Question 1

Part 1

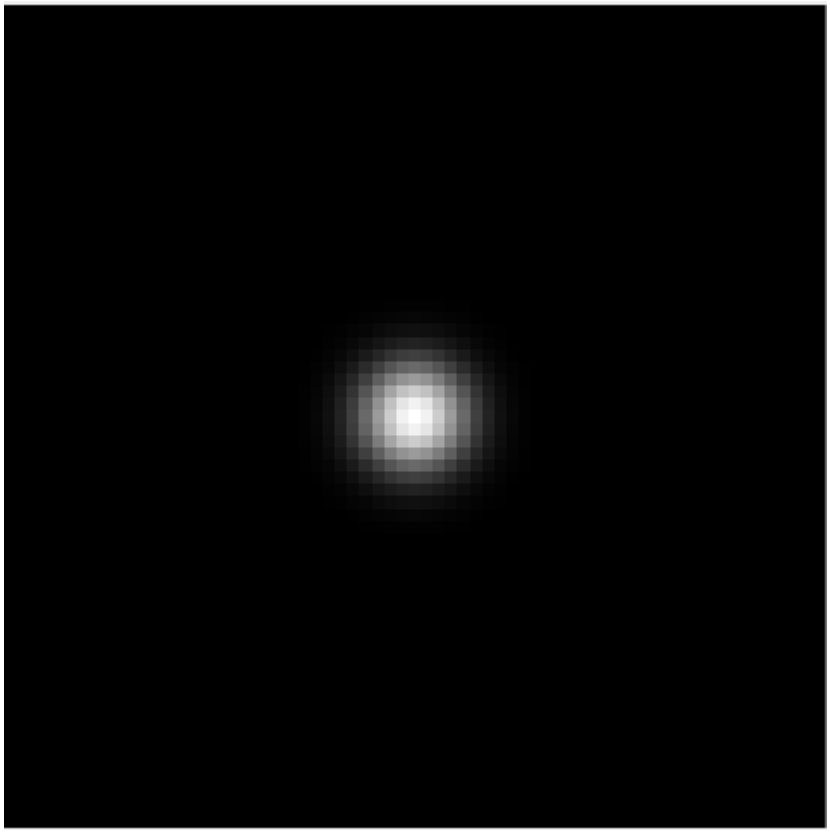


Image of Gaussian filter of kernel size = 67 and standard deviation = 3

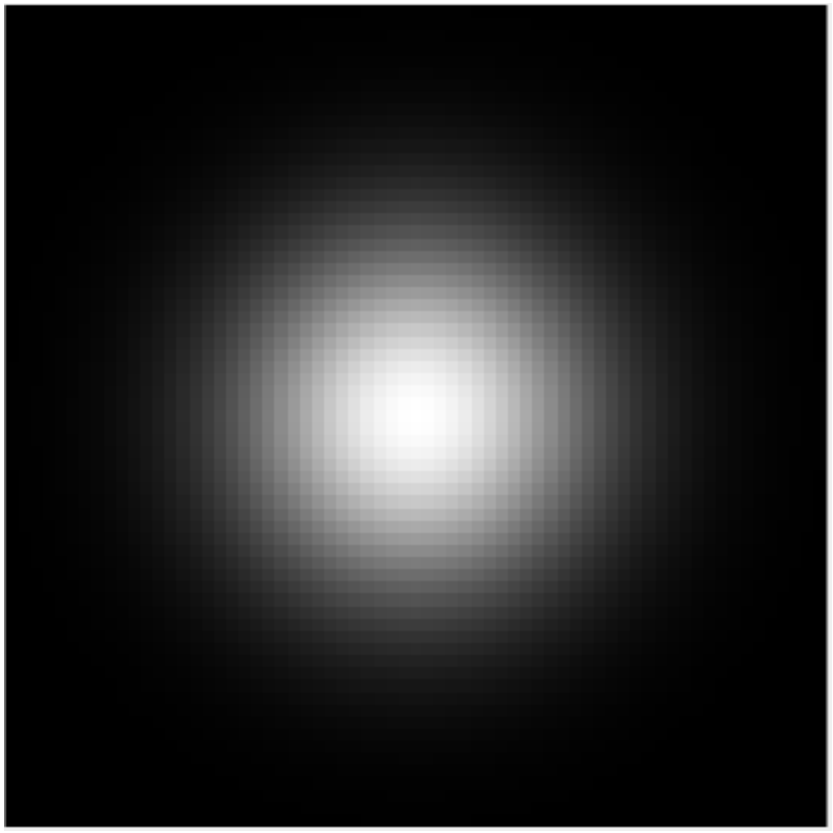


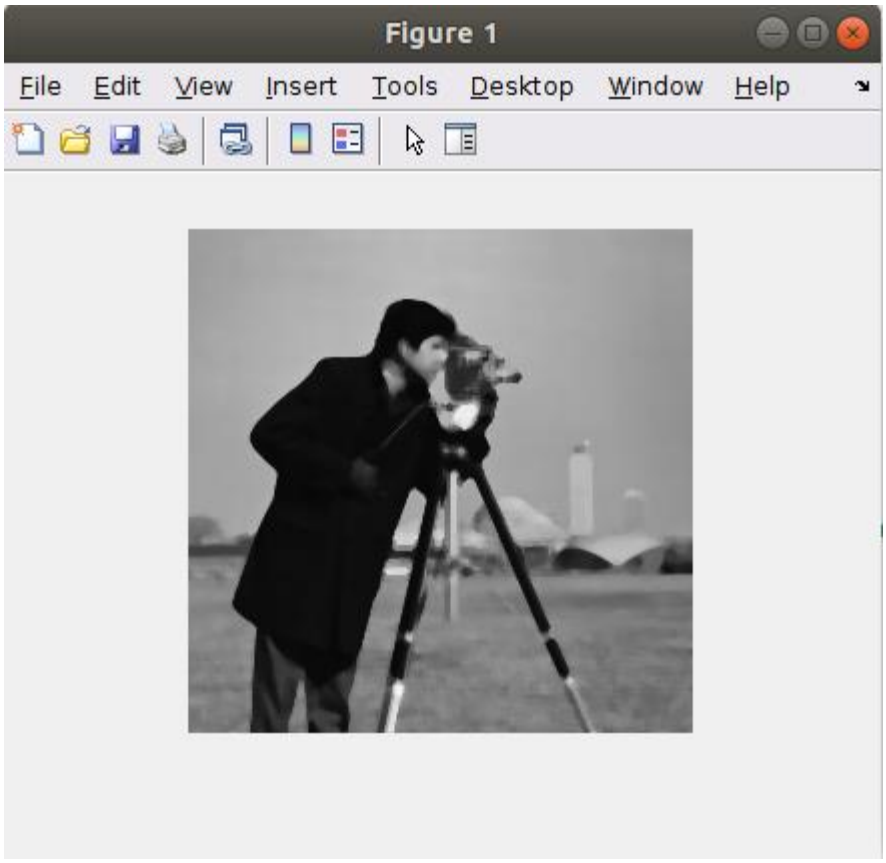
Image of Gaussian filter of kernel size = 67 and standard deviation = 10

The size of the blob increases with increase of standard deviation. The image height/width is directly proportional to the kernel size. Infact each pixel of the image maps to a particular element of the generated kernel matrix.

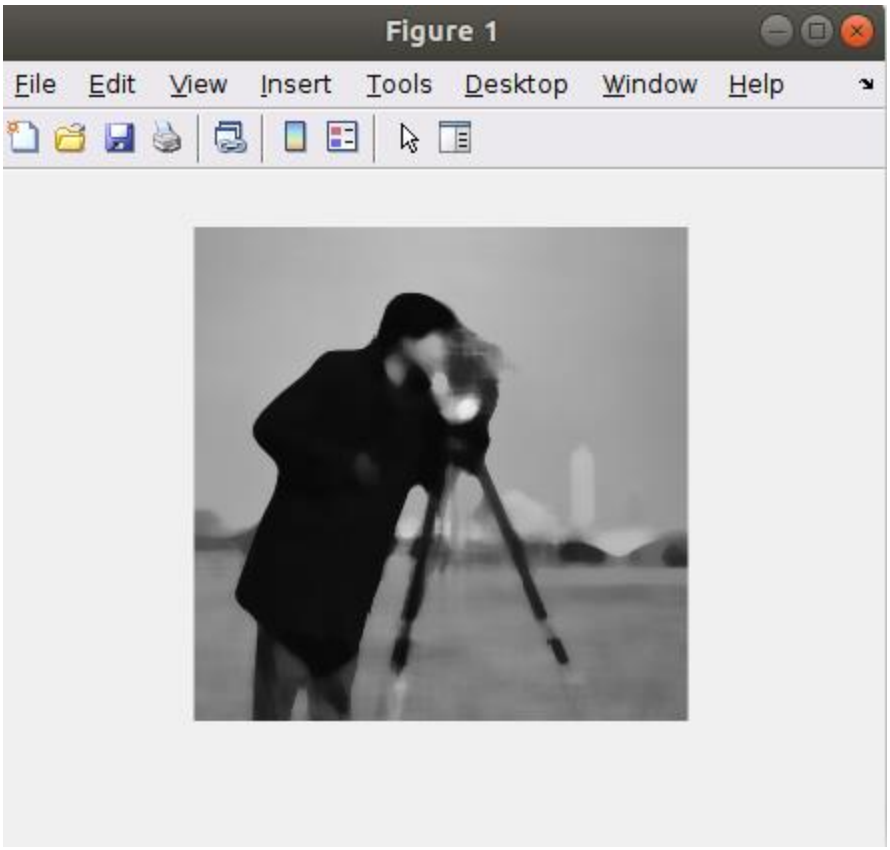
For large values of N, fspecial() Gaussian has corner values as 0s whereas my code has values of the order 10^-22. Therefore, fspecial() is more optimized.

