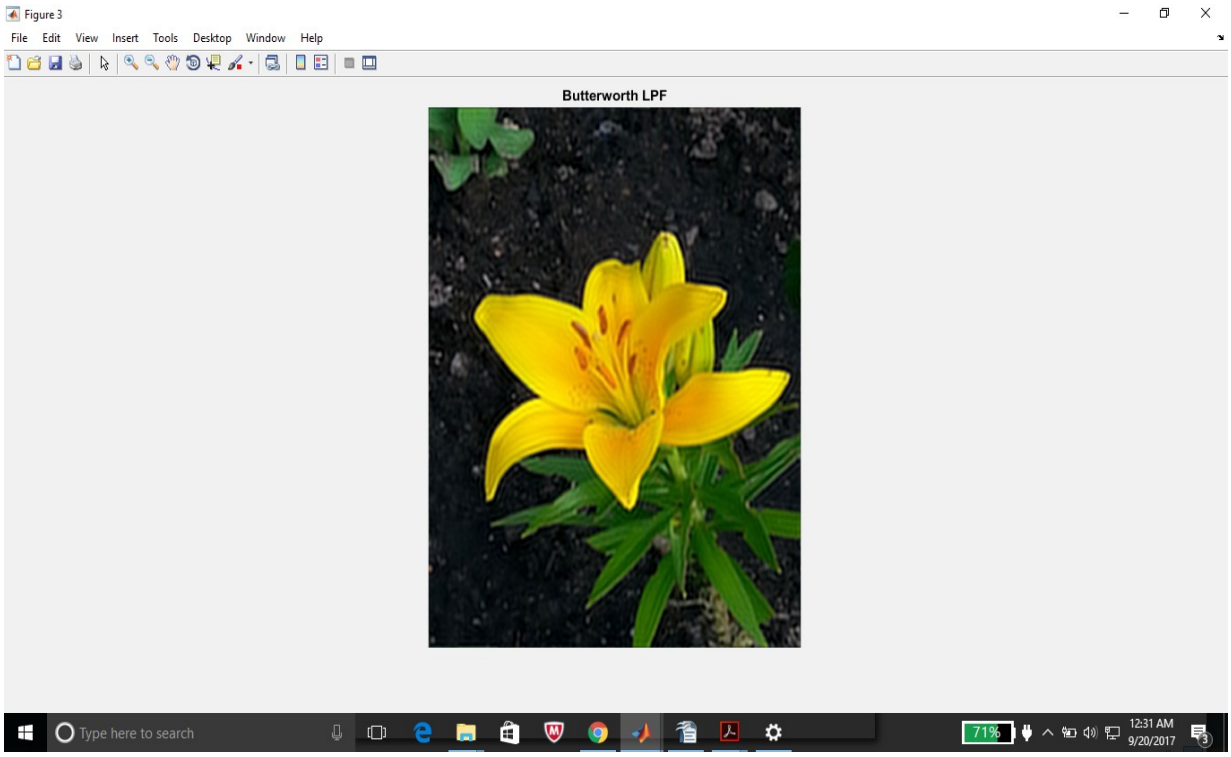
Input Image :



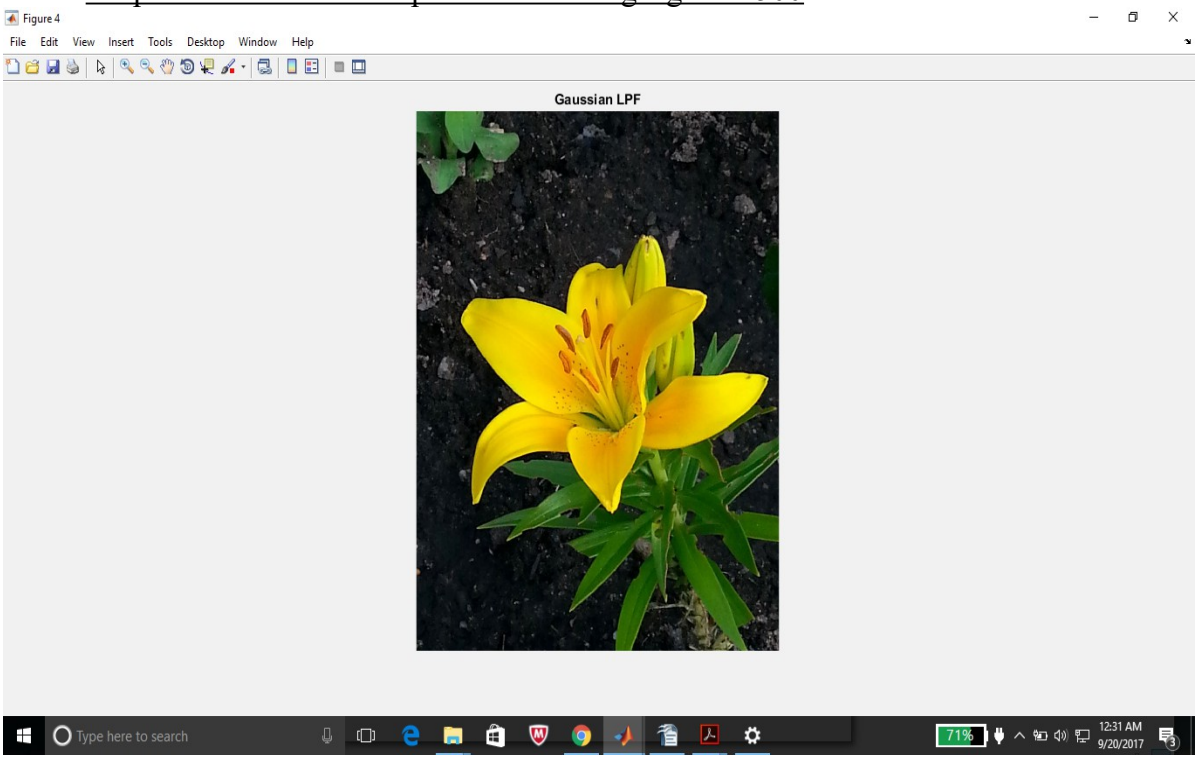
Output of Ideal Lowpass filter: Using D0 = 80



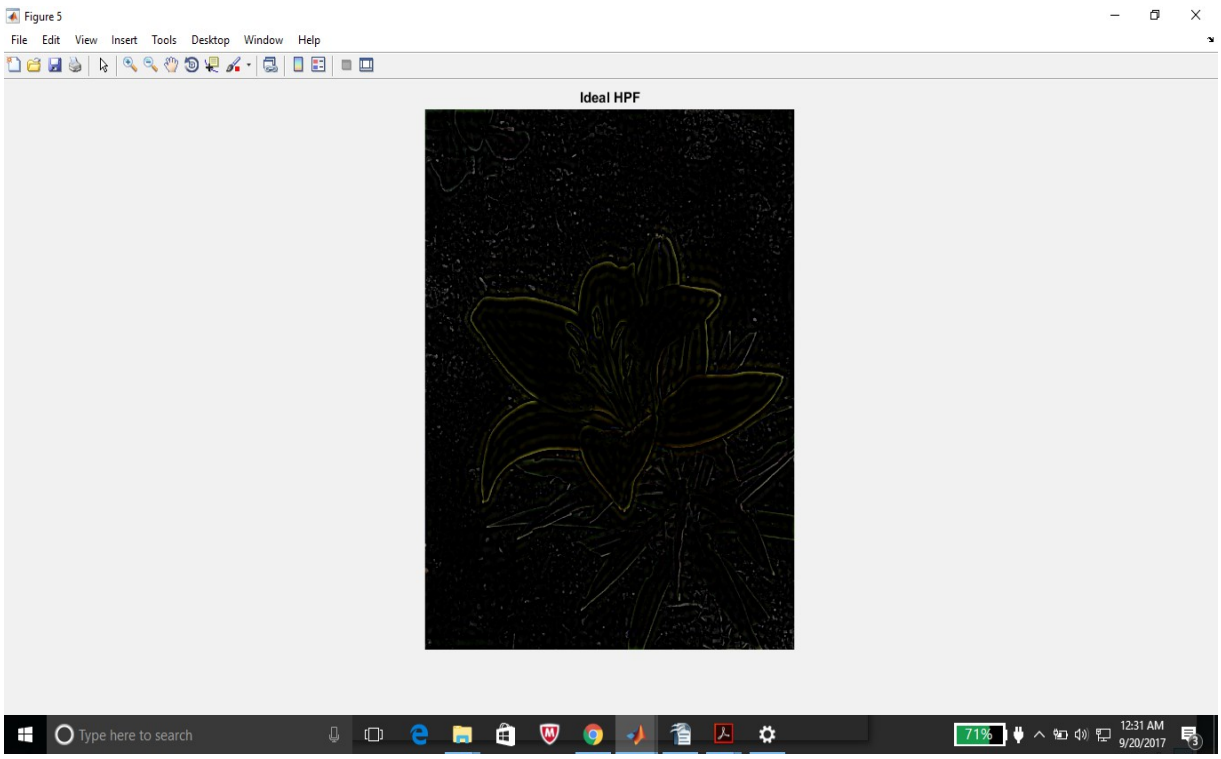
Output for Butterworth Lowpass Filter: Using order = 20 and D0 = 80



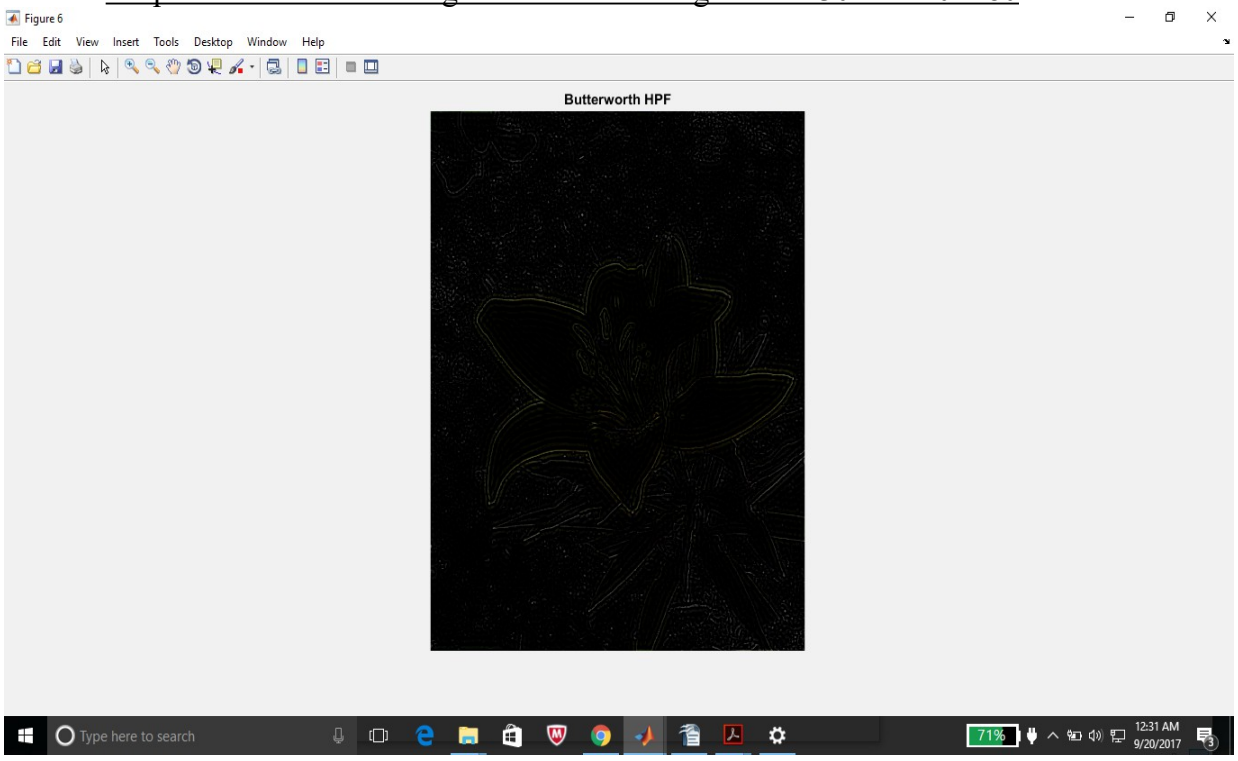
Output for Gaussian Lowpass Filter: Using sigma = 500



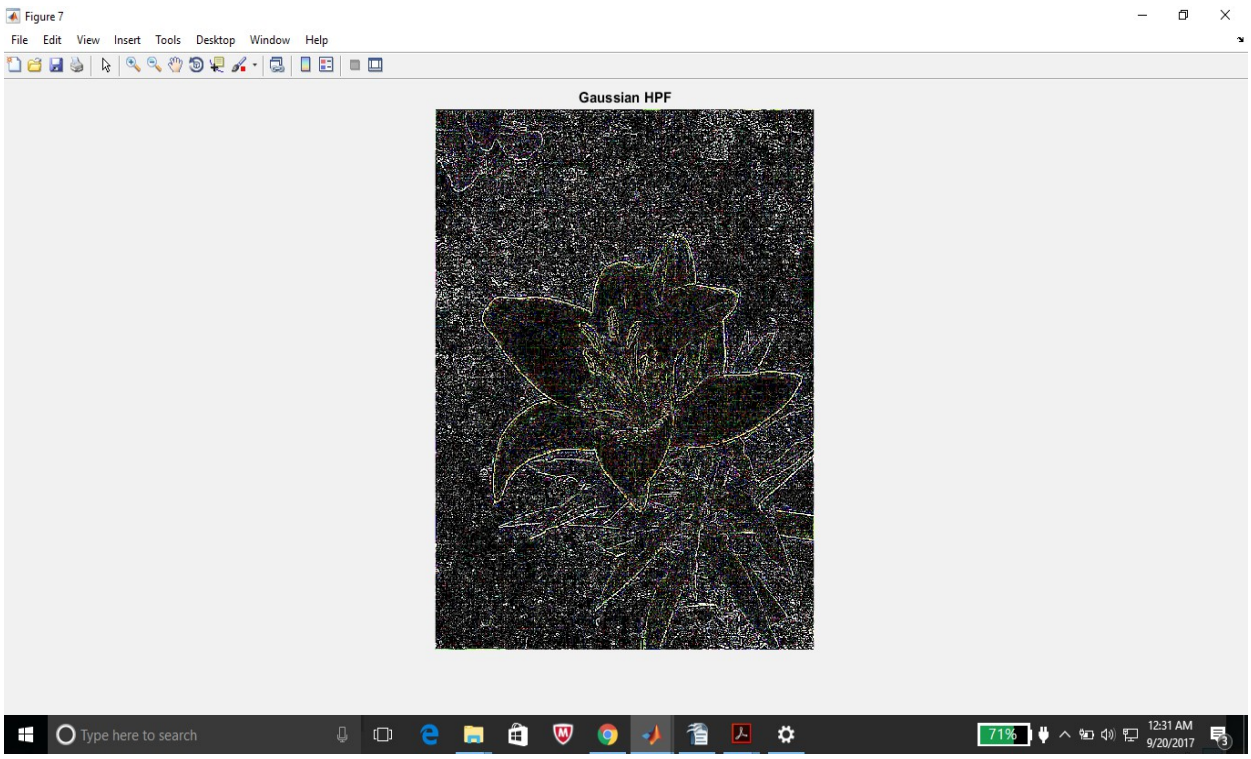
Output for Ideal High Pass Filter: Using D0 = 80



