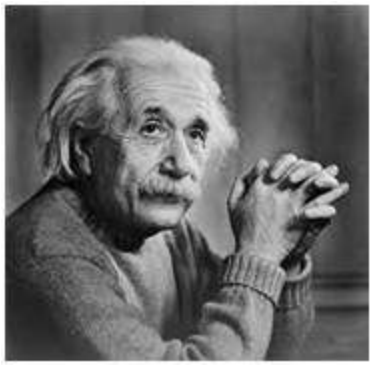
# DSAA Assignment 2

## Question 1:

1. *(q1a.m)* Created a gaussian filter function that takes the following two arguments as input:
    - N : Kernel size
    - sigma : Standard deviation sigma

    This function will return a N x N Gaussian filter with standard deviation sigma.

    Input image (182*186) :

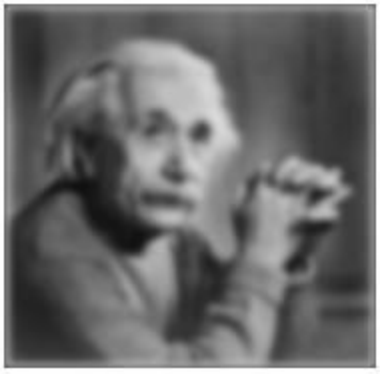


    Outputs with different filter values :

- N = 65 and sigma = 1 (left image is the function written by me and right image is using fspecial).

- N = 65 and sigma = 3

- N = 155 and sigma = 2



**2**. Median filtering is a nonlinear method used to remove noise from images.

- It is widely used as it is very effective at removing noise while preserving edges.
- It is particularly effective at removing ‘salt and pepper’ type noise.
- The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels.
- The pattern of neighbours is called the "window", which slides, pixel by pixel over the entire image 2 pixel, over the entire image.
- The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value.

While calculating values for the border elements and elements near the border, we don’t have enough elements to fit in the N*N window . There are different ways we can handle this problem:

- Avoid processing the near border elements.
- Pad the image matrix with zeros.
- Pad the image matrix by extending the border values.

The inbuilt function medfilt2() pads with zeros by default. One problem I noticed when padding with zeros is that the elements near the border attain the values of zero. Here are the outputs with the 2 different approaches:

Input image:

Output images:


**3**.

Original image:

Gaussian filter on cameraman.tif :

Gaussian filter,N=150,sigma=1

Gaussian filter,N=65,sigma=3

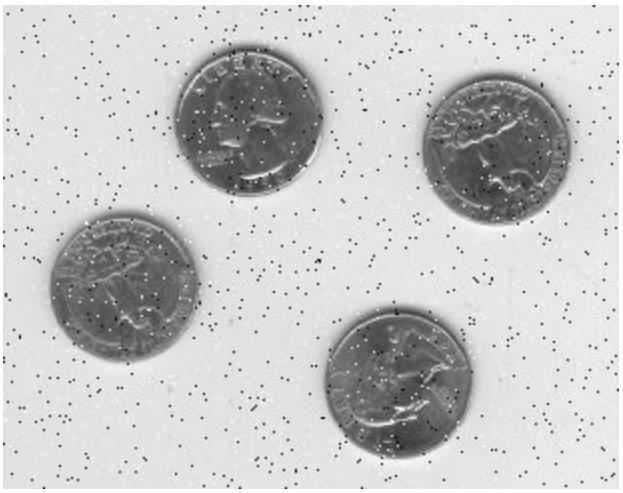
Gaussian filter,N=65,sigma=1

Median filter on cameraman.tif:



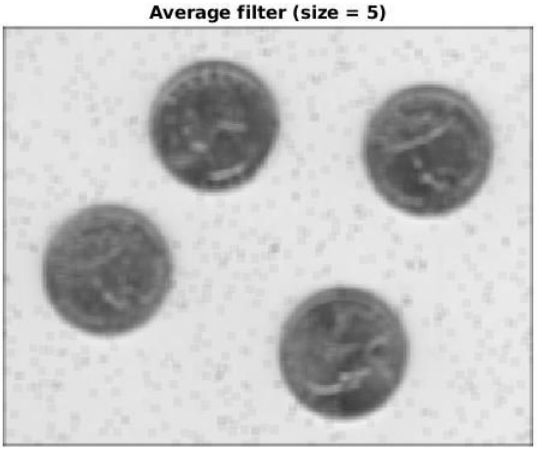

**4**.

Input image:



Gaussian filtered

Average filter (size = 5)



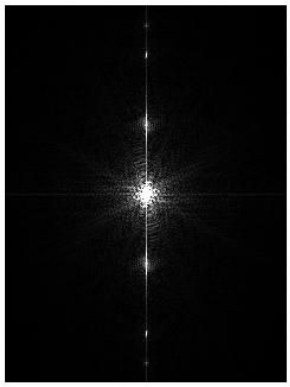
Median filtered

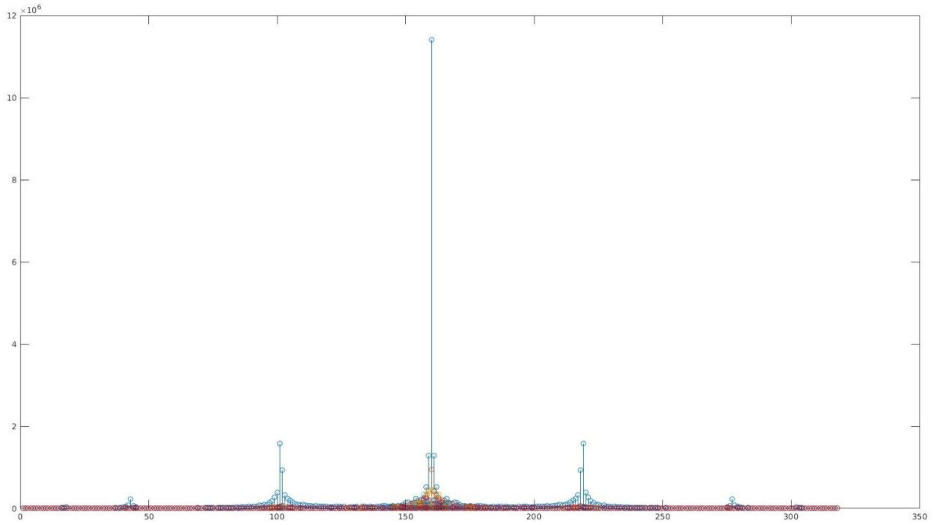
- The median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. It is particularly effective in removing ‘salt and pepper’ noise (bits have been flipped with probability 1%)
- Mean filtering is a simple, intuitive and easy to implement method of *smoothing* images, *i.e.* reducing the amount of intensity variation between one pixel and the next.
- Gaussian smoothing is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump.

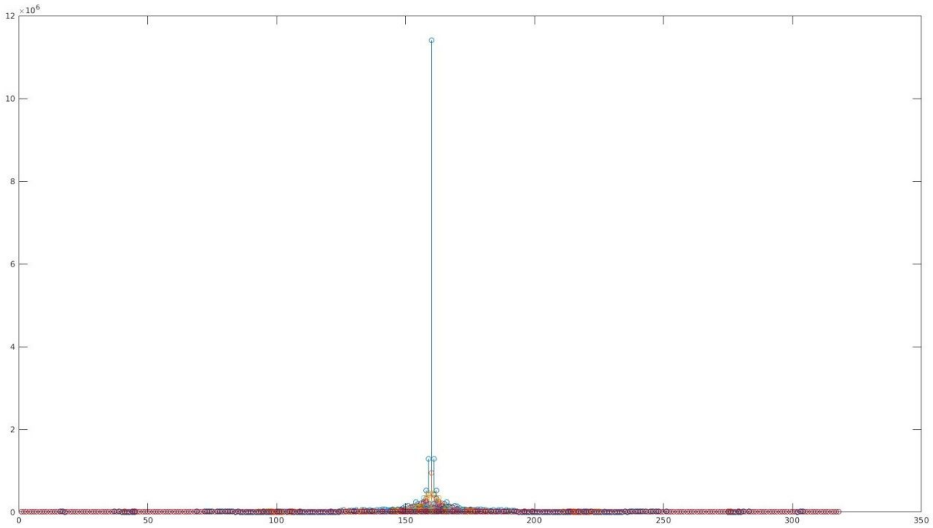
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