Part 3

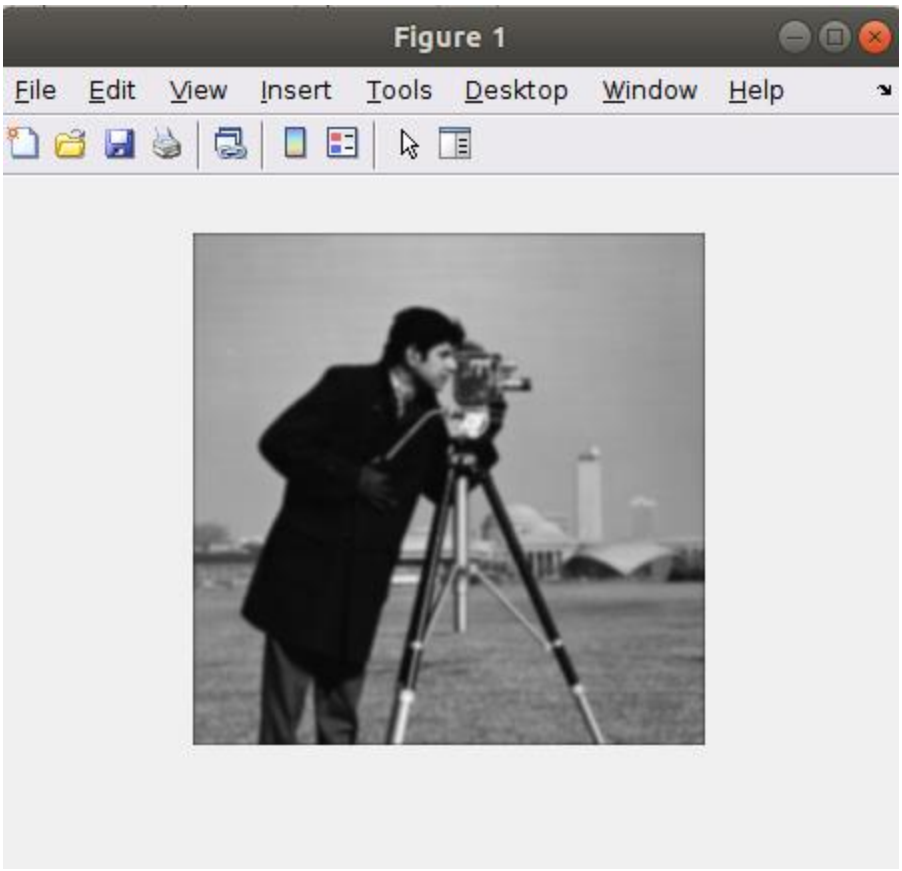
Median Filter with Kernel size 1



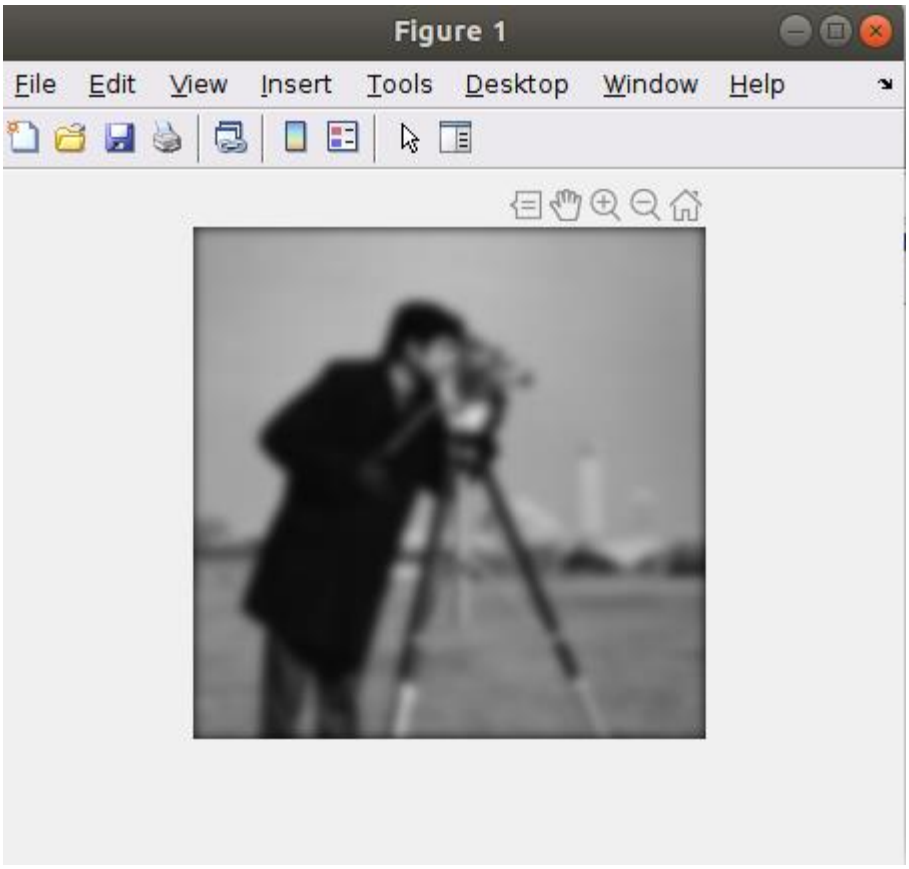
Median filter with kernel size 5



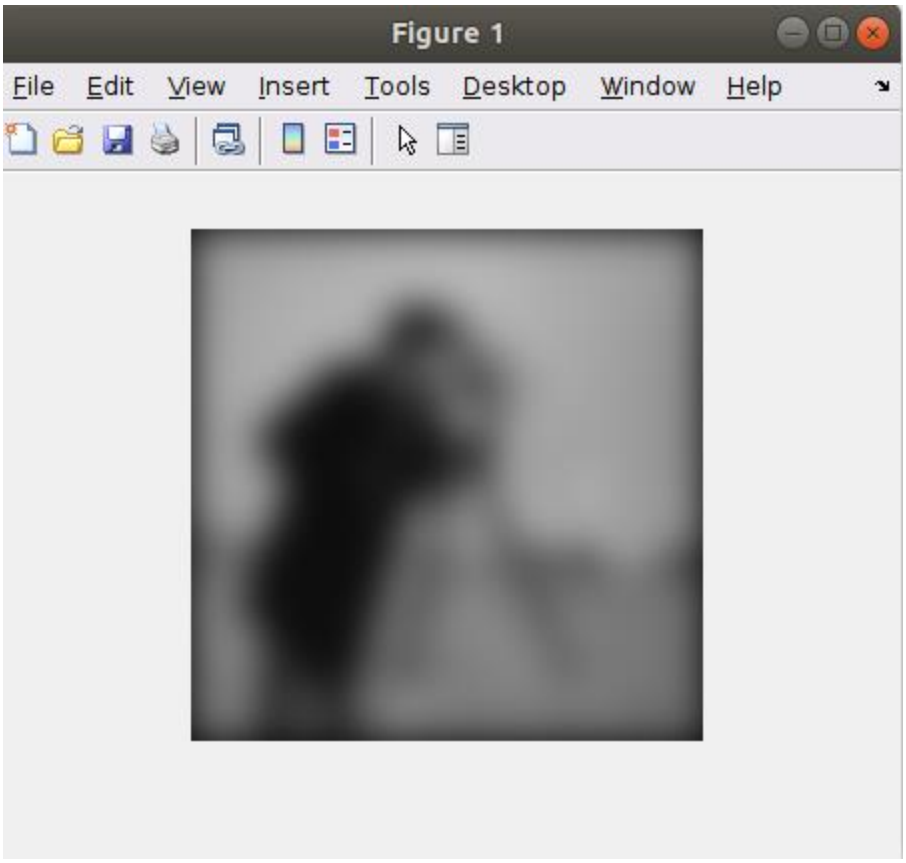
Median filter with kernel size 10



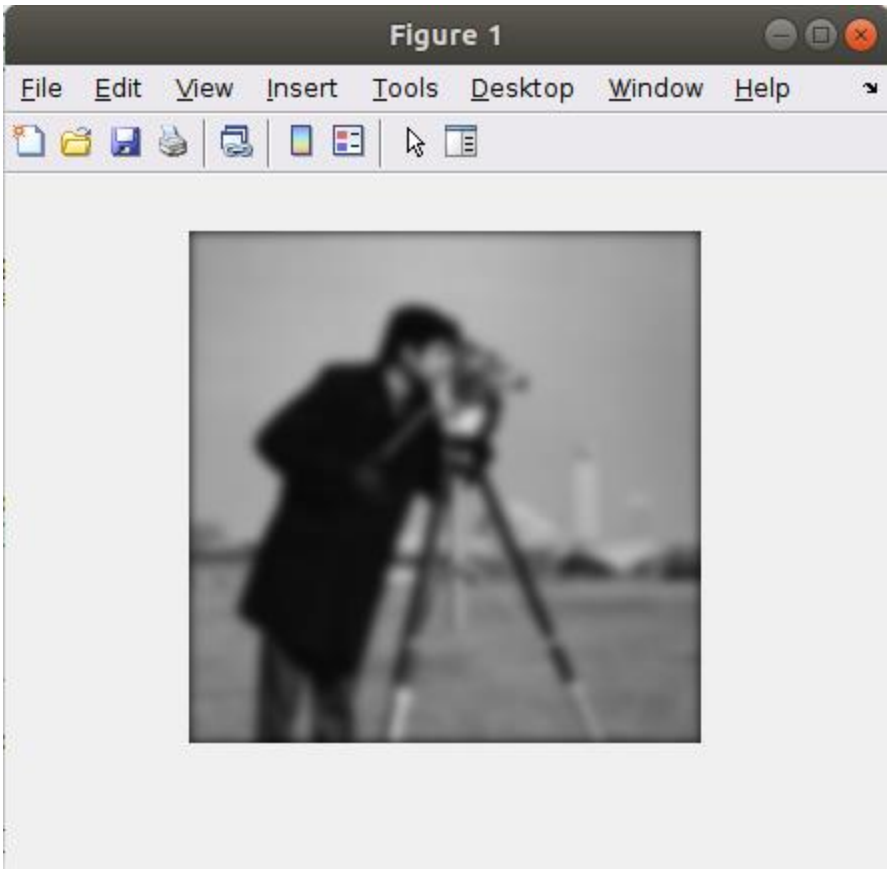
Gaussian filter with kernel size = 67 and standard deviation = 1