Output for Butterworth High Pass Filter: Using order = 30 and D0 = 80

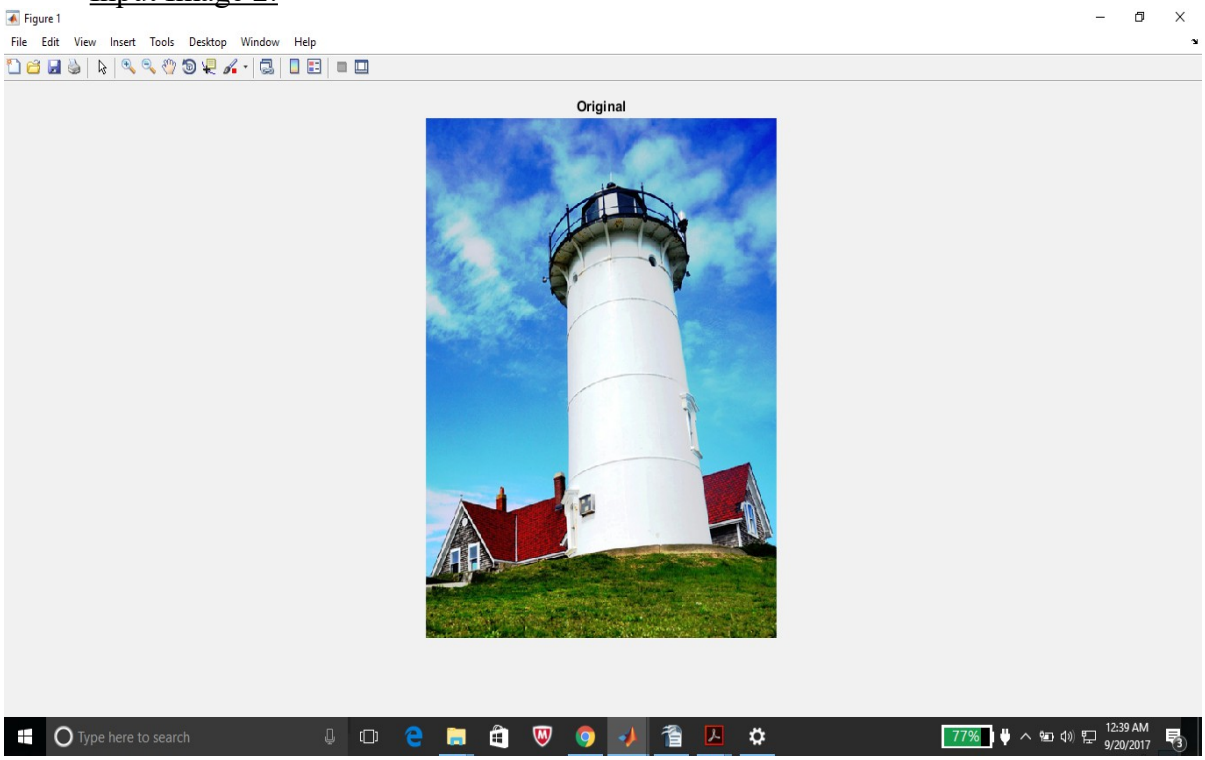


Output for Gaussian High Pass Filter: Sigma = 500

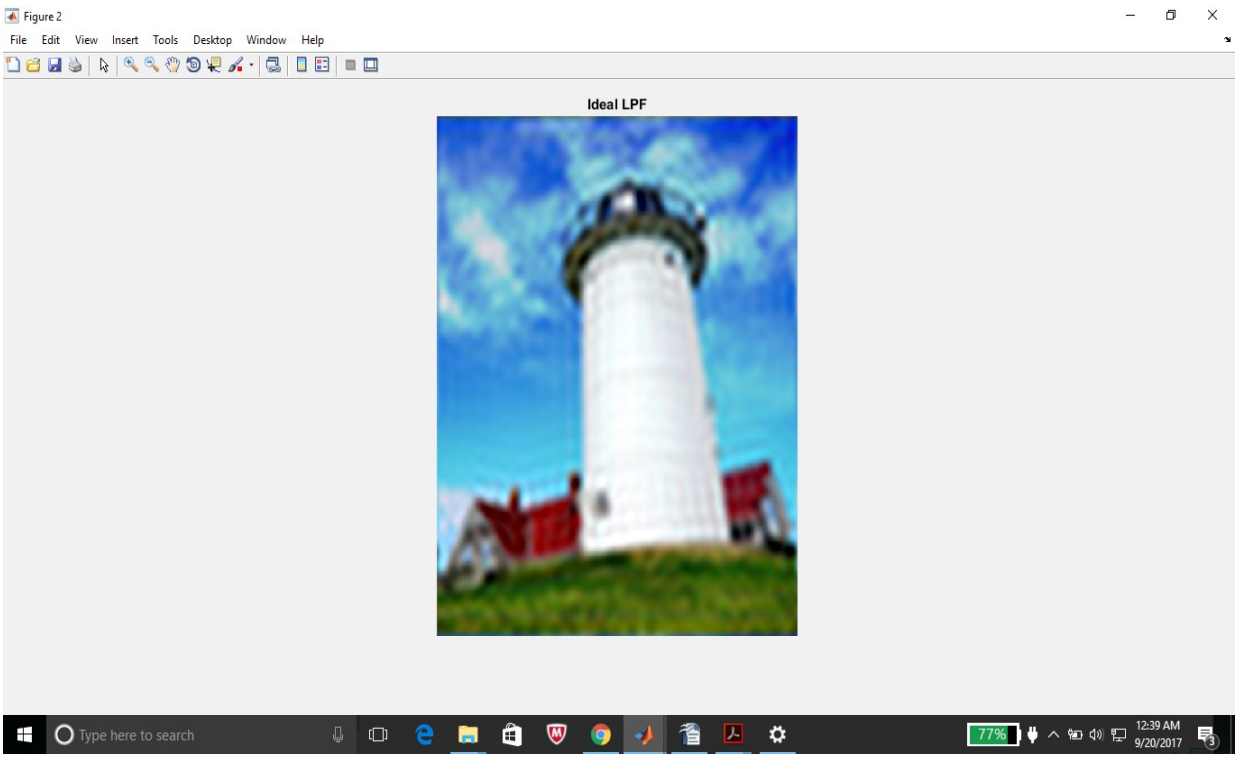


Parameters 2:

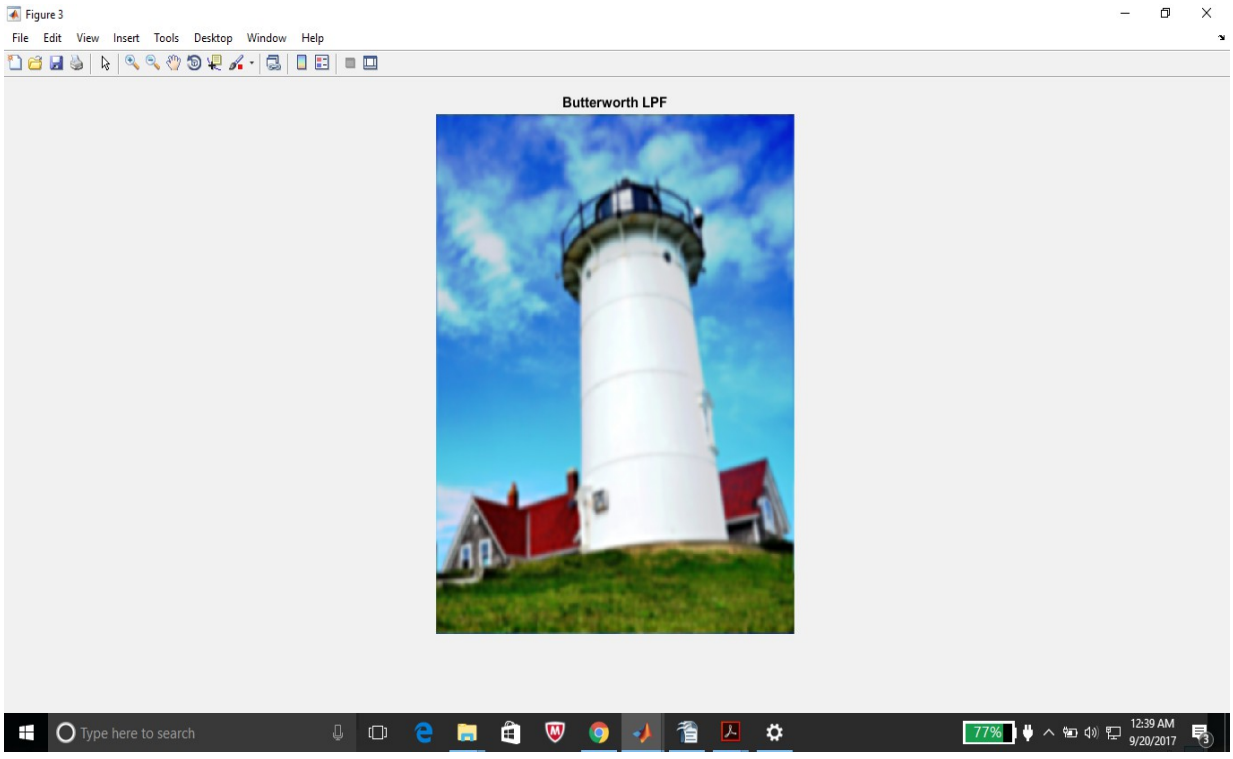
Input Image 2:



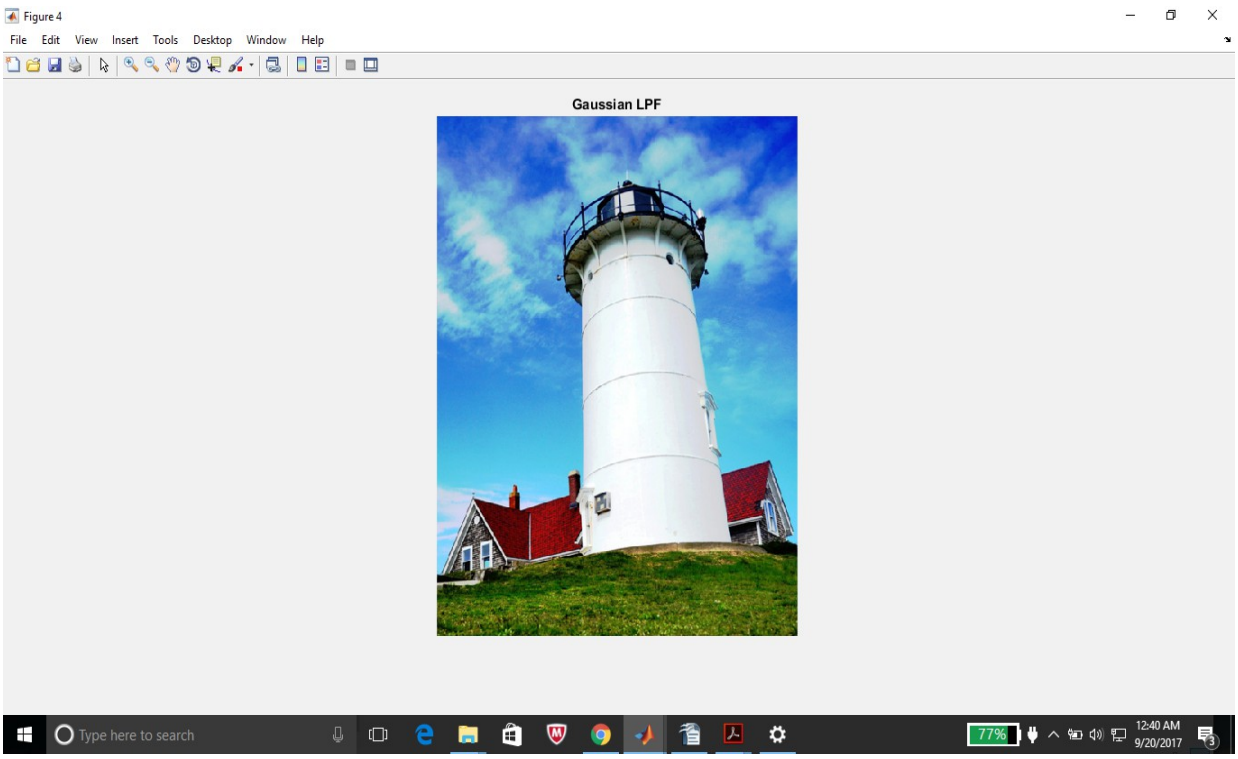
Output for Ideal Low Pass Filter: Using D0 = 50. We can clearly see the ringing effect present in this picture.



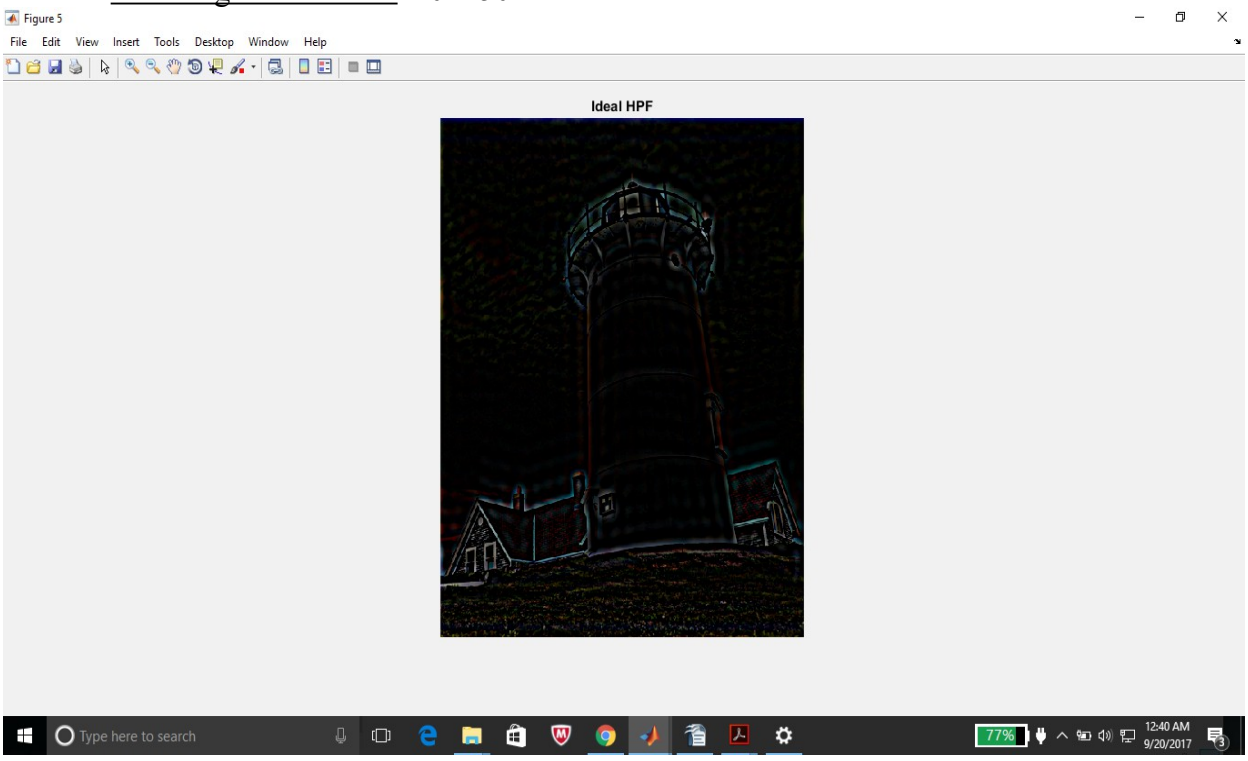
Output for Butterworth Low Pass Filter: Order = 2 and D0 = 50