Gaussian filter with kernel size = 67 and standard deviation = 3

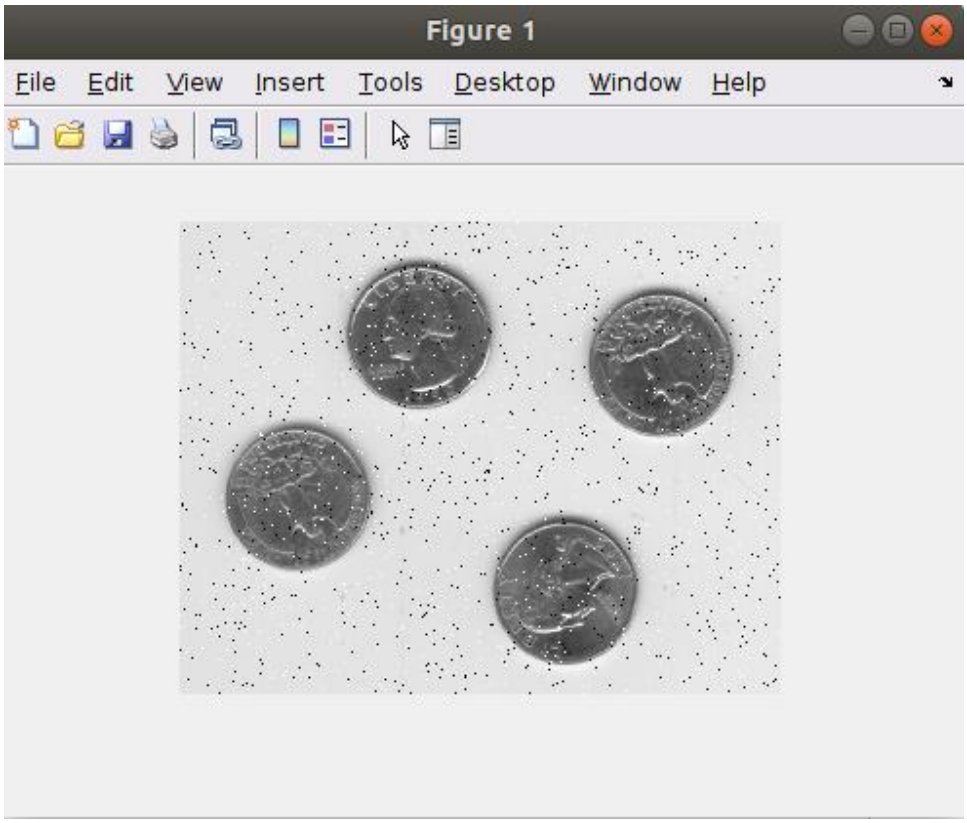


Gaussian filter with kernel size = 67 and standard deviation = 10



Gaussian filter with kernel size = 100 and standard deviation = 3

Part 4



Original image with salt and pepper noise

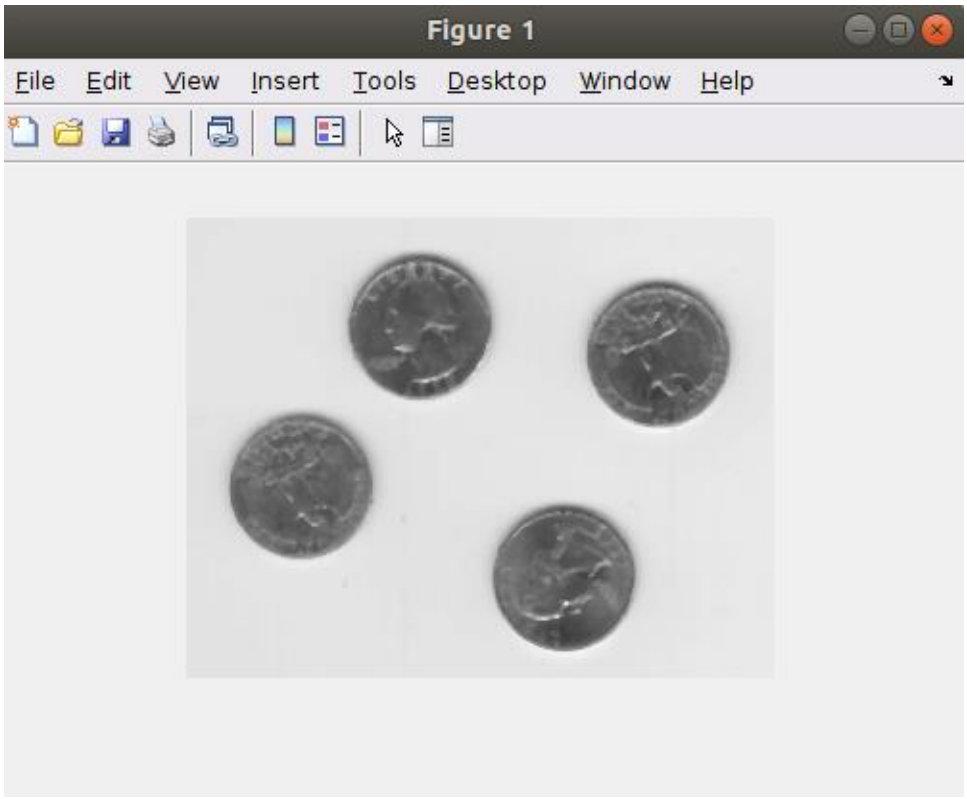


Image noise removed by applying median filter with kernel size = 3

The median filter is most suitable for this type of noise(called the salt and pepper noise). The median filter takes pixel with the median value in a surrounding and replaces the original pixel value with it. This is helpful as in salt and pepper noise only specific pixels show really low value(corresponding to the tiny black dots). These black dots are replaces with the pixel value with median value in that surrounding(surrounding size depends on kernel size) and as a result the low values of the tiny dots are increased thereby making the tiny black dots disappear.

Part 5

We take the FFT of the image and plot the spectrogram. We clearly see periodic white blobs along first 2 columns and last two columns (corresponding to 0 frequency along row). These white blobs correspond to the sinusoids (of particular frequency along column and zero frequency along row) giving rise to the strips. We try removing the white blobs manually. Then we compute the IDFT to get back the filtered image.

Original Image

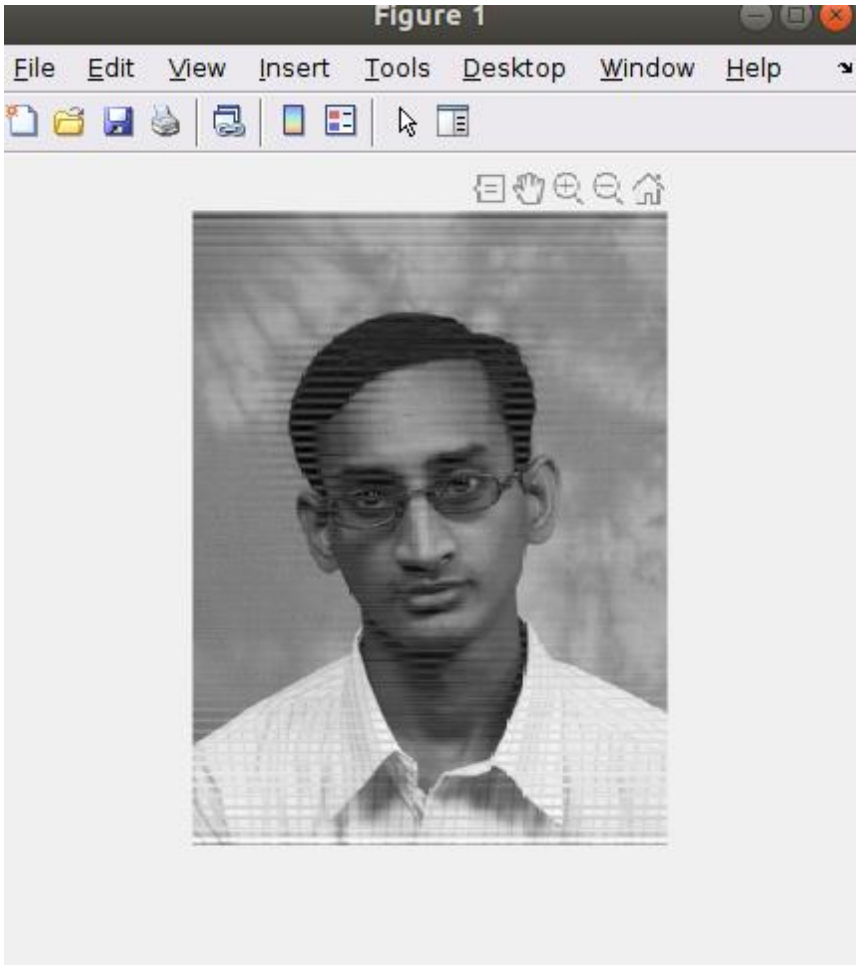


Image after removing noise

Question 2

We solve this problem by recursion.

a) Let W[i], H[i], denote the width, height after applying the ith filter.

Base Case:

W[0] = W;

H[0] = H;

Recursive Case:

W[i] = ceil [(Z + W[i - 1] - F) * 1.0 / S];

H[i] = ceil [(Z + H[i - 1] - F) * 1.0 / S];

For all i such that, 1 <= i <= N

Dimensions of the output of this convolution = <W[N], H[N], Channels>

b) Number of additions and multiplications = res, were res is calculated by the following code segment:

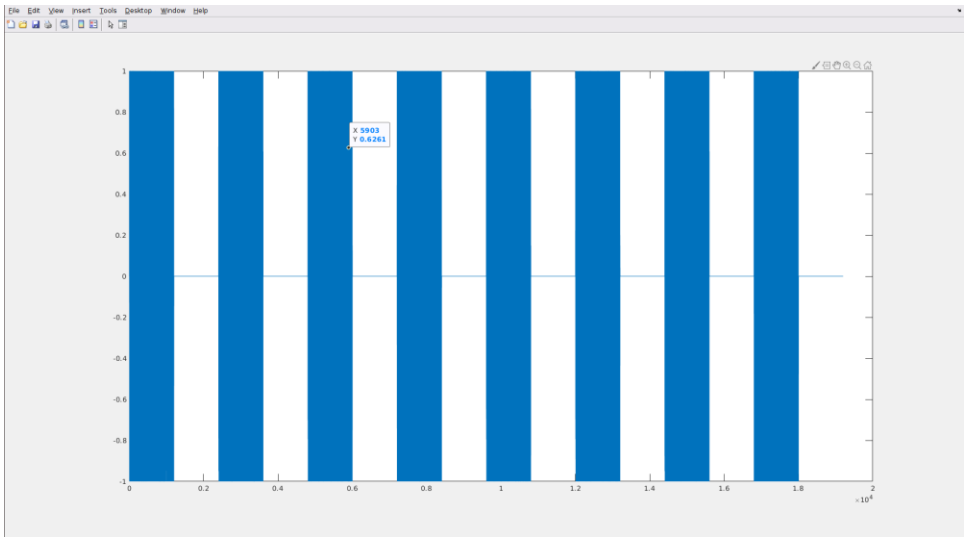
for i = 1 to N:

*res = res + (F * F * Channel * Width[i] * Height[i])*

end

Question 3

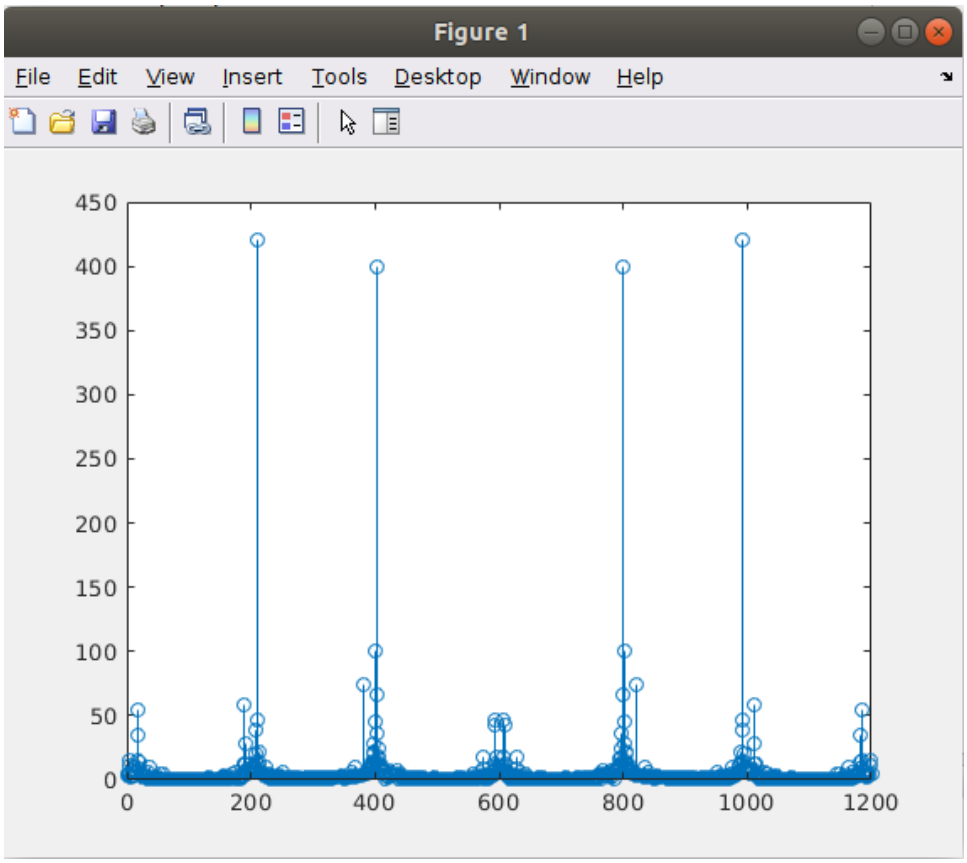
The problem asks to design the classic DTMF (Dual Tone Multifrequency) decoder.



The signal of the tone (to be decoded) plotted in time domain

We slice the signal into 8 parts (since there are 8 numbers). This is done by running a loop and slicing the moment we enter the white region after each blue region.

Then we compute the FFT of each slice separately, read the frequencies with the maximum and second maximum magnitude, normalize the frequency values and then try and map it with the frequencies of the standard DTMF frequencies corresponding to each number(given) to detect the number pressed.

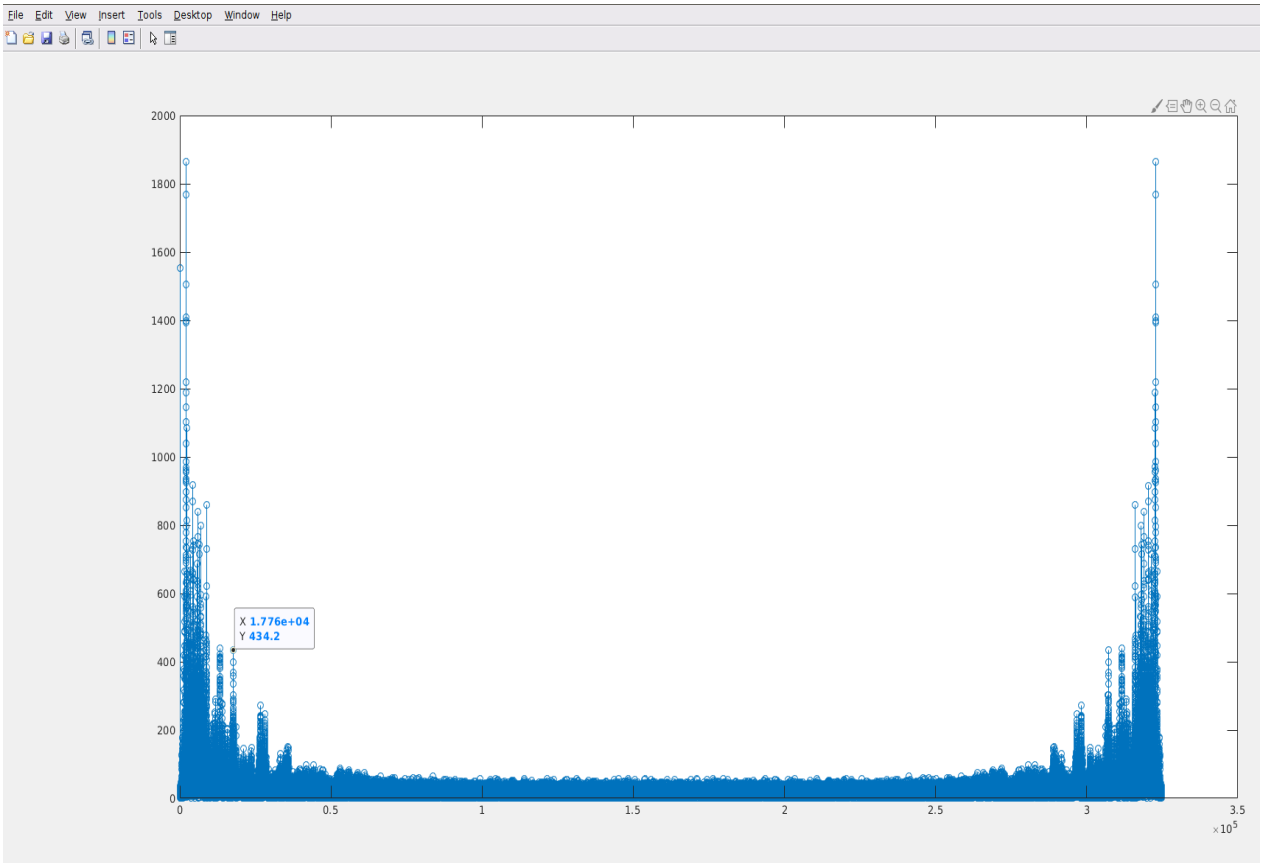


Plotting the magnitude vs frequency graph after running FFT on the last slice

The number corresponding to the tone is : **20161103**

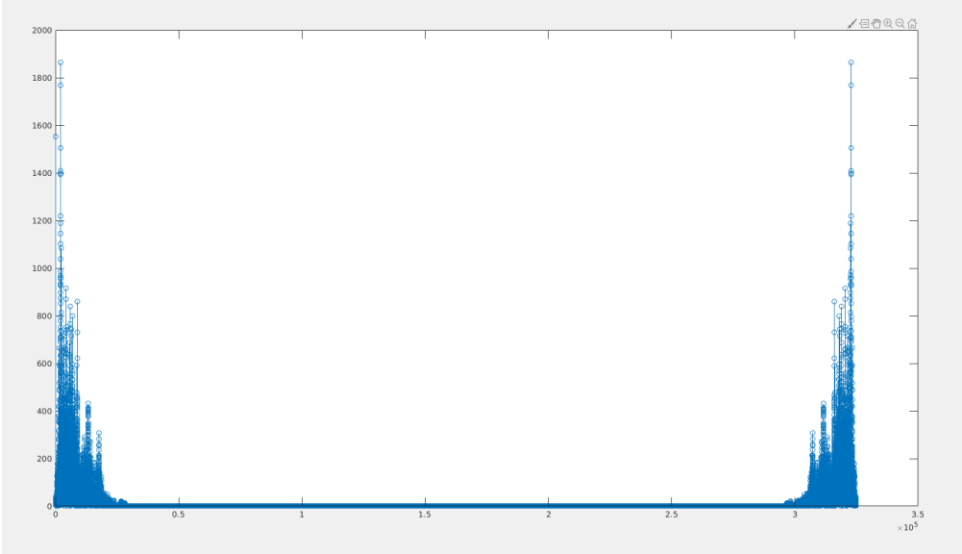
Question 4

Denoising signal 5.



The FFT plot (Magnitude vs Frequency) clearly shows that after frequency of around 1205 Hz which corresponds to the reading of 1.776e+04 in the given plot, there are no more prominent peaks giving us a hint that those frequencies are for the noise. So, we use a low pass filter to allow all the frequencies less than or equal to 1205Hz to pass through while block those greater than 1205Hz.

Such a filtering gives pretty decent results.

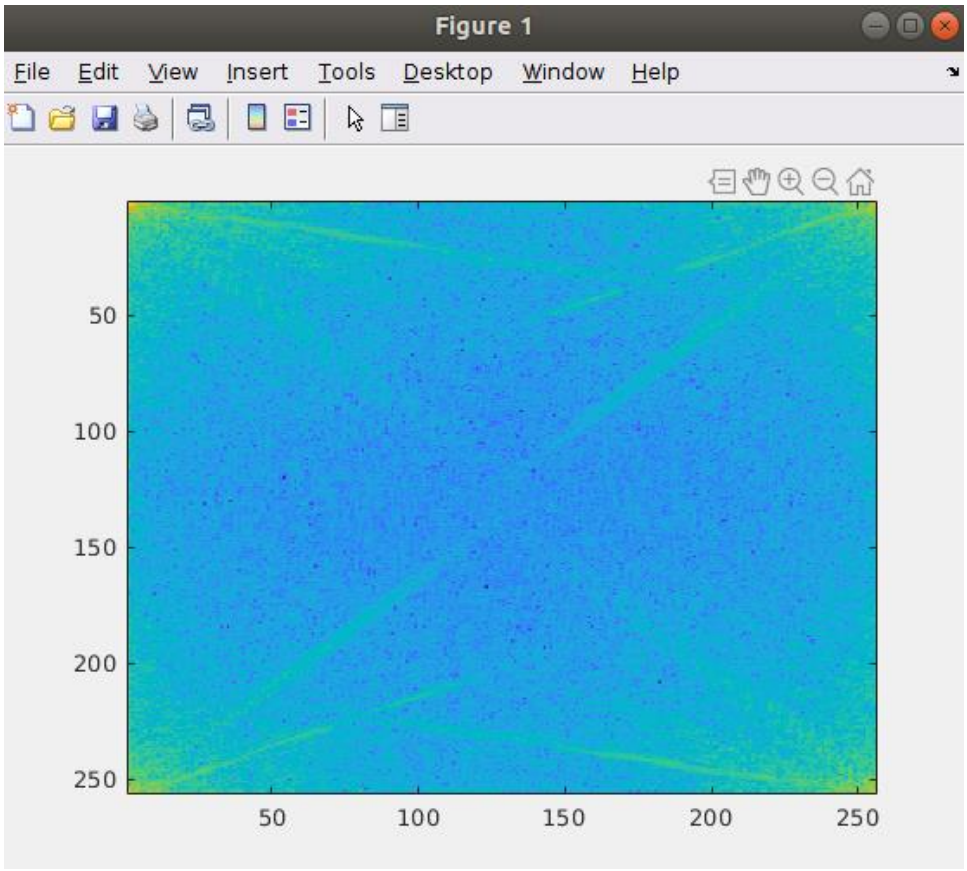


The FFT plot (Magnitude vs Frequency) after applying the low pass filter with cutoff frequency of around 1205Hz

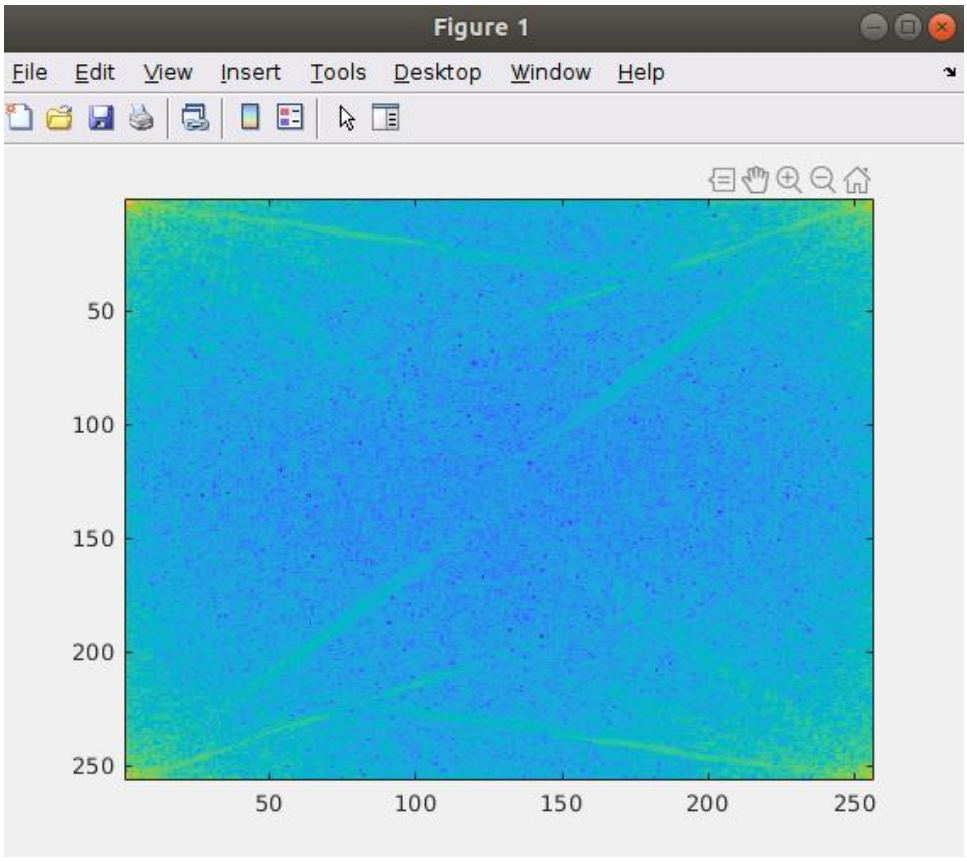
The plot clearly shows that after a threshold frequency the magnitude is made 0 indicating the blocking of the high frequencies by the low pass filters.

Question 5

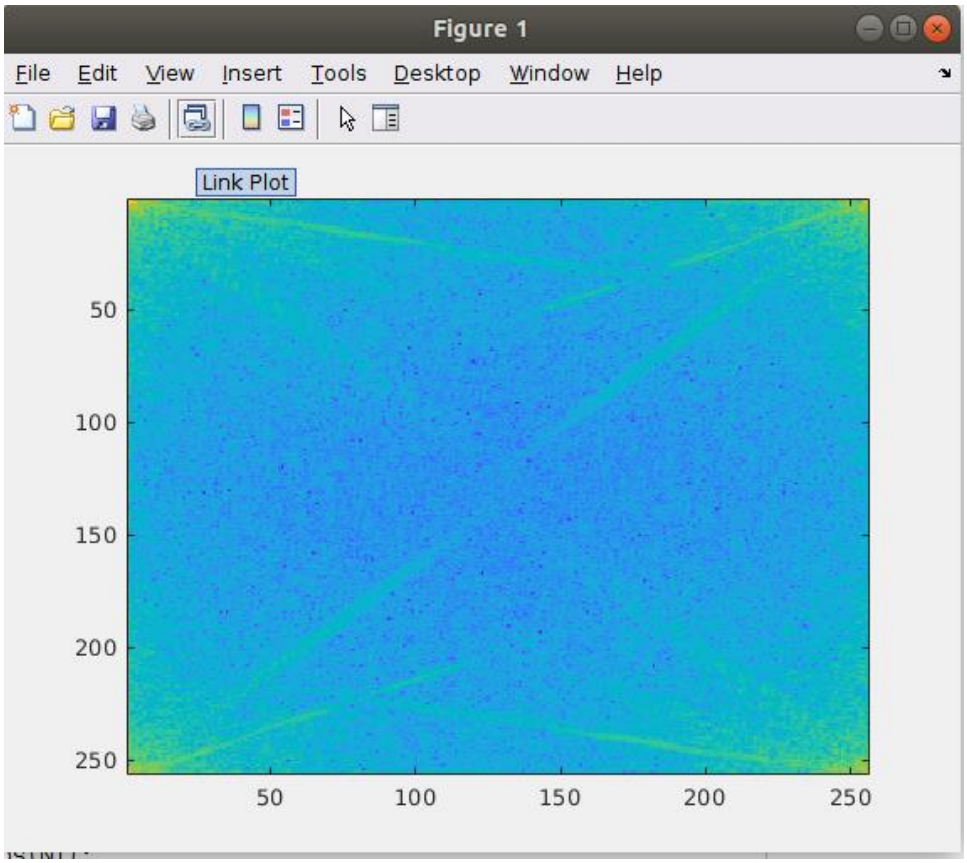
The famous picture of cameraman used in Image Processing