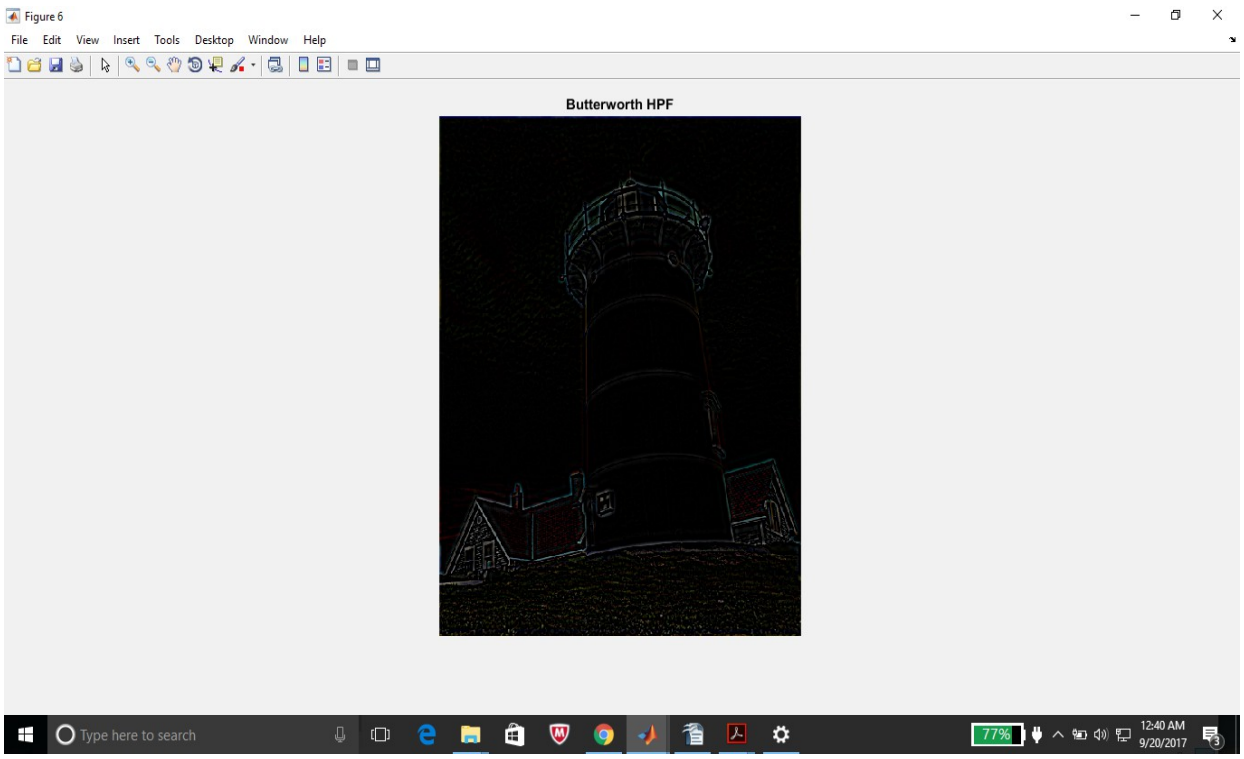
Output for Gaussian Low Pass Filter: Sigma = 350



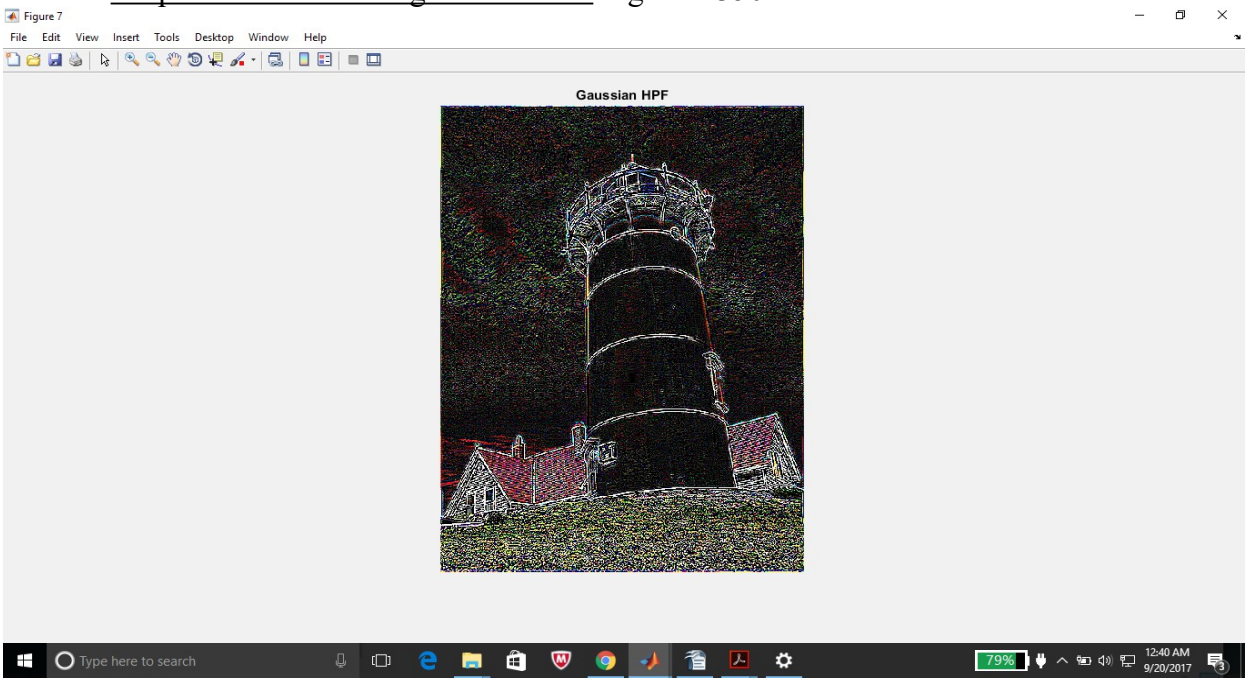
Ideal High Pass Filter: D0 = 50



Butterworth High Pass Filter: Order = 4 and D0 = 50

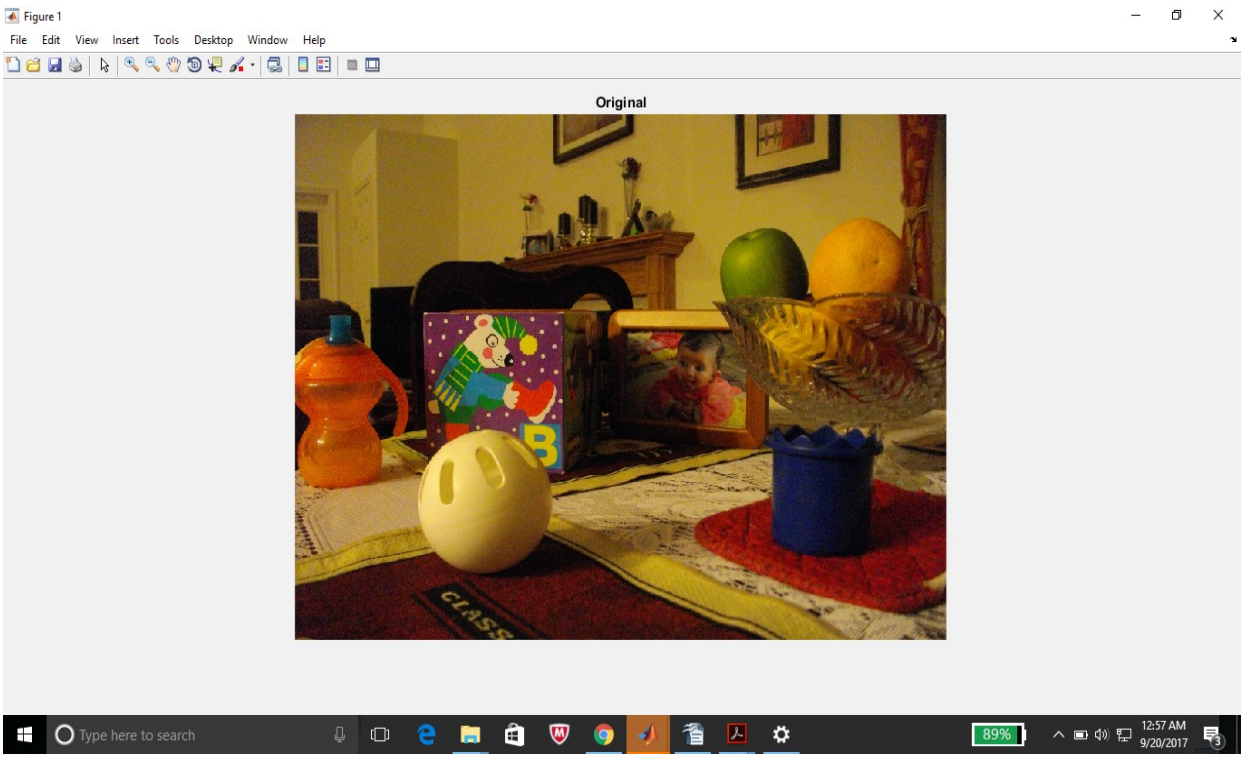


Output for Gaussian High Pass Filter: Sigma = 350

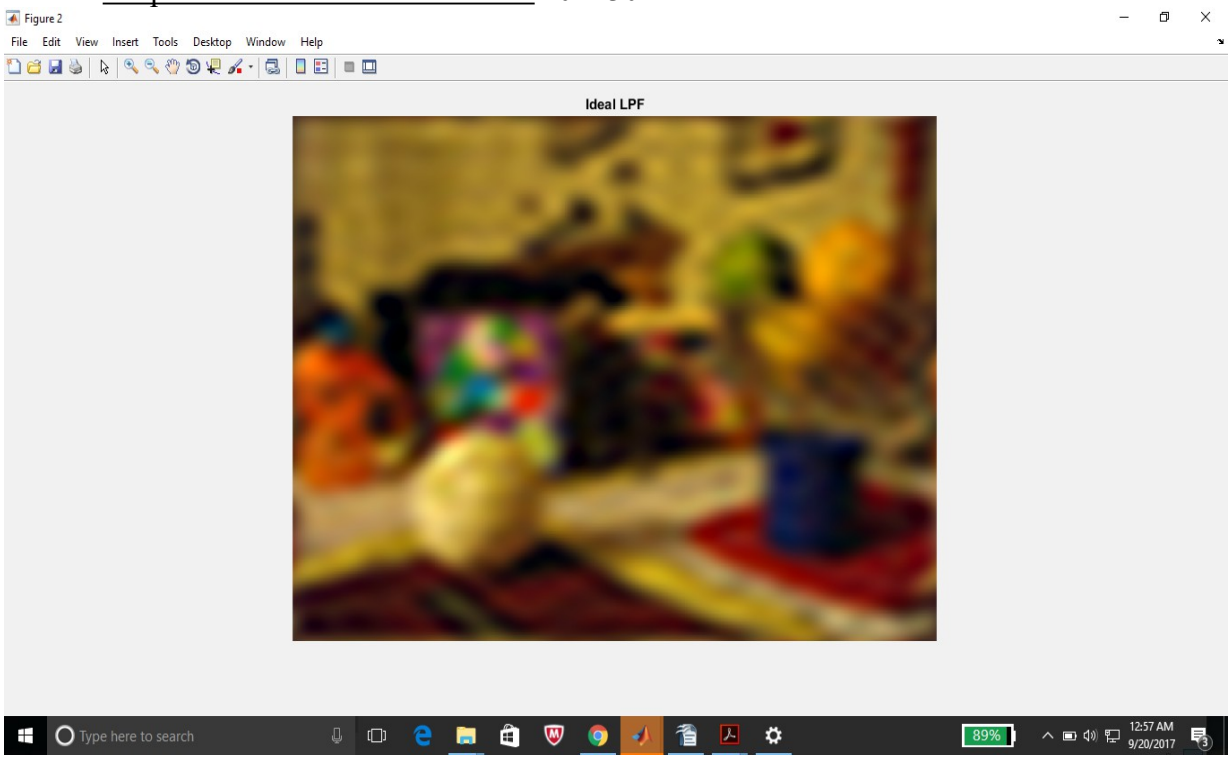


Parameters 3:

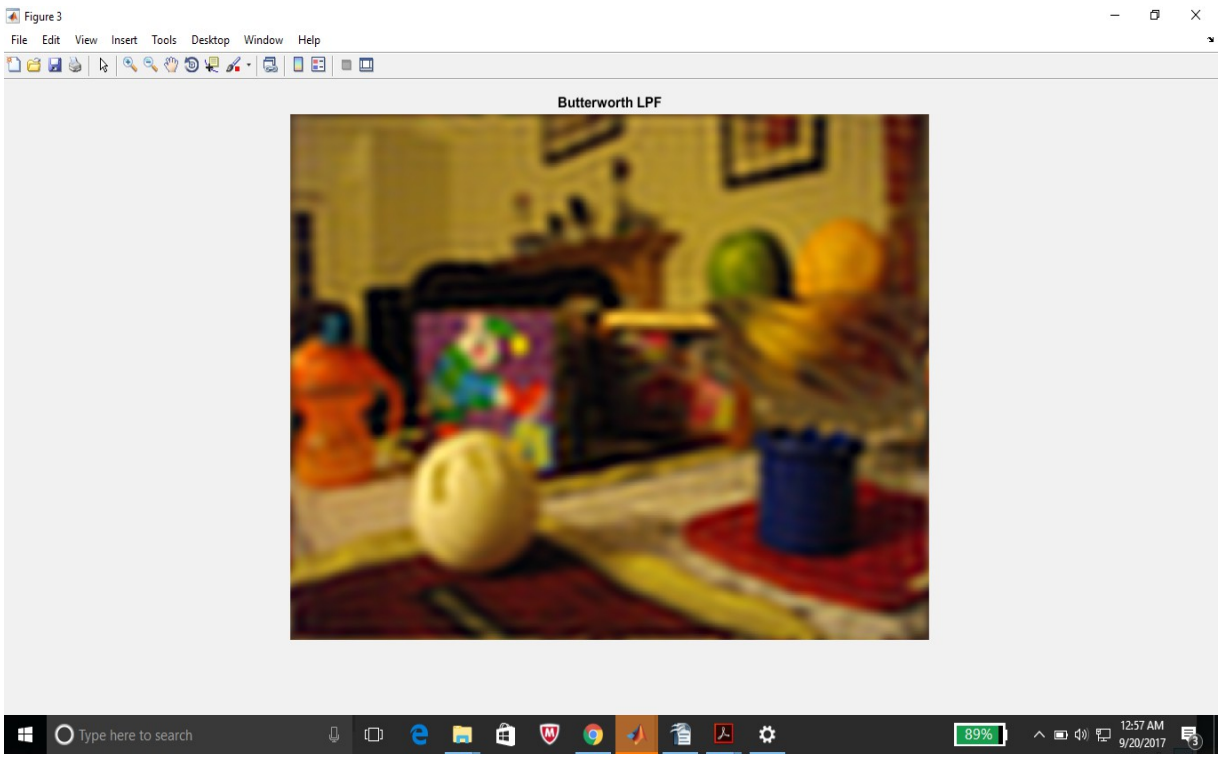
Input Image:



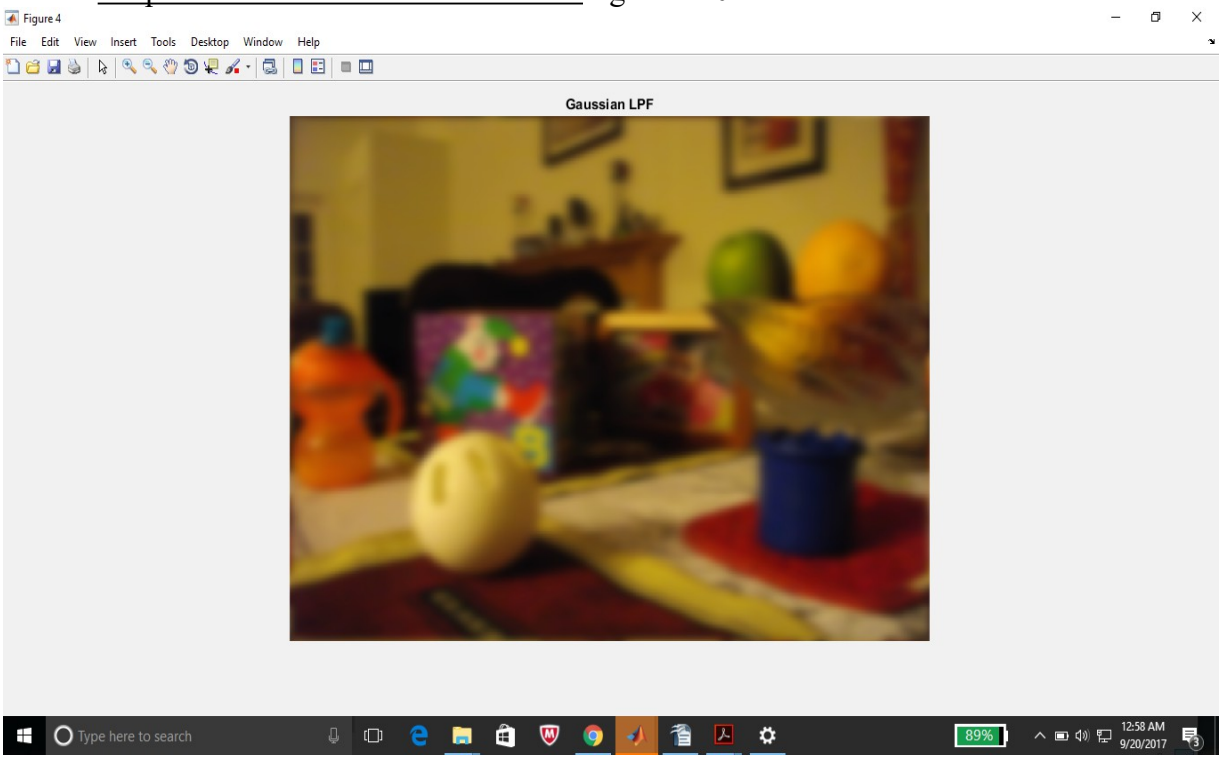
Output for Ideal Low Pass Filter: D0 = 30



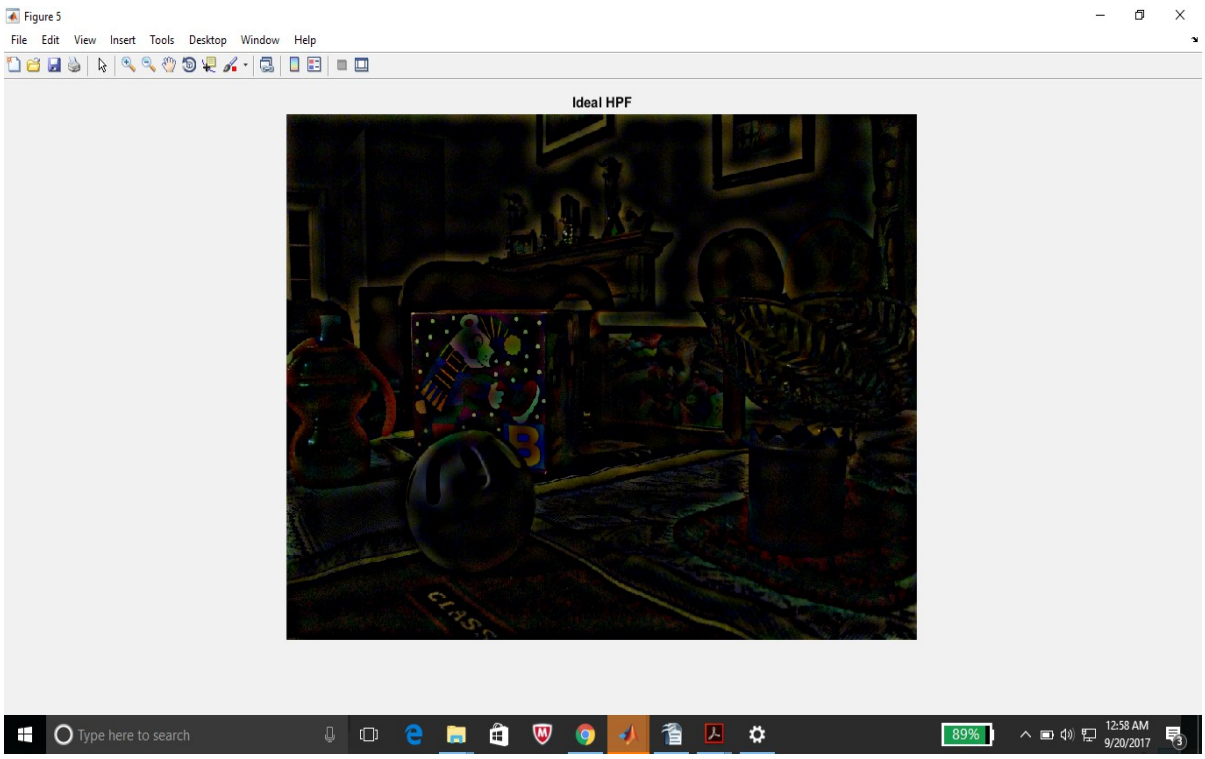
Output for Butterworth Low Pass Filter: D0 = 30, order = 15



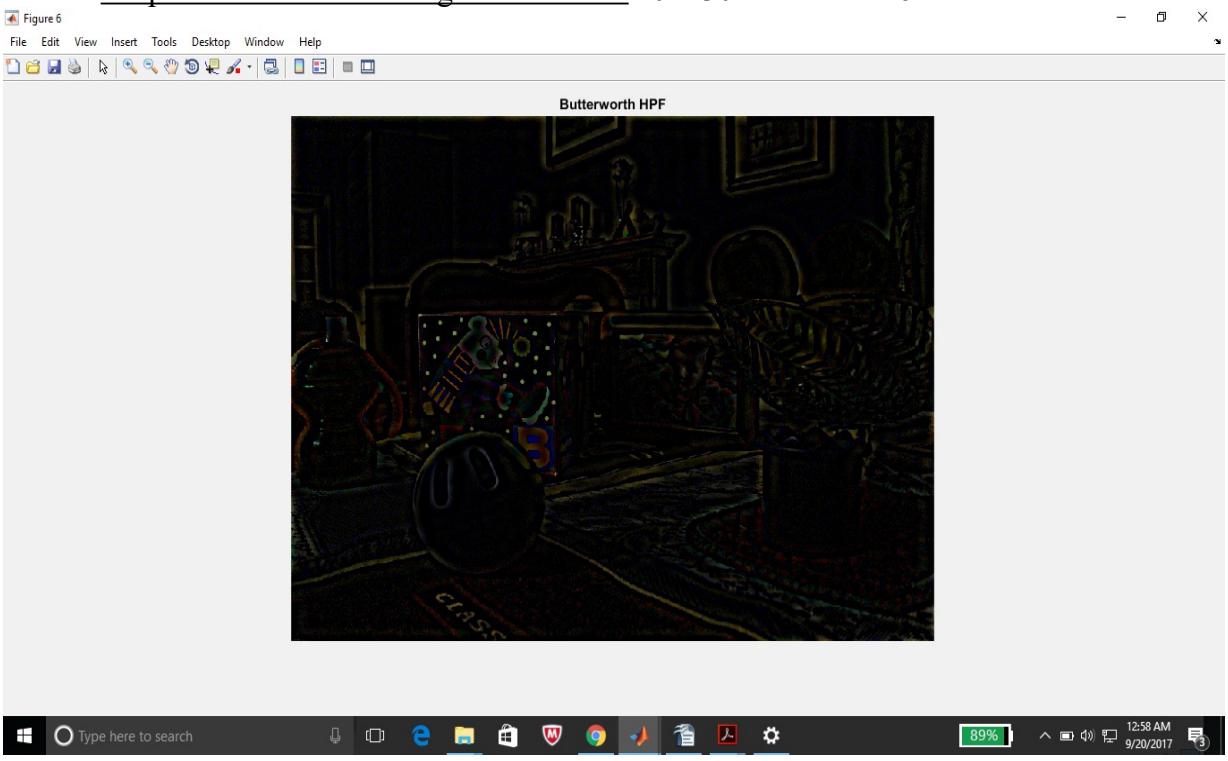
Output for Gaussian Low Pass Filter: Sigma = 20



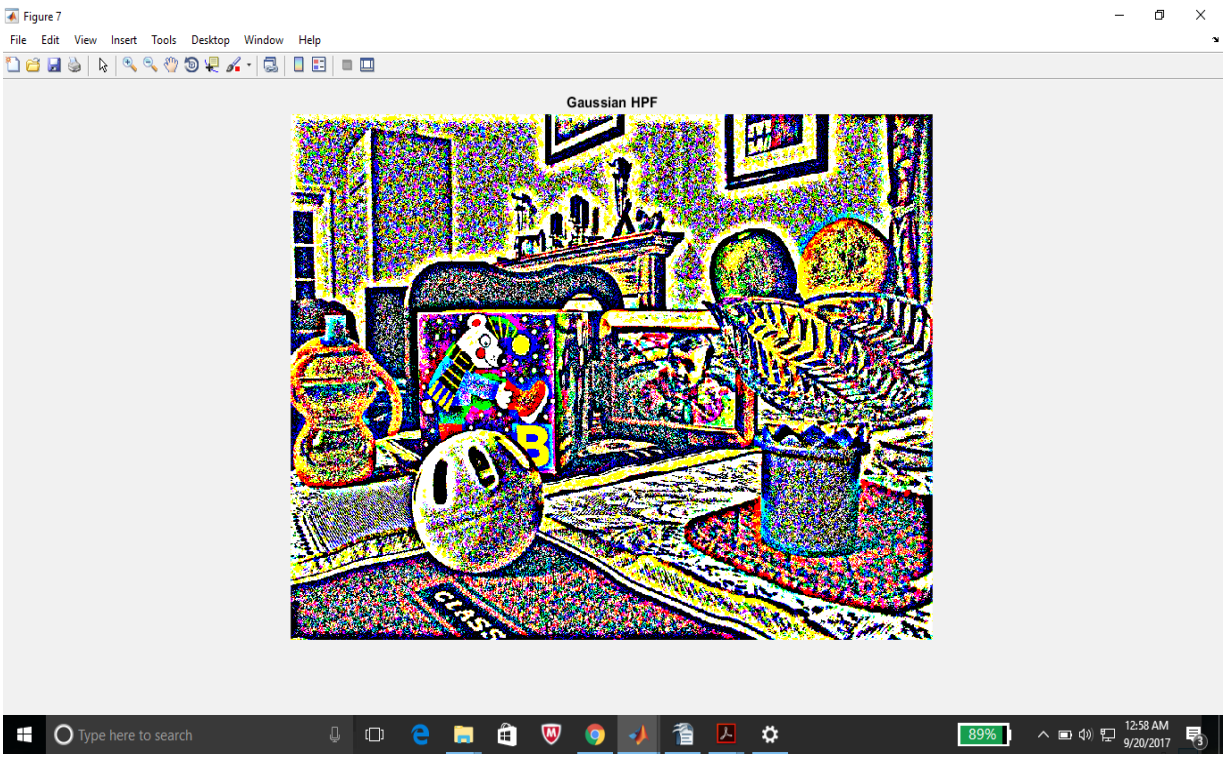
Output for Ideal High Pass Filter: D0 = 30



Output for Butterworth High Pass Filter: D0 = 30 and order = 6



Output for Gaussian High Pass Filter: Sigma = 20

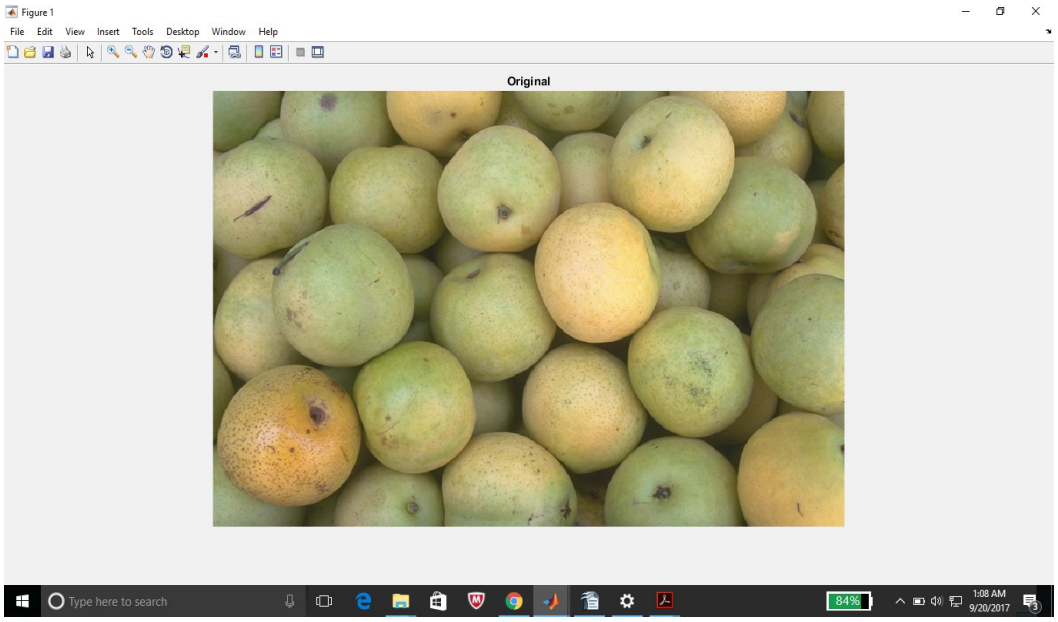


Observations: For ideal low pass filter, ringing effect can be noticed which distorts the image. If we keep the order of a Butterworth low pass filter high enough, the same ringing effect can be seen but for lower orders, the results are less distorted. Gaussian low pass filter gives more blurry output for a higher value of sigma. Note that the blurriness in the output of a Gaussian low pass filter is generally not as high as that in the ideal low pass filter. The Butterworth filter gives intermediate results depending on the order chosen. For the ideal high pass filter, the edges are usually more blurred (not as sharp) when compared to Butterworth filter of order around 2-4. The Gaussian high pass filter gives an output which does has slightly more prominent edges as the blurring introduced is not that high. Higher sigma gives more blurred results for both low pass and high pass filters.