The output of running my own written 2d-DFT code on the cameraman image

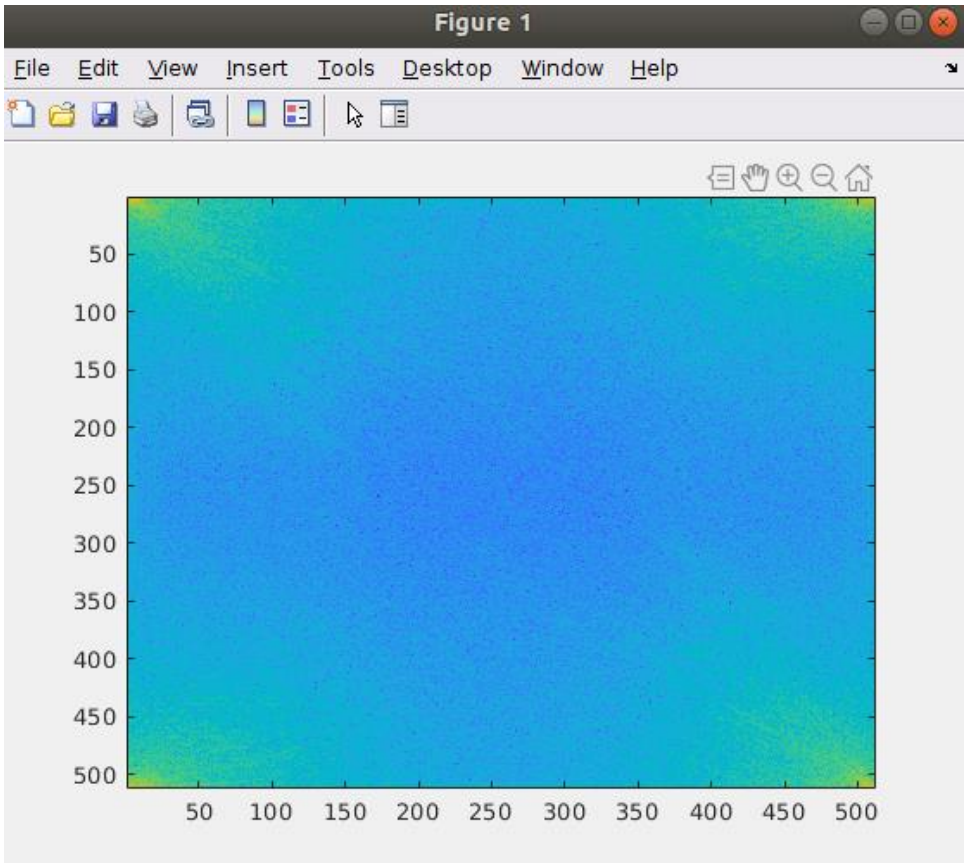


The output of running my own 2d-FFT code on the cameraman image

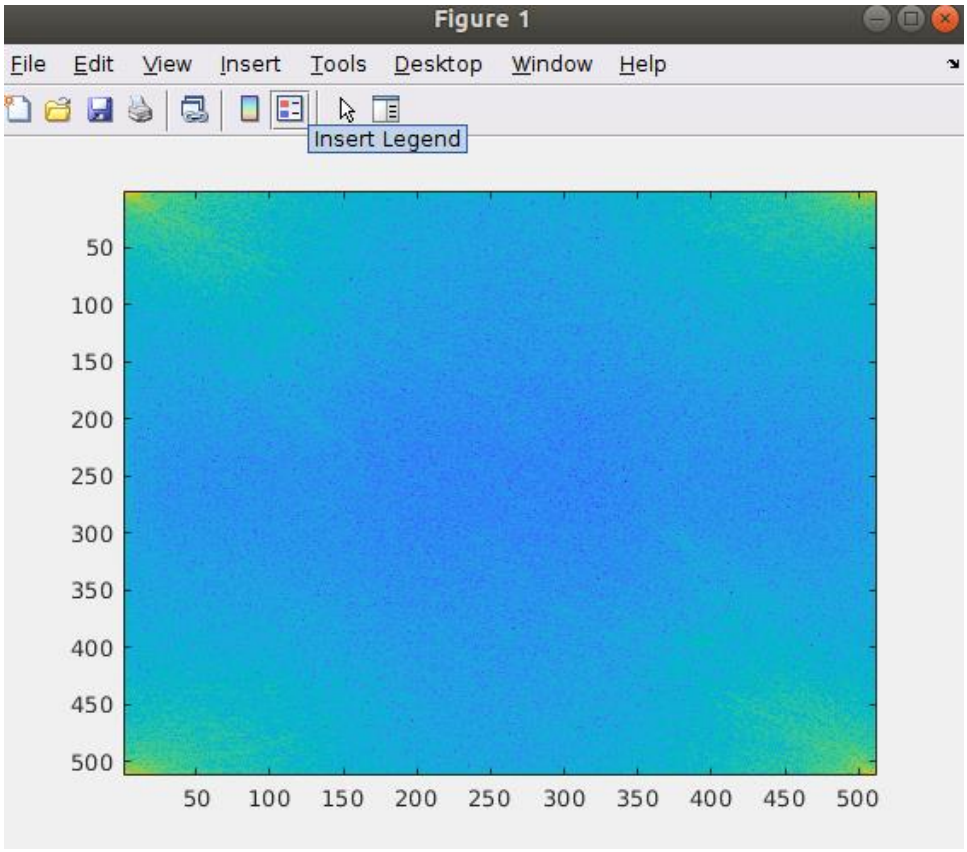


The output of running inbuilt 2d-FFT code on the cameraman image

The famous picture of Lenna used in image processing

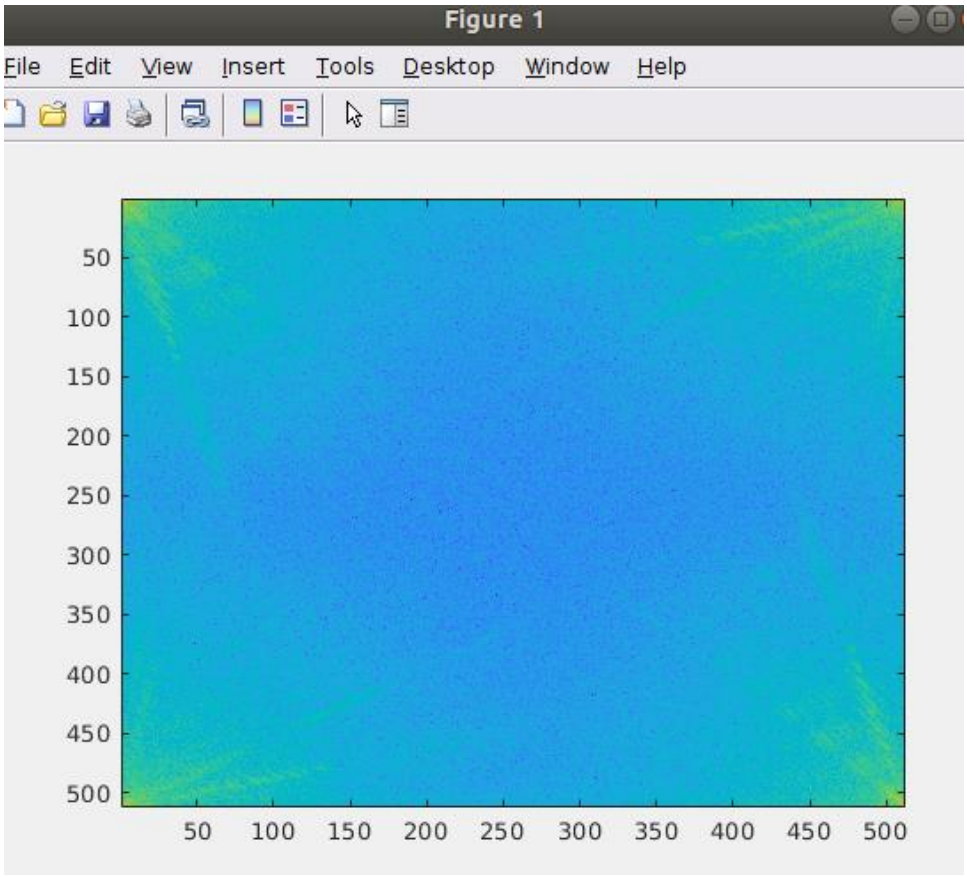


The output of running inbuilt 2d-FFT code on the Lenna image

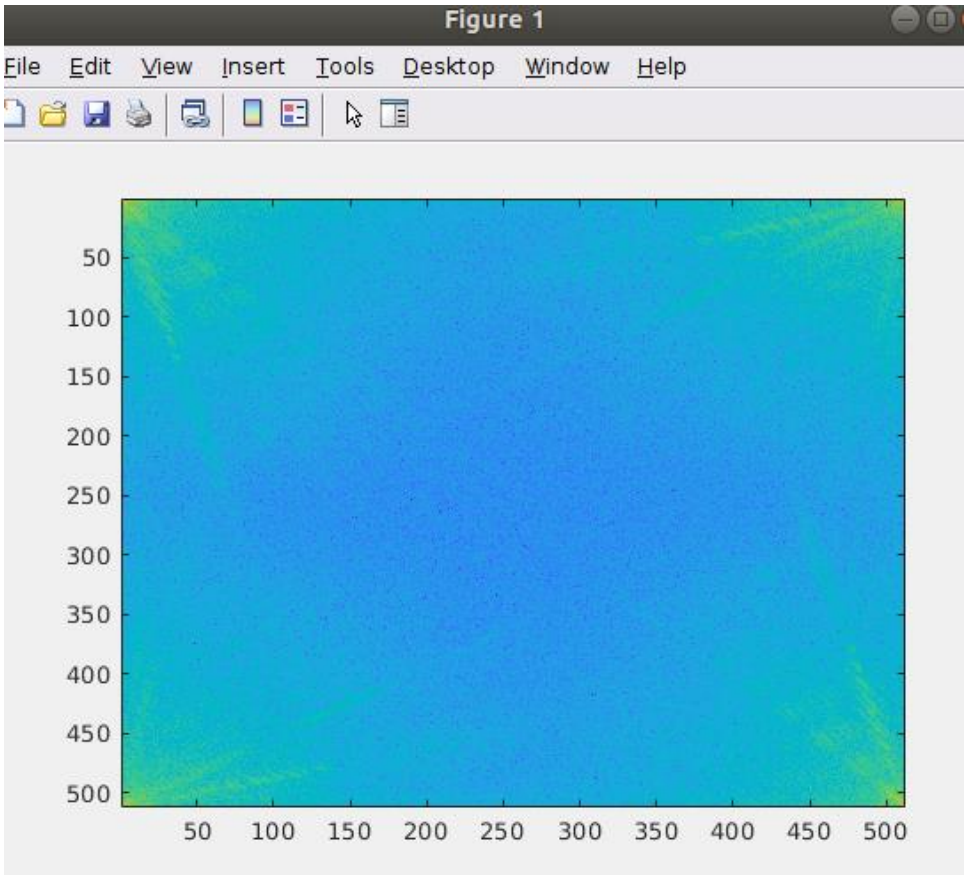


The output of running my own 2d-FFT code on the Lenna image

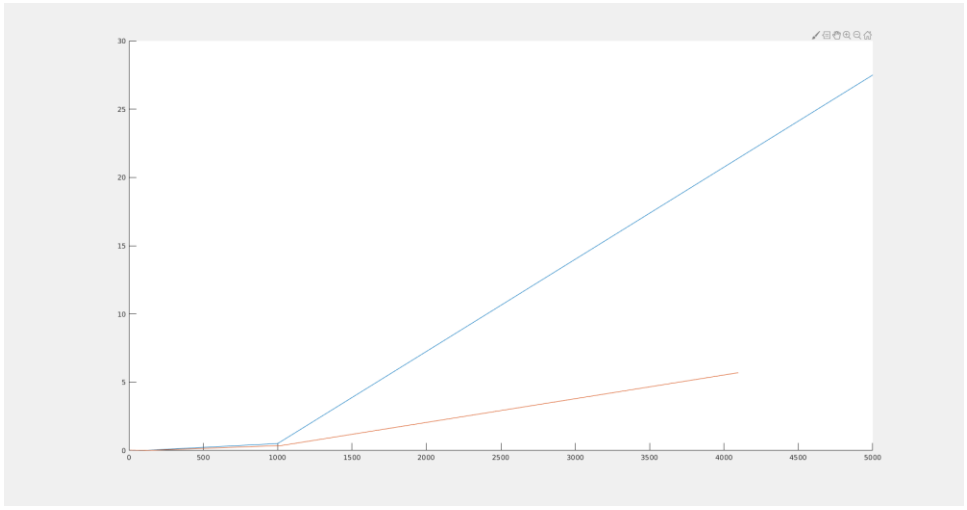
Grayscale image of fruits



The output of running my own 2d-FFT code on the fruits image



The output of running the inbuilt 2d-FFT code on the fruits image



Time vs input size plot for FFT(in red) and DFT(in blue)

The complexity of DFT is O (N * N) whereas the complexity of FFT is O (N * log N). Therefore, we can infer that for DFT the time complexity is N / log(N) more than FFT. For large values of N we can see how well the FFT performs compared to DFT. On a 5000 X 5000 matrix, FFT took around 5 seconds whereas DFT took around 29 seconds as shown in the above plot.

The two dimensional FFT we implemented works for N1 X N2 matrices where N1,N2 is a power of 2. If N1 is not a power of 2, we pad the columns of zeros until N1(=number of columns) becomes a power of 2. Similarly if N2 is not a power of 2, we pad rows of zeros at the end until N2 becomes a power of 2.

Question 6

Original image

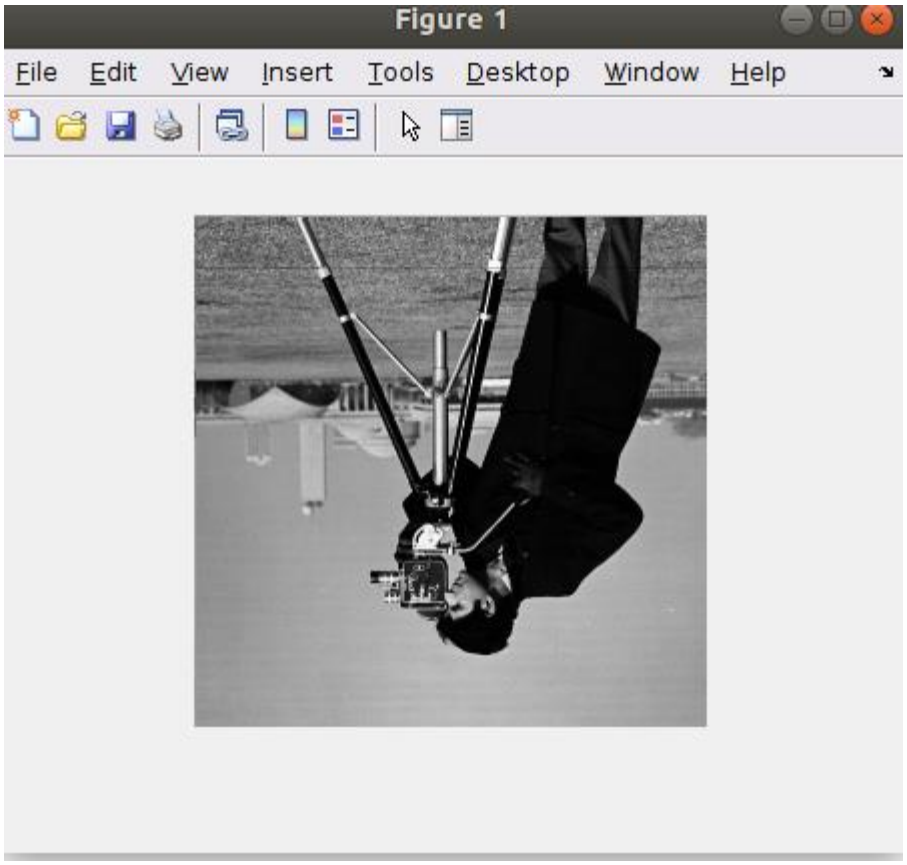


Image after applying Fourier Transform on the Fourier Transform of the original image

The image is an inverted version of the original image.

Let the length of the image be N2 and the breadth of the image be N1. In other words the image has N2 rows of pixels and N1 columns of pixels. The dimension of the image is N2 X N1.

I denotes the image matrix of dimension N2 X N1.

The Fourier Transform is given by $F = W_{N2} * I * W_{N1}$

If we again take the Fourier Transform, we get $F' = (W_{N2} * W_{N2}) * I * (W_{N1} * W_{N1})$

As an example , let us take W_4

```
>> dftmtx(4)

ans =

   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 - 1.0000i  -1.0000 + 0.0000i   0.0000 + 1.0000i
   1.0000 + 0.0000i  -1.0000 + 0.0000i   1.0000 + 0.0000i  -1.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 + 1.0000i  -1.0000 + 0.0000i   0.0000 - 1.0000i
```

If we take $W_4 * W_4$

```
>> ans * ans

ans =

     4     0     0     0
     0     0     0     4
     0     0     4     0
     0     4     0     0
```

We can clearly see the inverted 3 X 3 Identity matrix from row 2-4 and column 2-4. Therefore, we infer that apart from the first row of pixels and the first column of pixels, the rows are inverted first when the image is multiplied by (W_{N2}*W_{N2}) followed by the columns being inverted when multiplied by (W_{N1} * W_{N1}). Therefore, the image appears to be inverted.

Another example to strengthen our belief about this observation of $(W_N)^2$

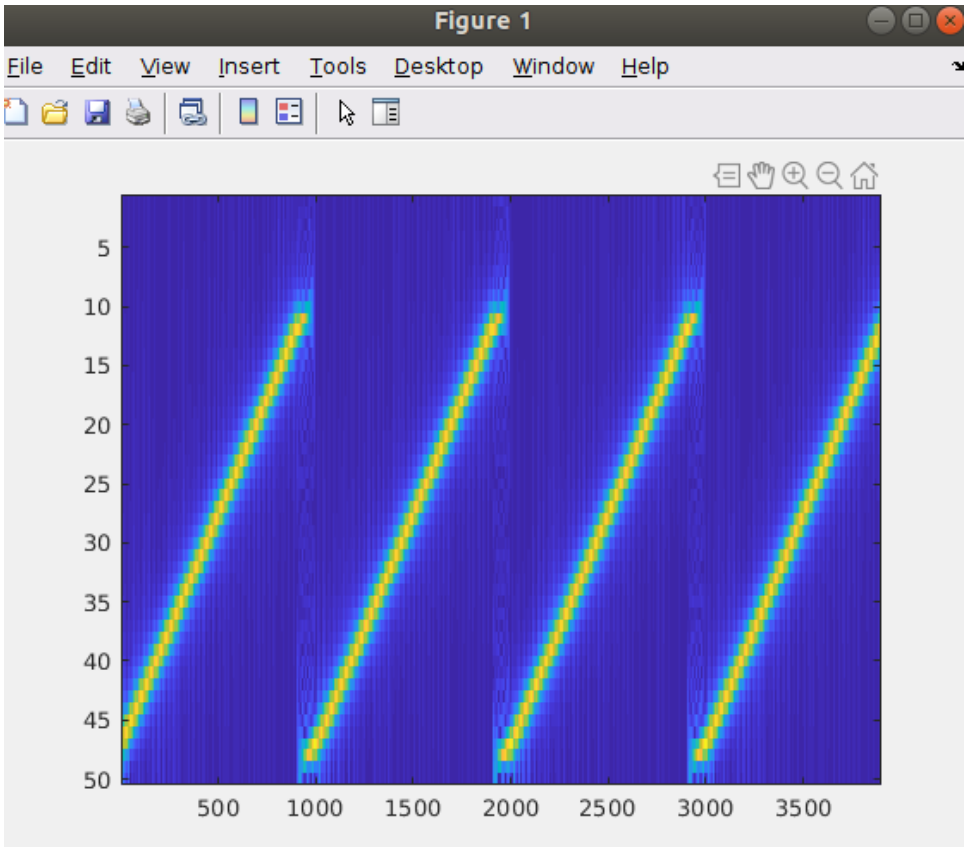
```
ans =

   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
   1.0000 + 0.0000i   0.5000 - 0.8660i  -0.5000 - 0.8660i  -1.0000 + 0.0000i  -0.5000 + 0.8660i   0.5000 + 0.8660i
   1.0000 + 0.0000i  -0.5000 - 0.8660i  -0.5000 + 0.8660i   1.0000 + 0.0000i  -0.5000 - 0.8660i  -0.5000 + 0.8660i
   1.0000 + 0.0000i  -1.0000 + 0.0000i   1.0000 + 0.0000i  -1.0000 + 0.0000i   1.0000 + 0.0000i  -1.0000 + 0.0000i
   1.0000 + 0.0000i  -0.5000 + 0.8660i  -0.5000 - 0.8660i   1.0000 + 0.0000i  -0.5000 + 0.8660i  -0.5000 - 0.8660i
   1.0000 + 0.0000i   0.5000 + 0.8660i  -0.5000 + 0.8660i  -1.0000 + 0.0000i  -0.5000 - 0.8660i   0.5000 - 0.8660i

>> ans * ans

ans =

    6.0000         0         0         0         0         0
         0    0.0000   -0.0000         0    0.0000    6.0000
         0   -0.0000    0.0000         0    6.0000    0.0000
         0         0         0    6.0000         0         0
         0    0.0000    6.0000         0    0.0000   -0.0000
         0    6.0000    0.0000         0   -0.0000    0.0000
```

This can be fixed in the frequency domain by taking W_N mirrored about diagonal instead of W_N and then carrying out the multiplication when applying the FFT for the first time and then the second FFT is performed as usual.

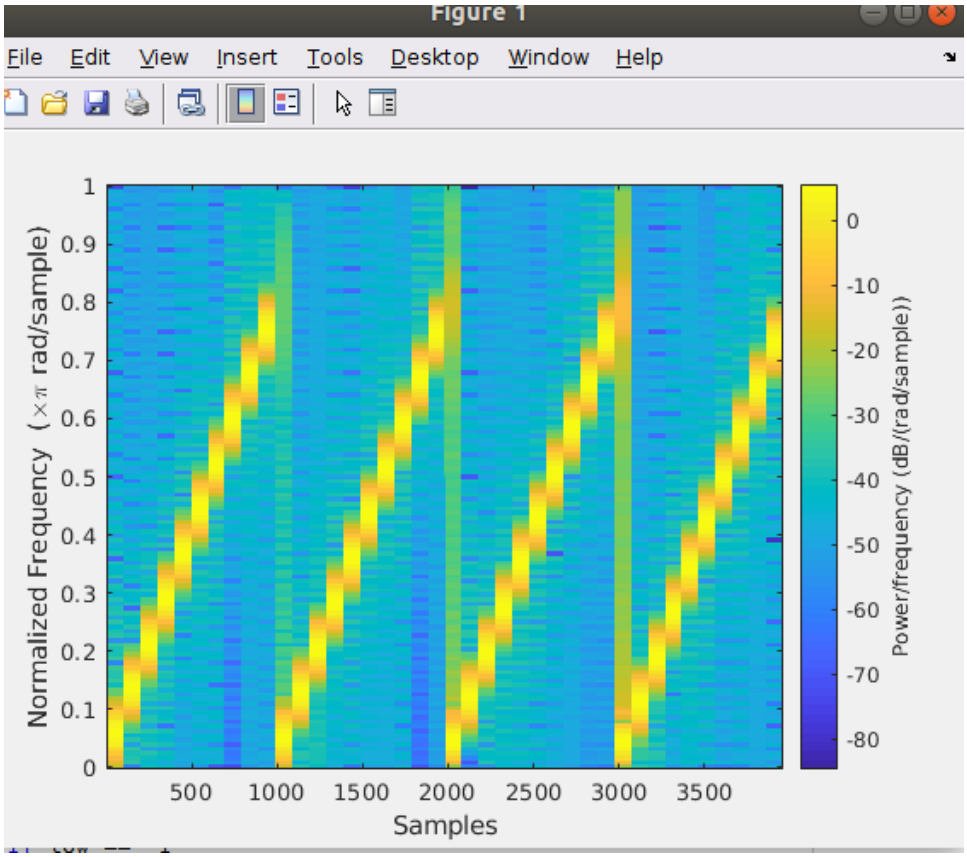
When we take W_N mirrored about diagonal we are actually doing IDFT. In other words, the first time we do an IDFT and then a DFT and we get back the original image.