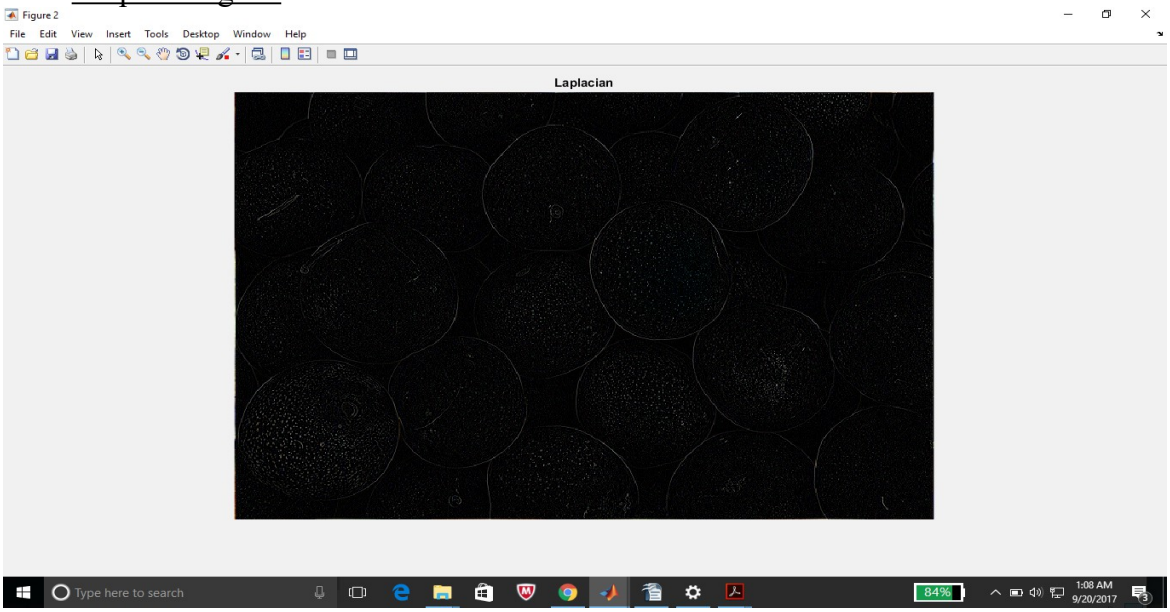
Results: Generally Butterworth or Gaussian low pass filters should be used with a moderate D0 value and a moderate sigma, respectively since they do not introduce the ringing effect. The order of the Butterworth filter should be small (2-4) to avoid ringing effect. Also, for high pass filters, the Butterworth filter of a slightly low order seems to give good results.

2c) Laplacian

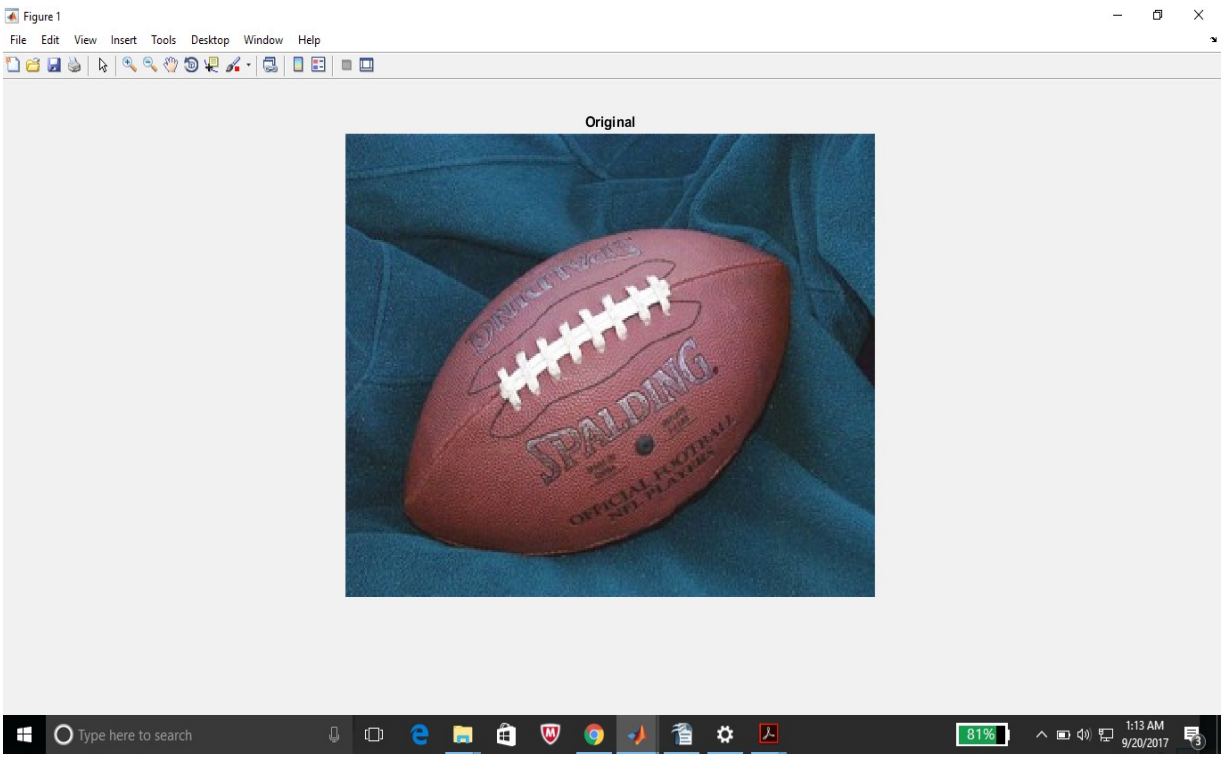
Input Image 1:



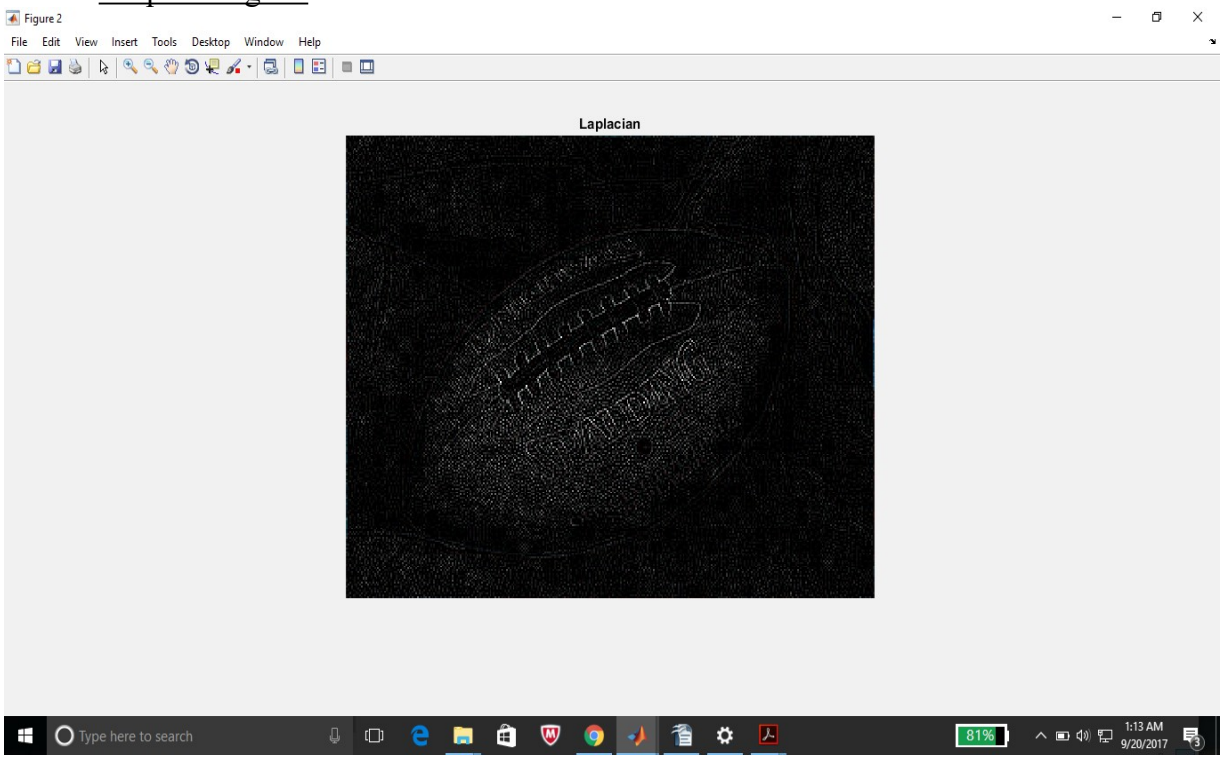
Output Image 1:



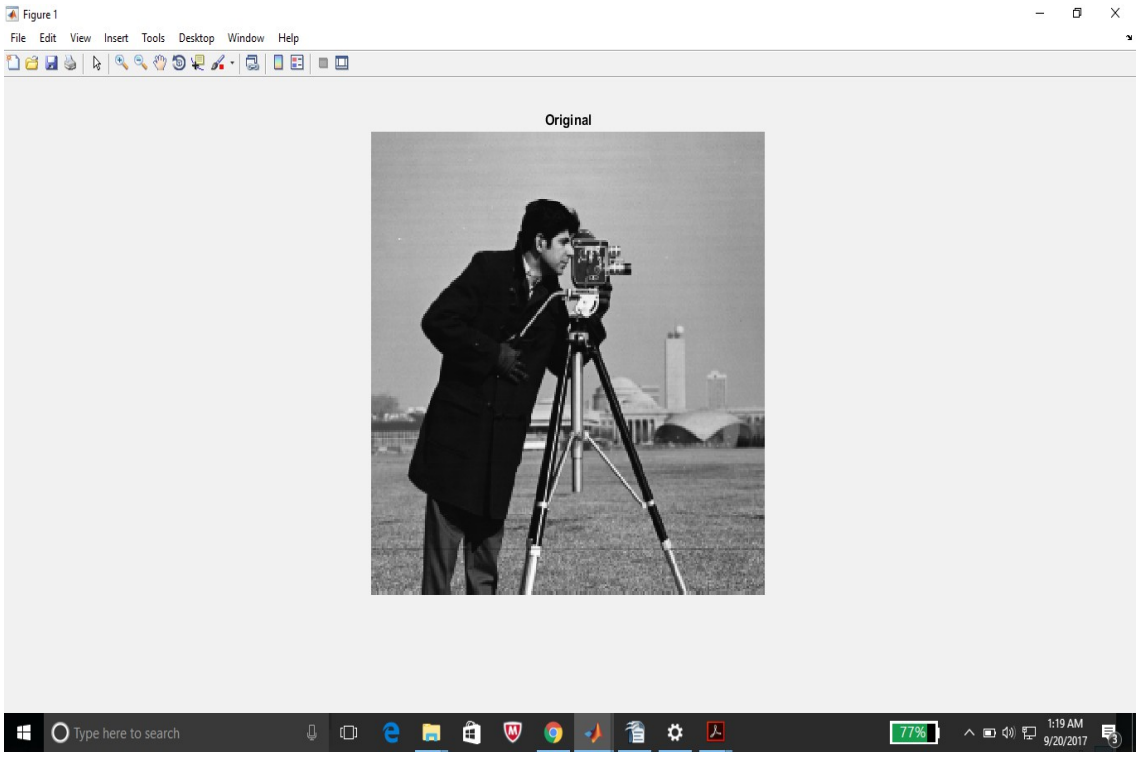
Input Image 2:



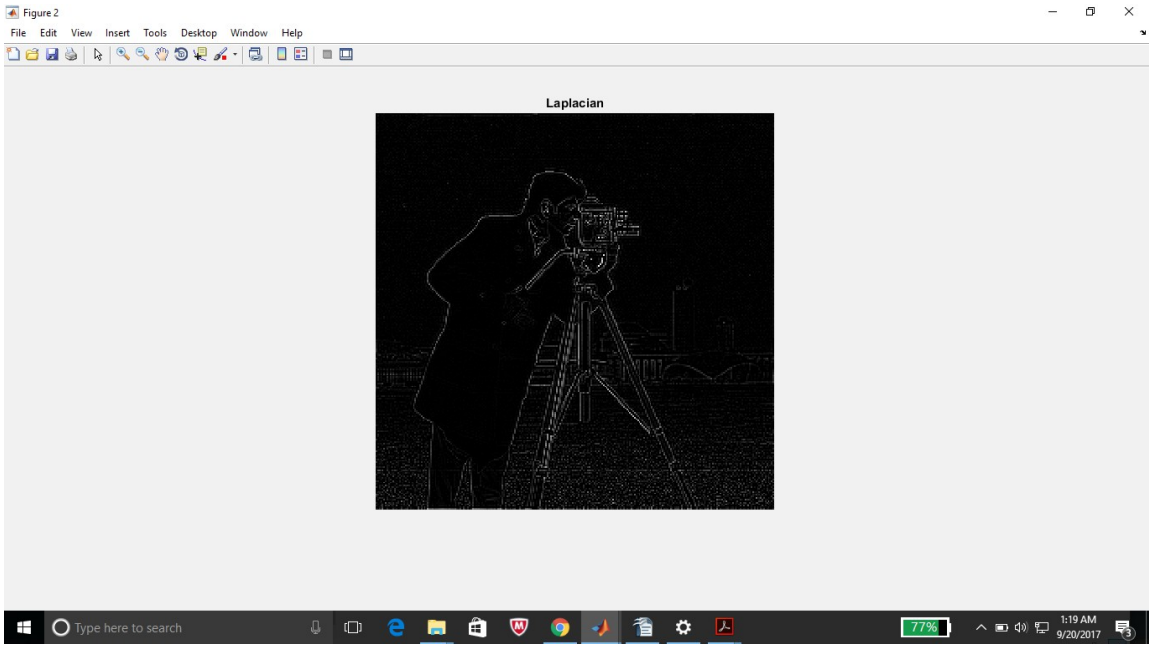
Output Image 2:



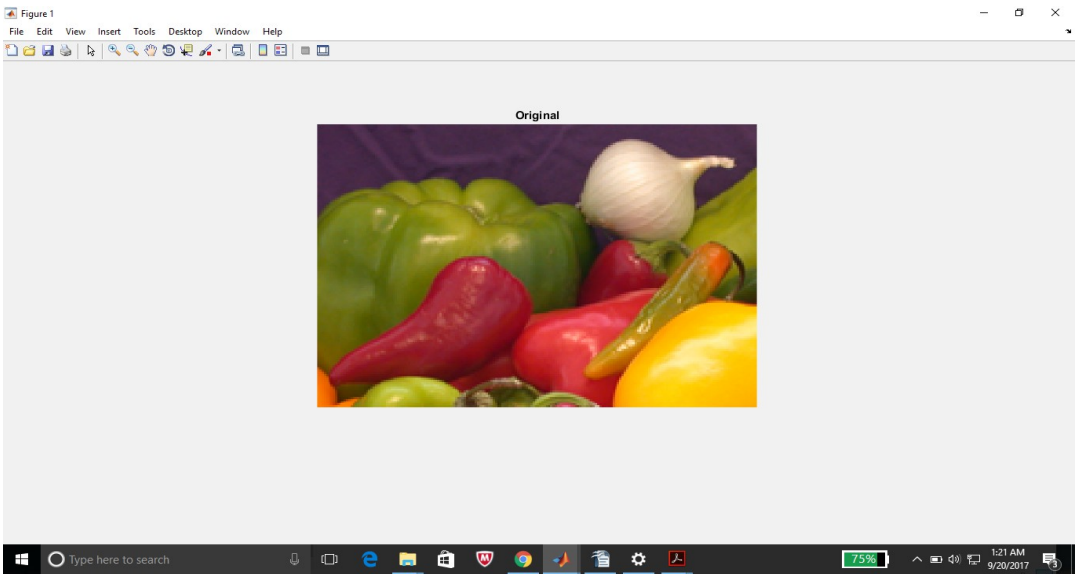
Input Image 3:



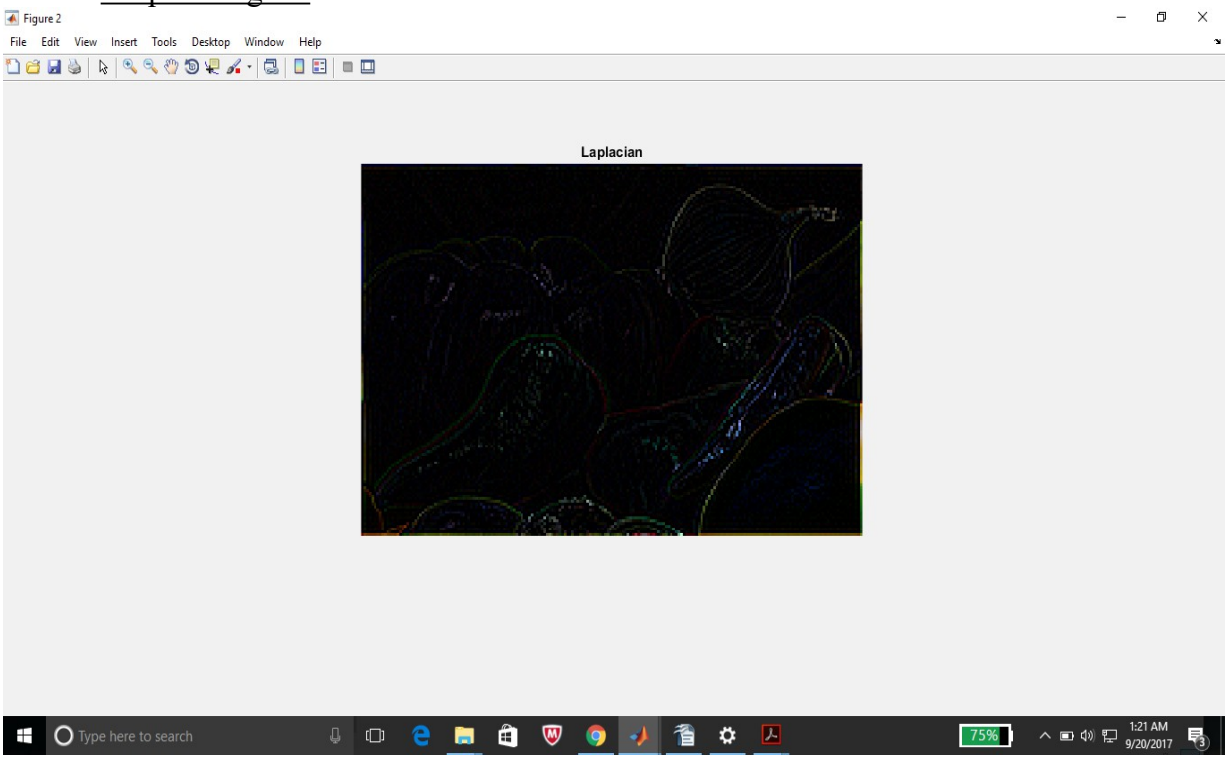
Output Image 3:



Input Image 4:



Output Image 4:

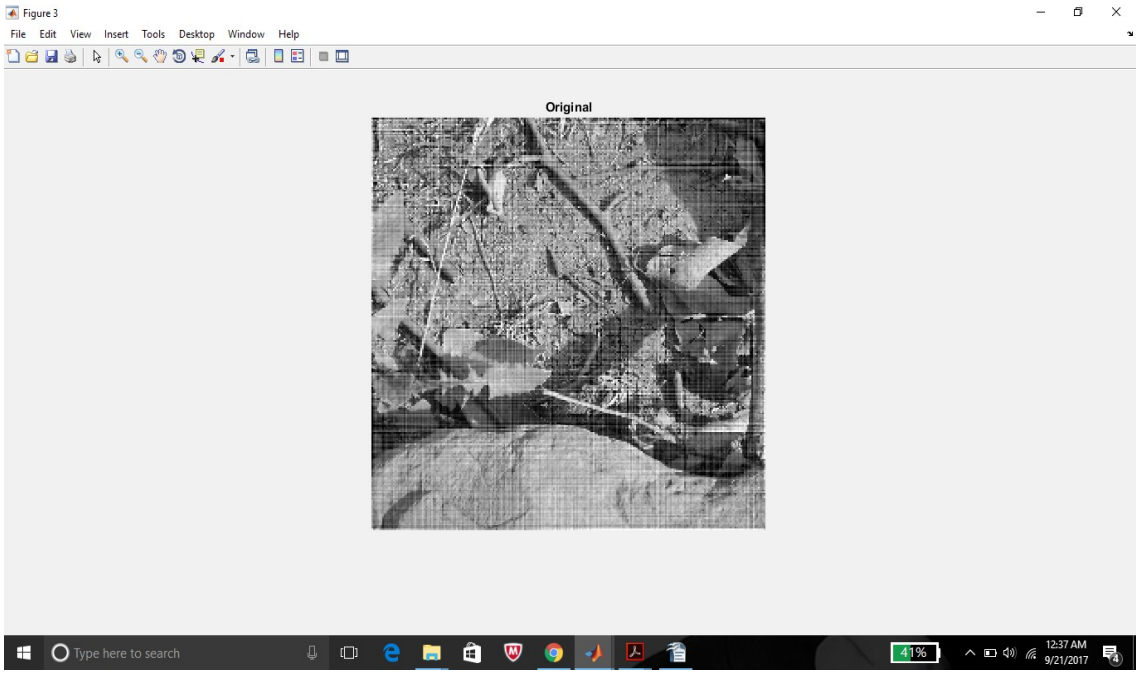


Observations: We can see that the laplacian of an image highlights the edges of the image. I also noticed that on multiplying the FFT of the image with -(u*u+v*v) gave large negative values and the way to offset this was to take the IFFT and then subtract the minimum value from the image and then multiply the new image by 255/maximum value.

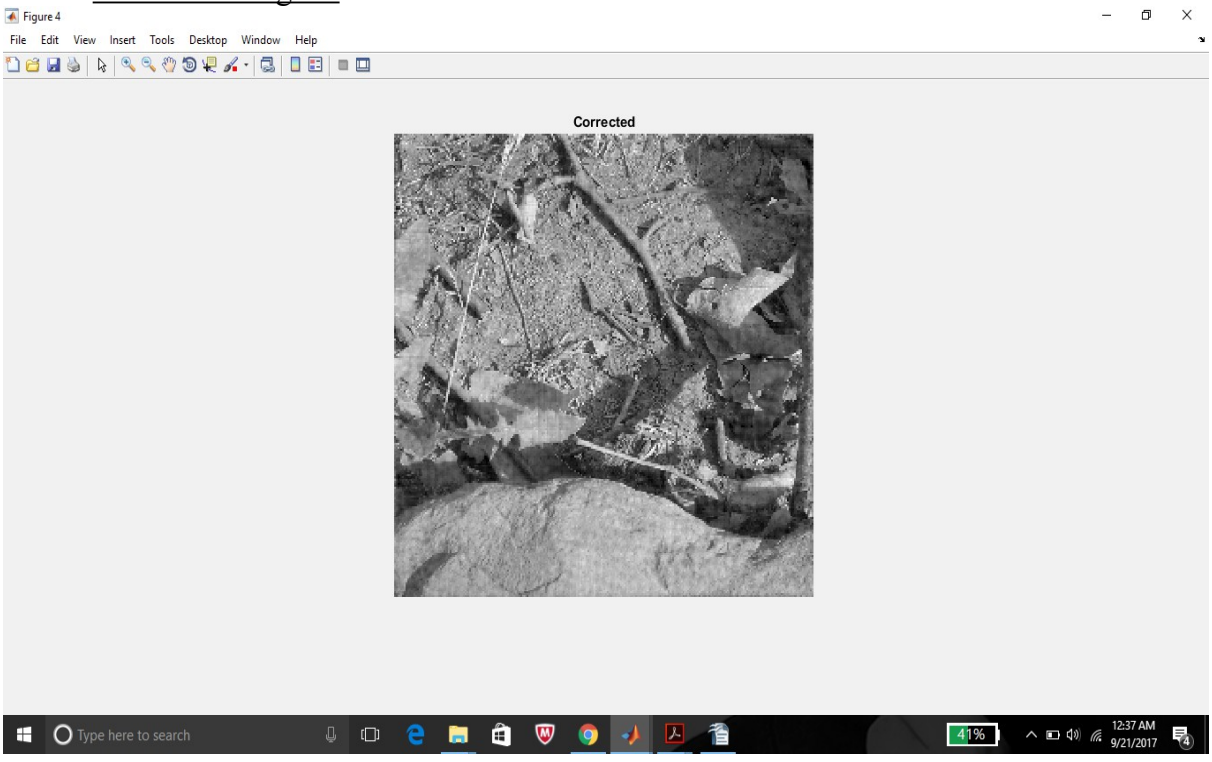
Results: The laplacian of the image gives the edges present in that image.

2d) Notch Pass Filter

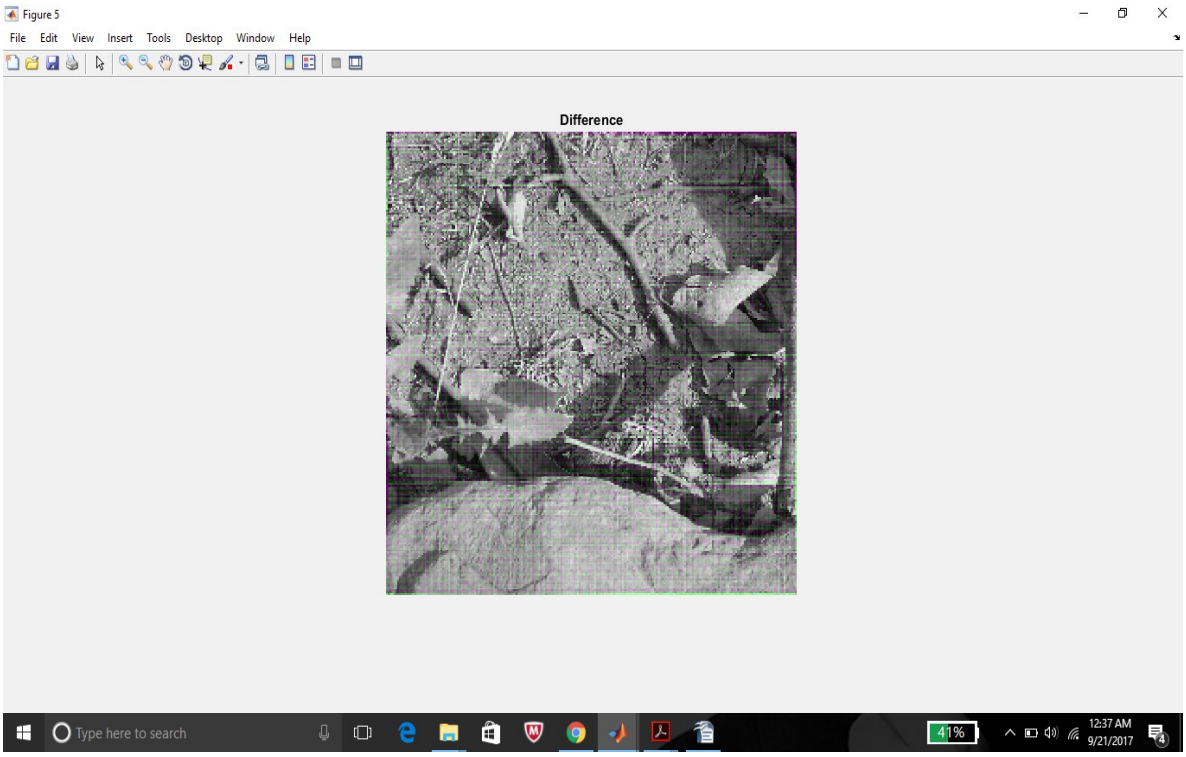
Input Image 1:



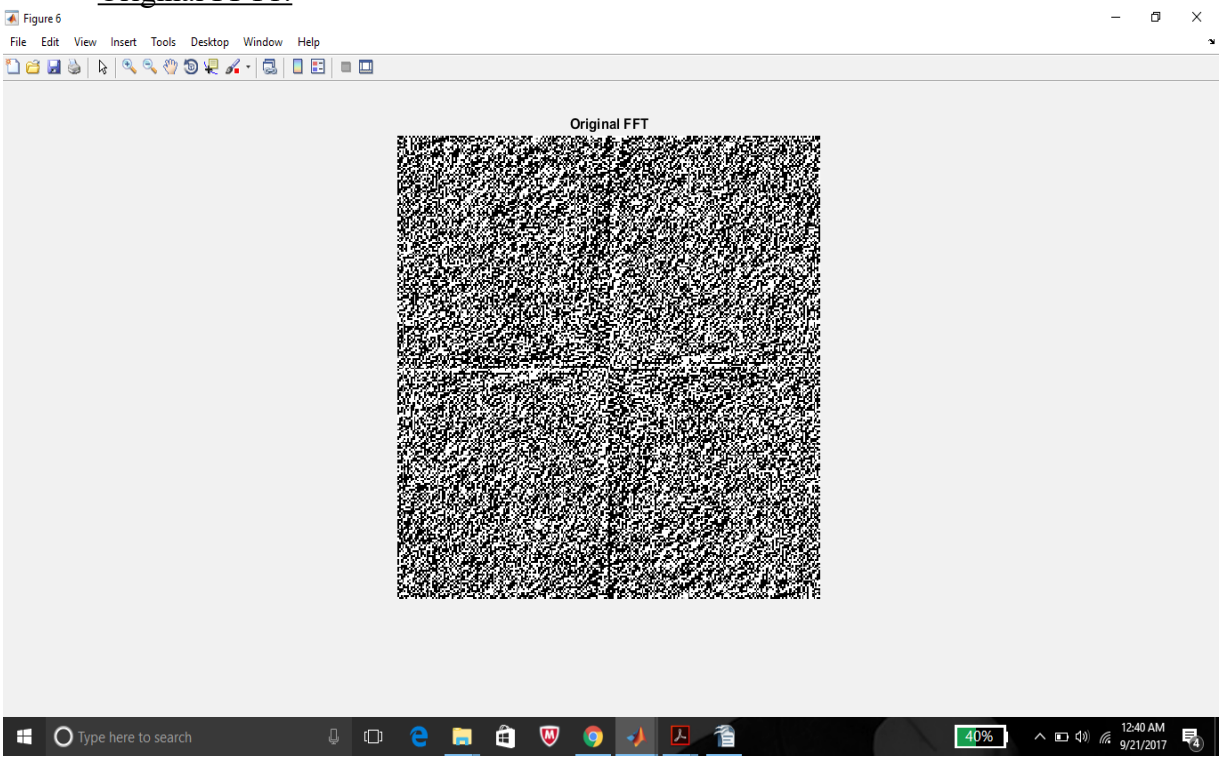
Corrected Image 1:



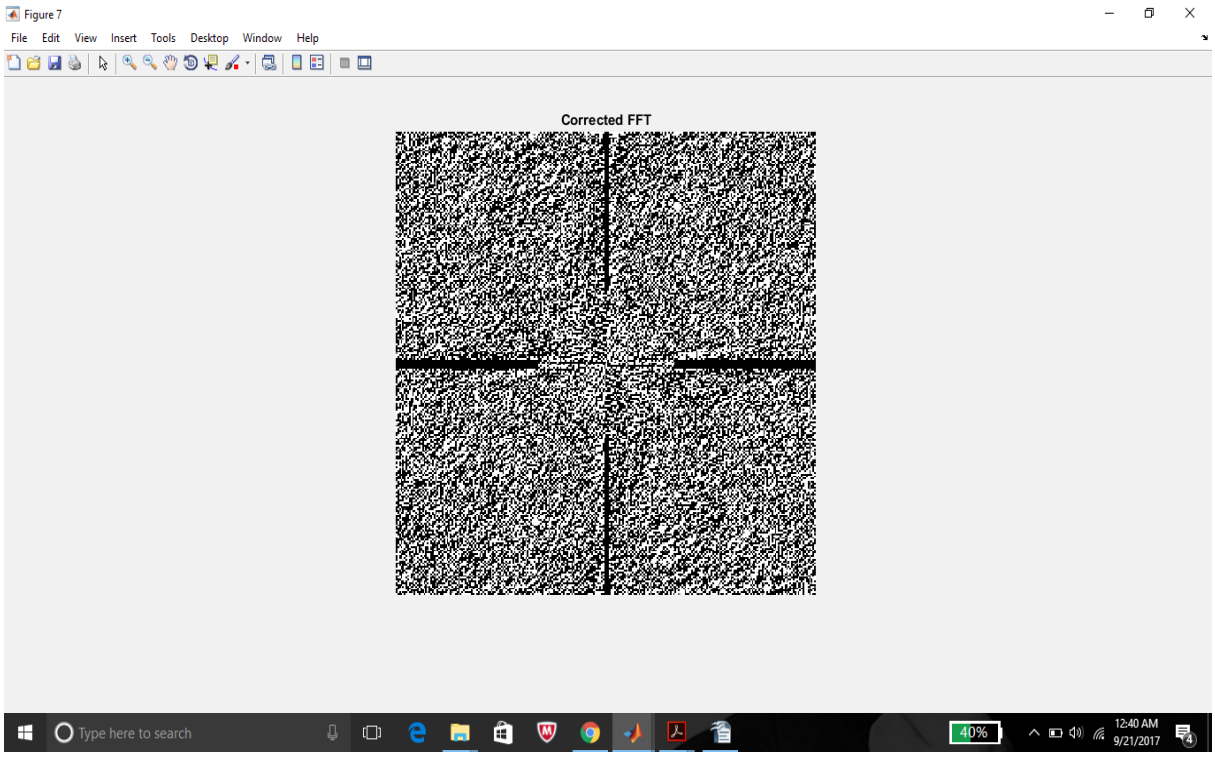
Difference1:



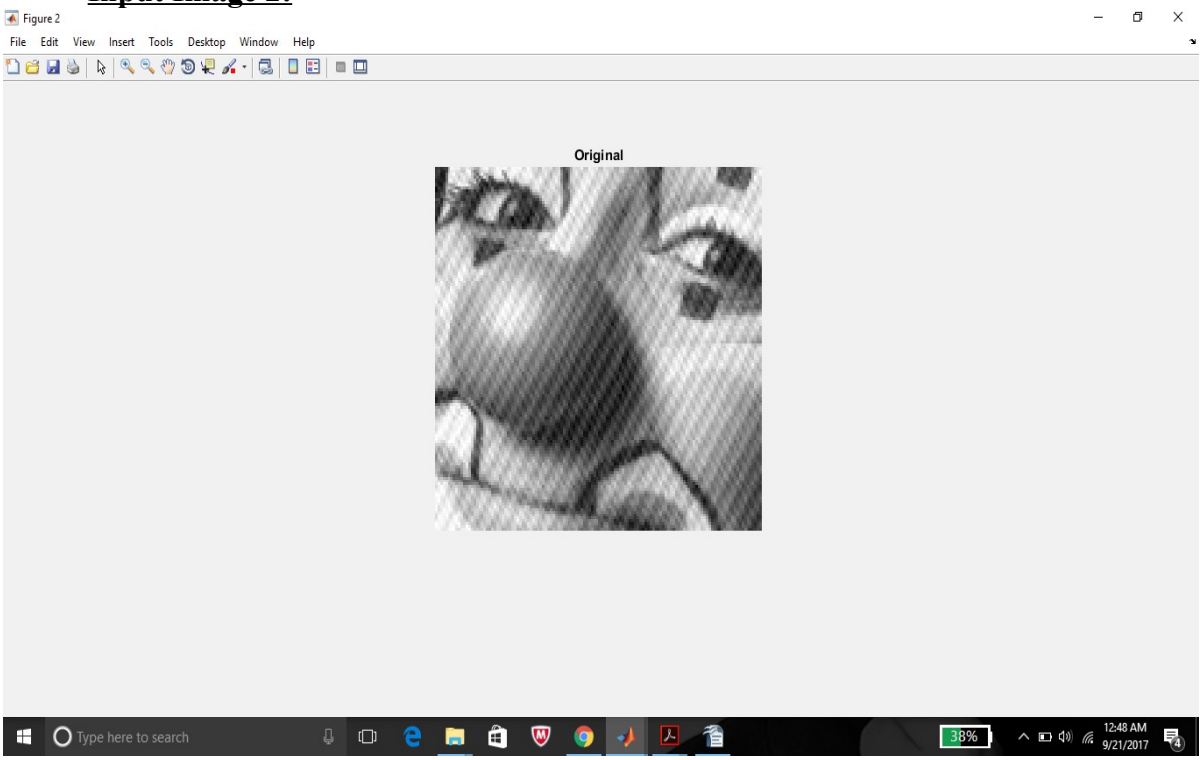
Original FFT1:



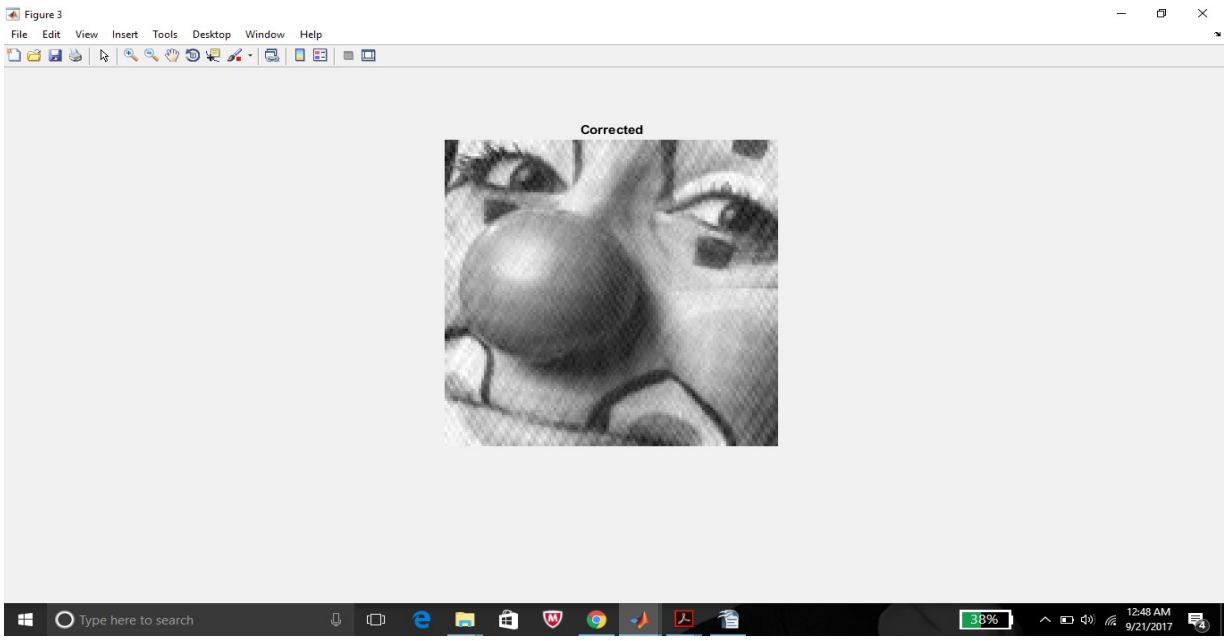
Corrected FFT1:



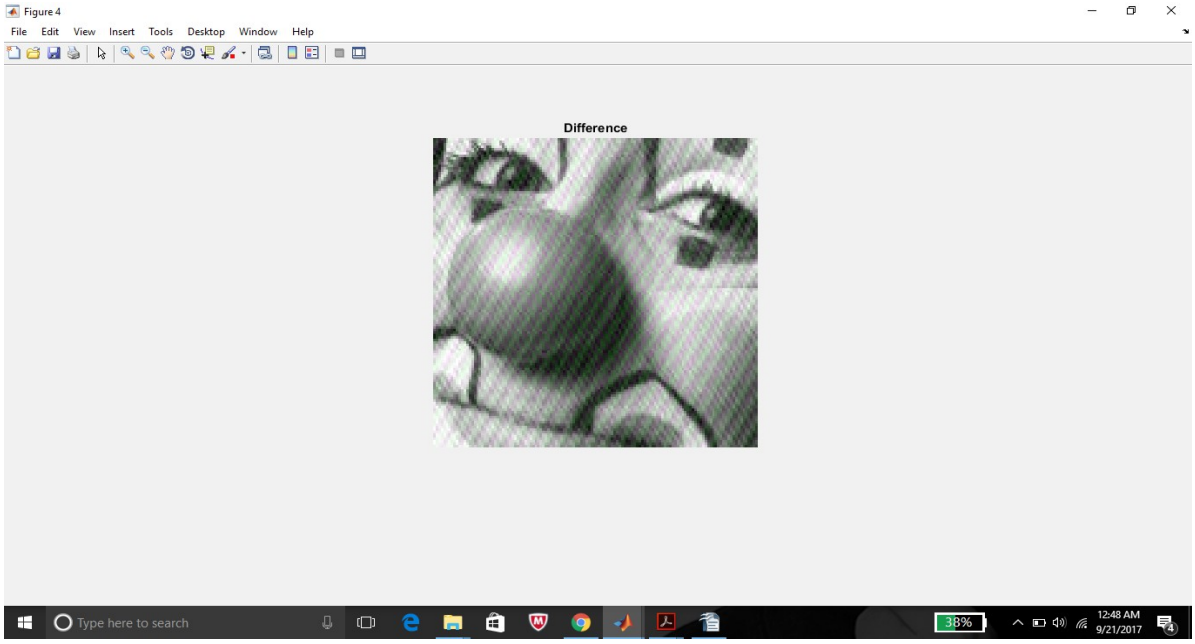
Input Image 2:



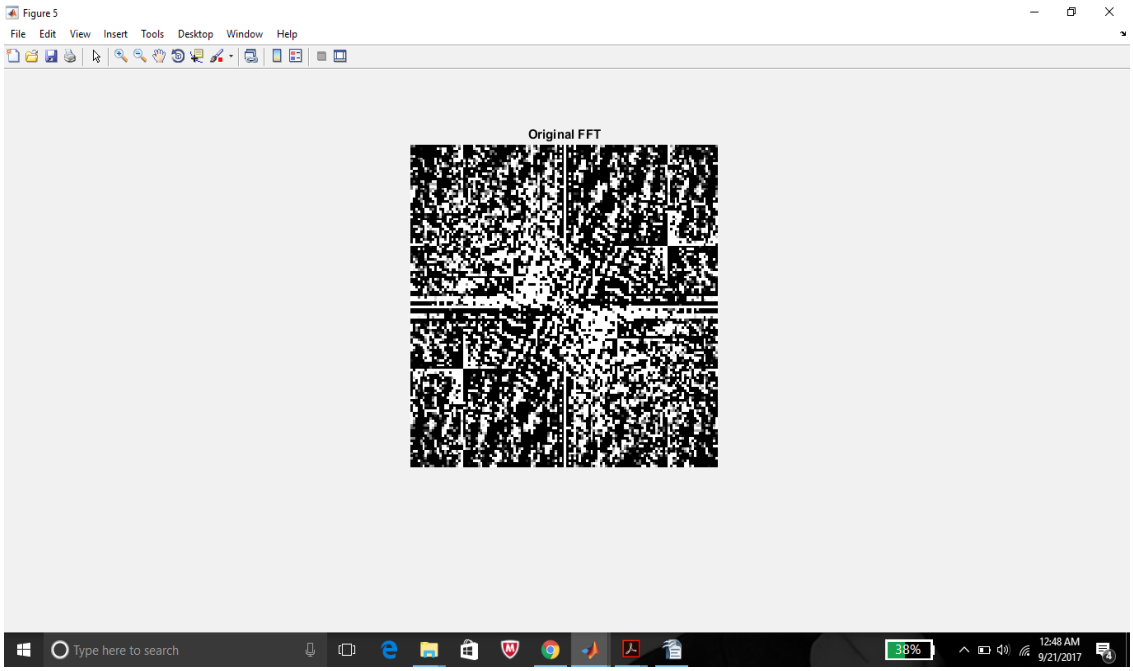
Corrected Image 2:



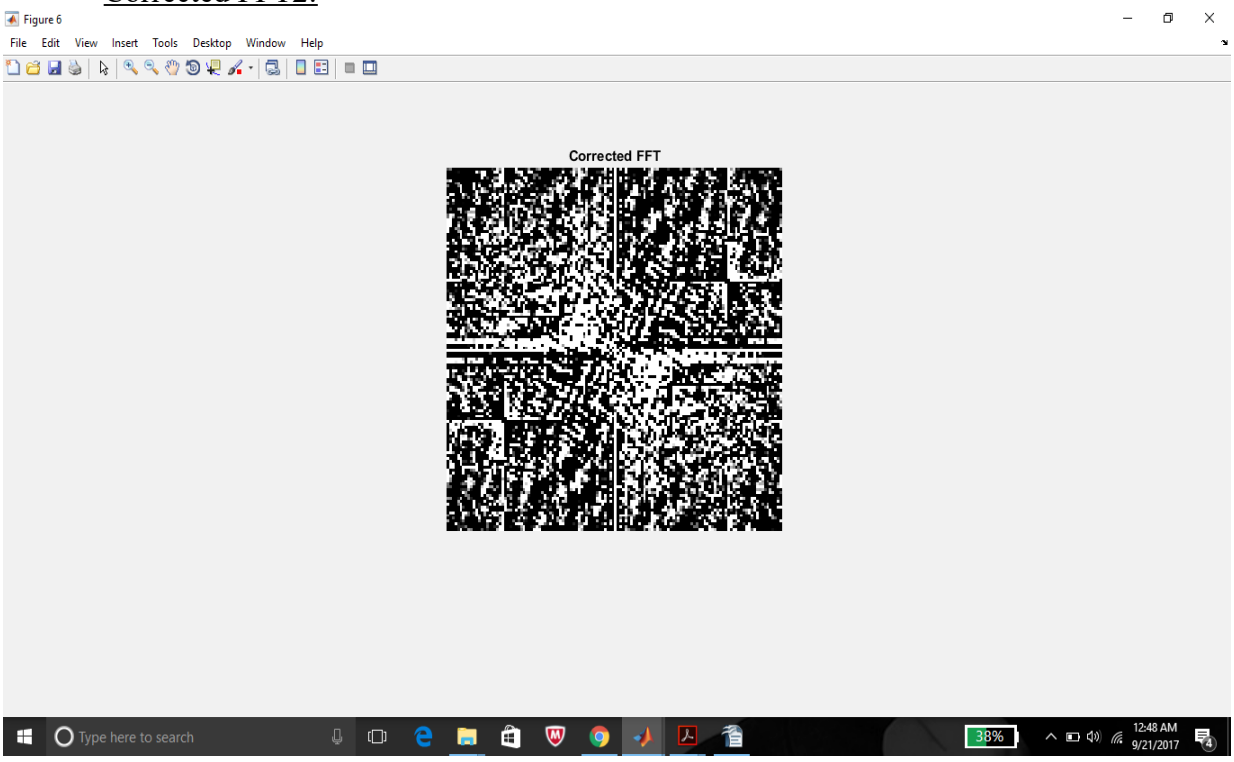
Difference2:



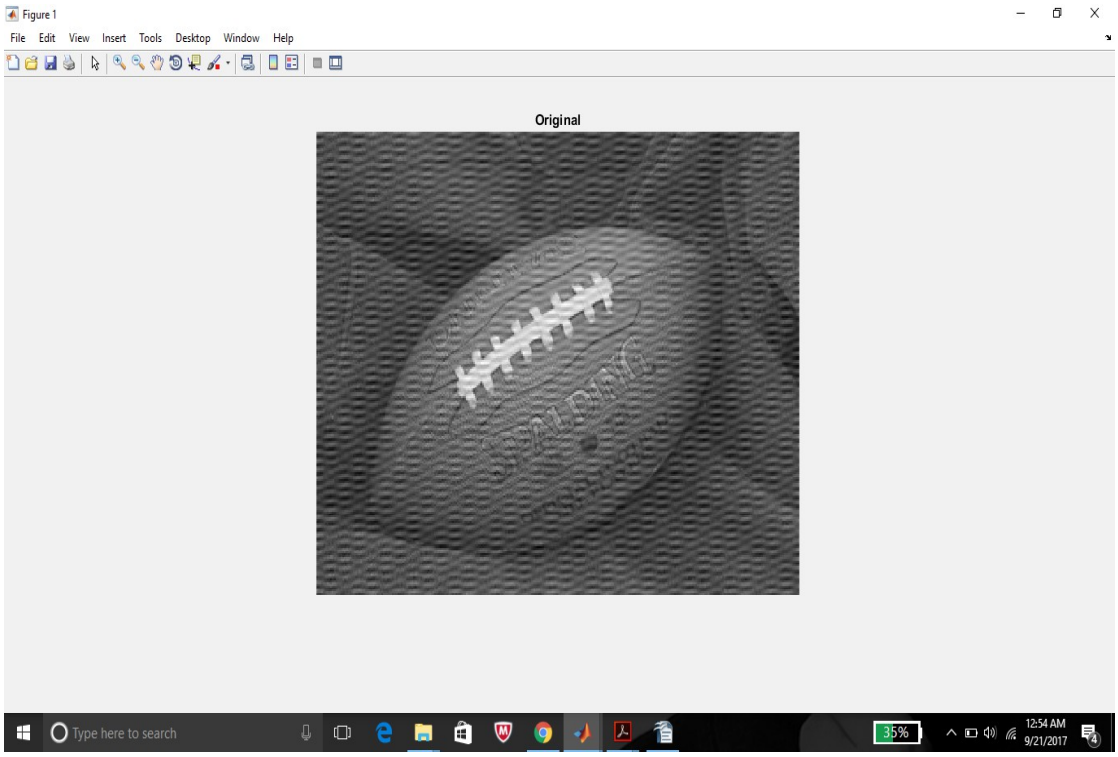
Original FFT2:



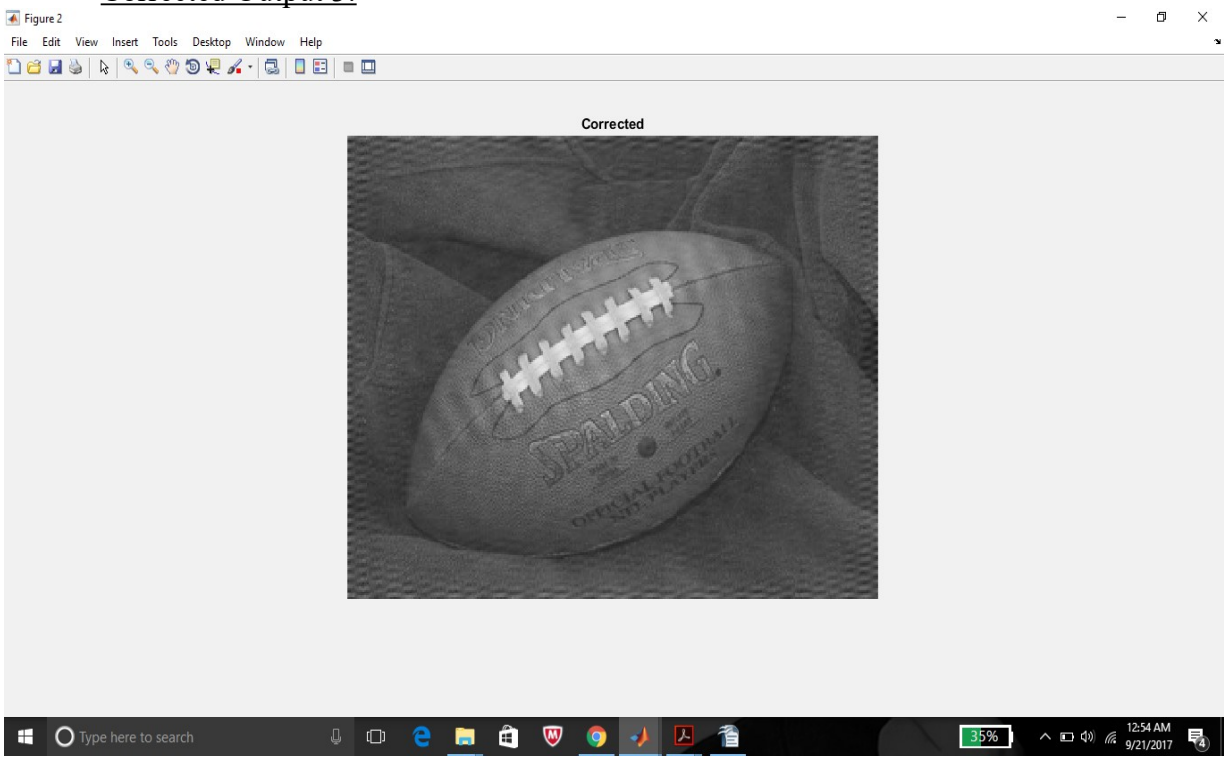
Corrected FFT2:



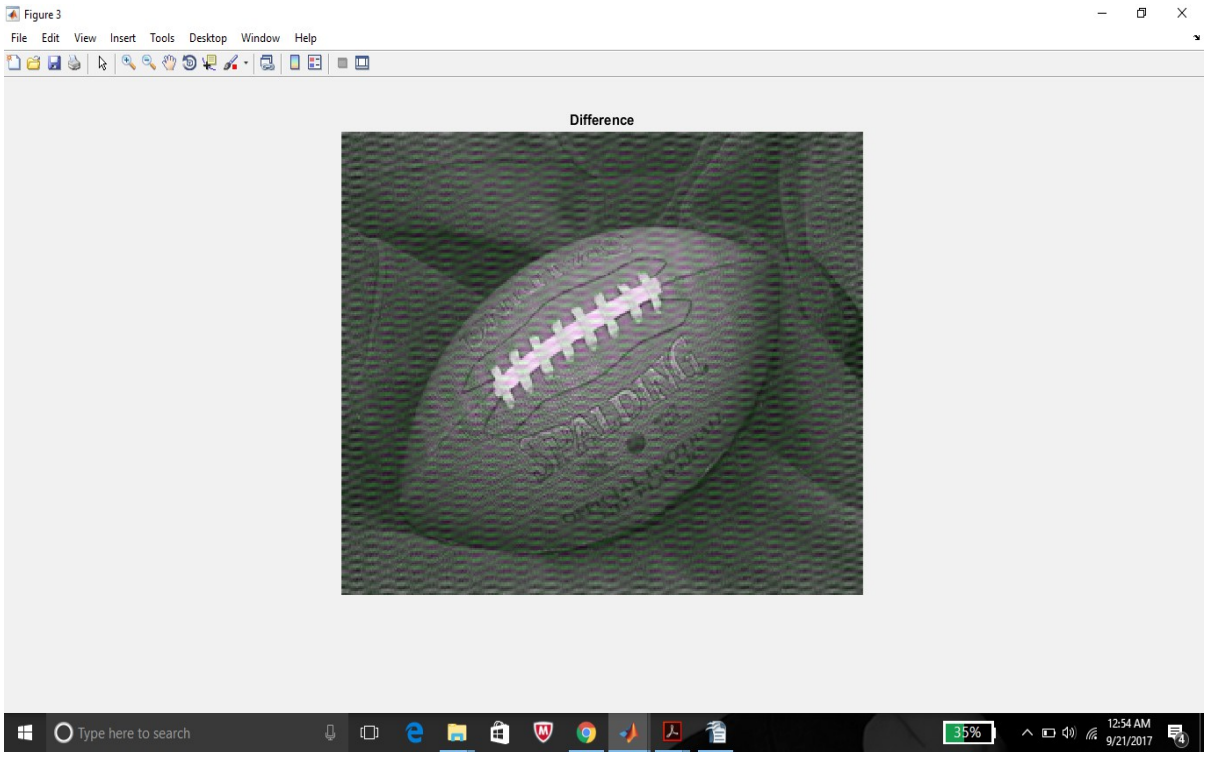
Input Image 3:



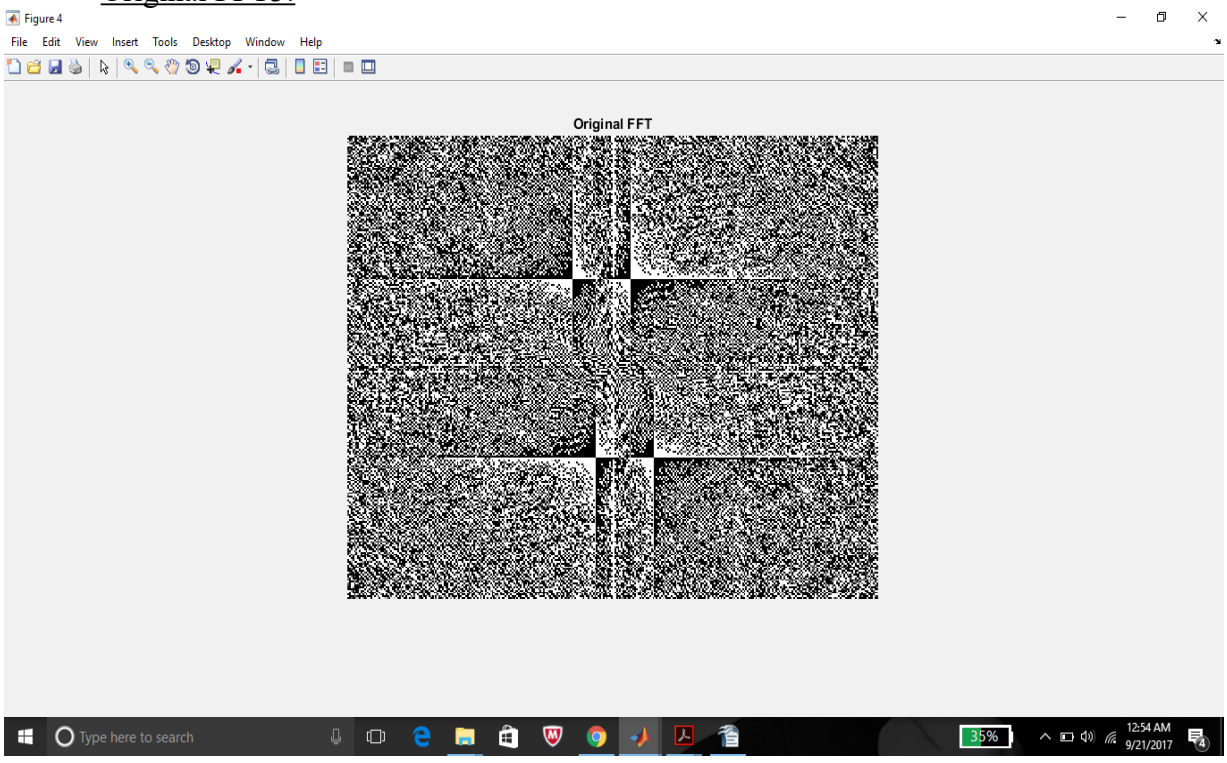
Corrected Output 3:



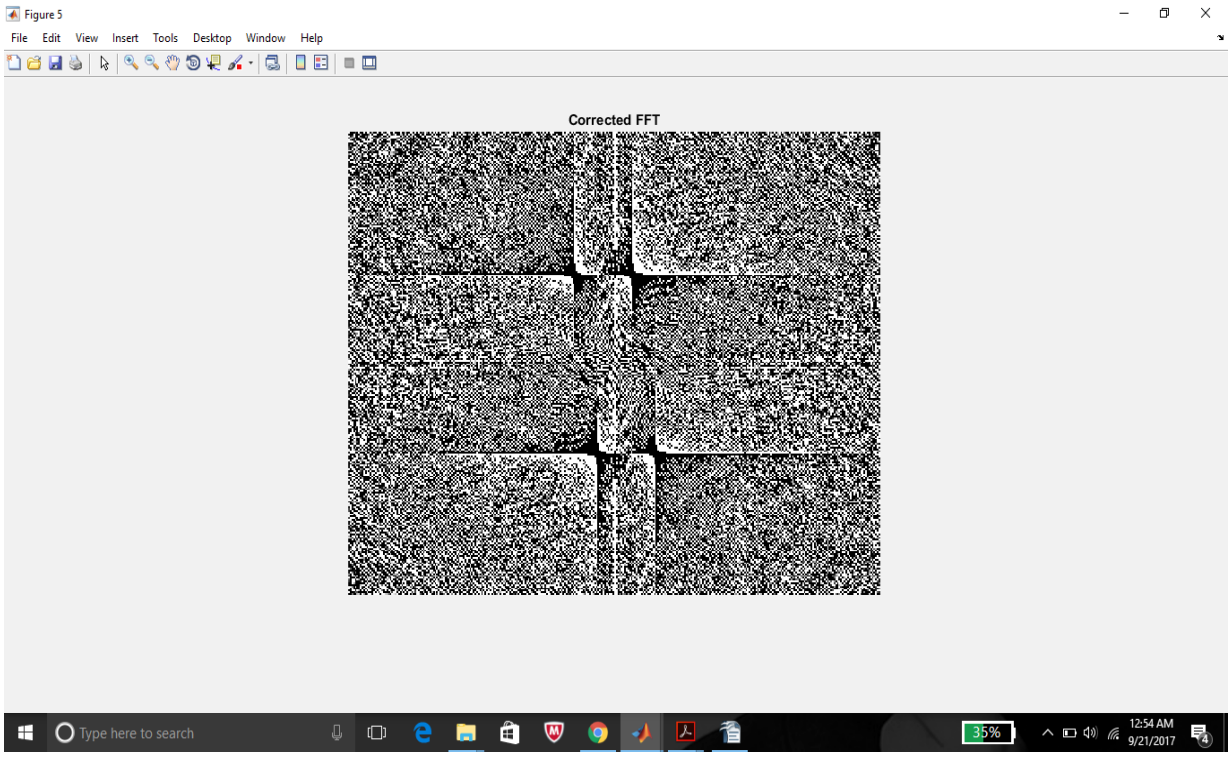
Difference3:



Original FFT3:



Corrected FFT3:



Observations: It was noticed that if the notch filter boundary was not chosen properly, the output image would either be noisy or would be very smooth and have spots. This is why the notch filter must be carefully designed individually for each of the images.

Results: The corrected outputs show less noise to a certain extent while the corrected FFTs are slightly changed.