Question 7

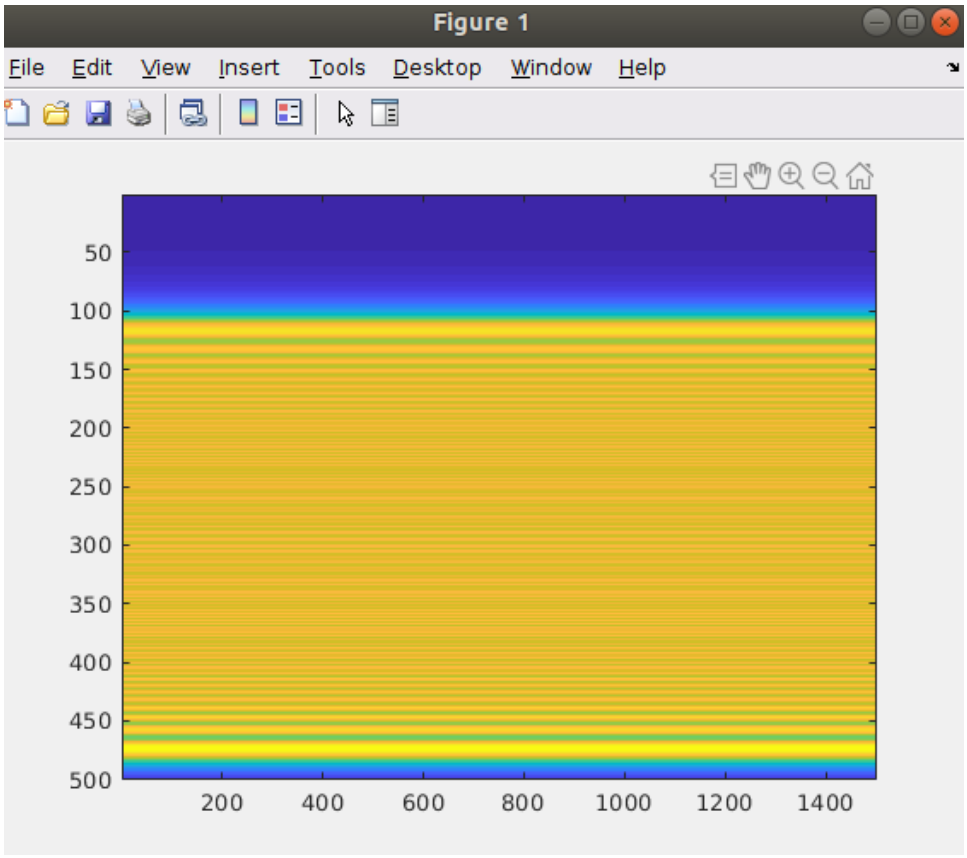
Part 1



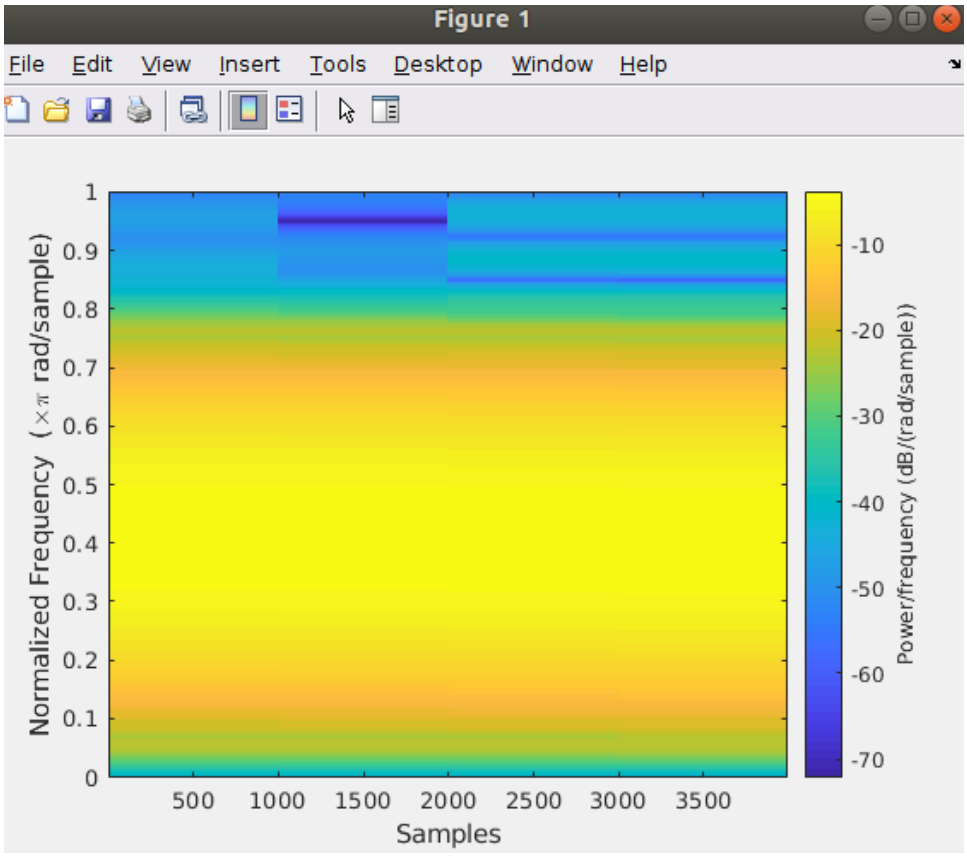
Spectrogram generated by my code having window size 100 and step size 1



Spectrogram generated by inbuilt spectrogram() having window size 100 and step size 1



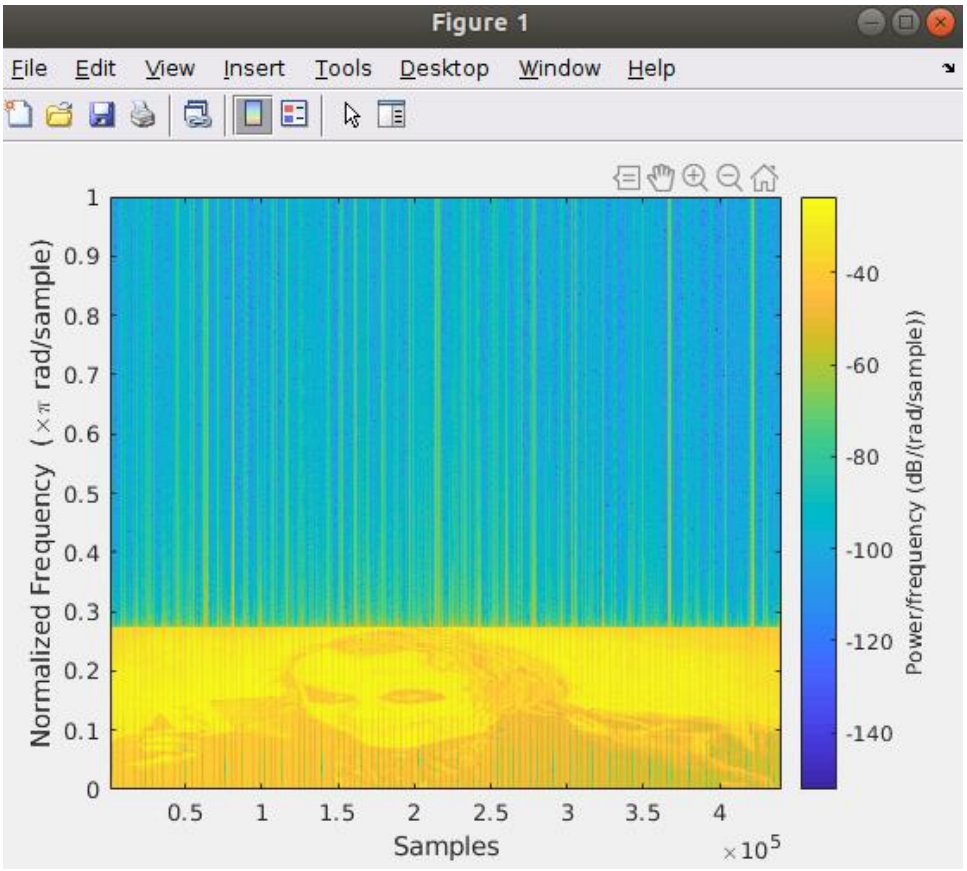
Spectrogram generated by my code having window size 1000 and step size 2



Spectrogram generated by inbuilt spectrogram() having window size 1000 and step size 2

The spectrograms are similar in shape but are not normalized. Also, the values along the y axis is inverted due to the property of inbuilt imagesc() used for displaying.

Part 2

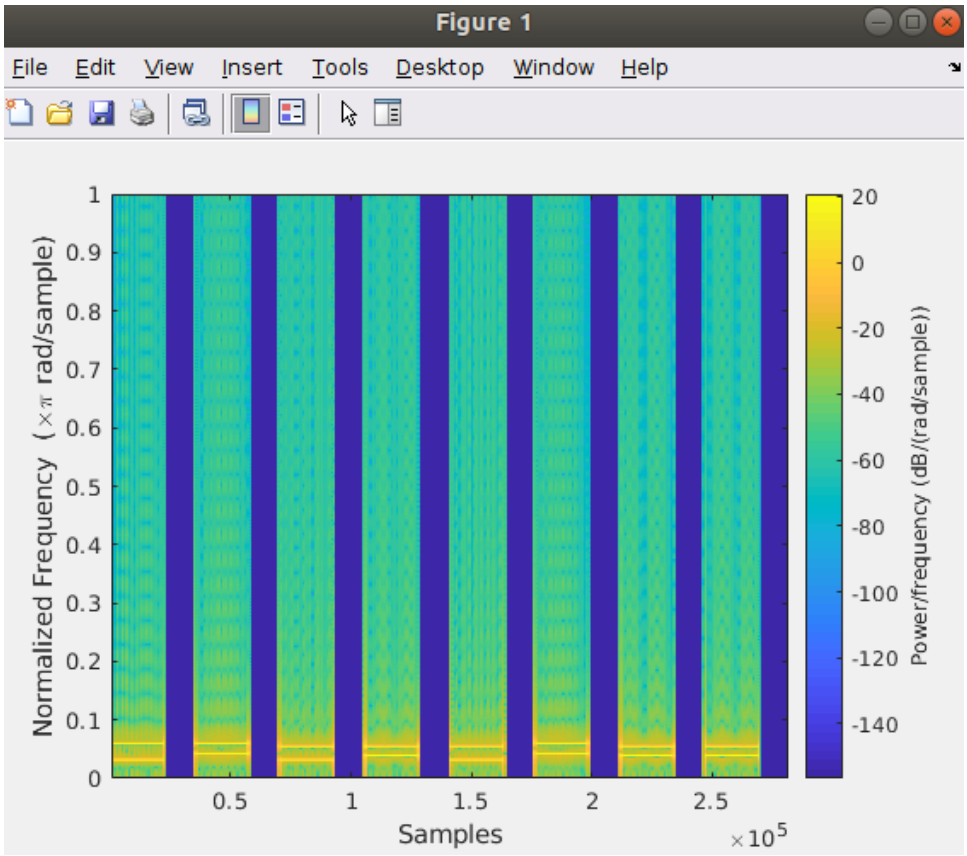


Spectrogram of the sound file message.wav with window size of 2000.

This reveals us that the password is: Joker

(From the movie Batman!)

Part 3



Spectrogram of my roll number (20171077) according to DTMF standard frequencies using window size = 2000