**5**.

- Observing the fft of the image( centred using fftshift and normalized) , we can see some frequency outliers in the central vertical line:





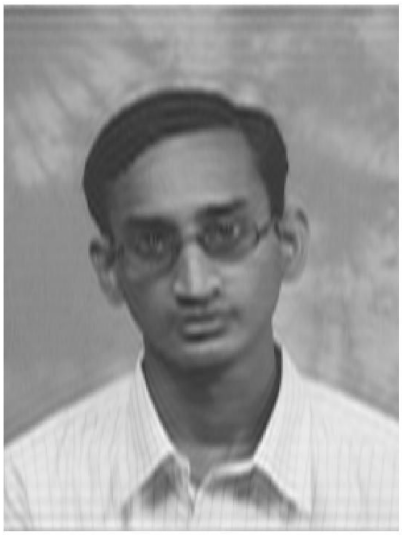
- Observing the stem plot of the fft of the image(centered using fftshift), we can see the are some spikes outside the central range.
- I tried to remove these outlier frequencies by :
    1. Setting a threshold value and removing the spikes if they exceed the threshold in the outlier range.
    2. Simply setting the values in the outlier range to zero.

Results obtained:

Original image-

Output images :

Setting to zero-



Setting to zero if threshold exceeded-

# Question 2:

**a.**

The dimension of the output of the convolution will be :
dimensions = ceil( (Width+2*Z -F+S) / S ) * ceil( ( Height + 2*Z - F + S) / S ) * C

**b.**

For obtaining a single value, F*F*C point-wise multiplications and F*F*C point-wise additions will be Performed. Hence :

Total additions =
(F * F * C ) * dimensions

Total multiplications =
(F * F * C ) * dimensions

# Question 3:

- Used fft on each of the individual dial tones to determine constituent frequencies.
- Used stft on the whole signal to determine frequencies in each window.
- Window length = length of signal/ number of tones
- Each segment's frequencies were compared to individual dial tones and ouptut was returned as shown:

Output found : 2 0 1 6 1 1 0 3

# Question 4:

- Used a bandpass filter to get frequencies between 200 and 4000 Hz after observing the spectrogram.
- Applied moving mean and gaussian filters after that on the signal.
- Used channel 1 as it seemed to have lesser noise.
- Smoothdata, lowess and window size = 20 gave a comparably good result

# Question 5:

DFT and FFT have been implemented in q5dft.m and q5fft.m.
My implementation of fft was found to be around 80 times slower when run on a 4*4 matrix compared to fft2().

# Question 6:

Consider this (4*4) matrix :

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$(1/4)*(1/4)*\text{fft}(\text{fft}(X)) = \begin{bmatrix} 1 & 4 & 3 & 2 \\ 13 & 16 & 15 & 14 \\ 9 & 12 & 11 & 10 \\ 5 & 8 & 7 & 6 \end{bmatrix}$$

All the rows and columns get rotated/flipped except the first row and column.
This can be understood by taking the simple case of 1D dft:
Let X = [ a b c d ]
fft(fft(X)) = (¼)*W4*((¼)*W4*X')

$$= (1/4) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} * X = [\, a\ d\ c\ b \,]$$

Basically, all the elements except the first got rotated/flipped about the middle; this can be extrapolated to 2 dimensional dft.

original image

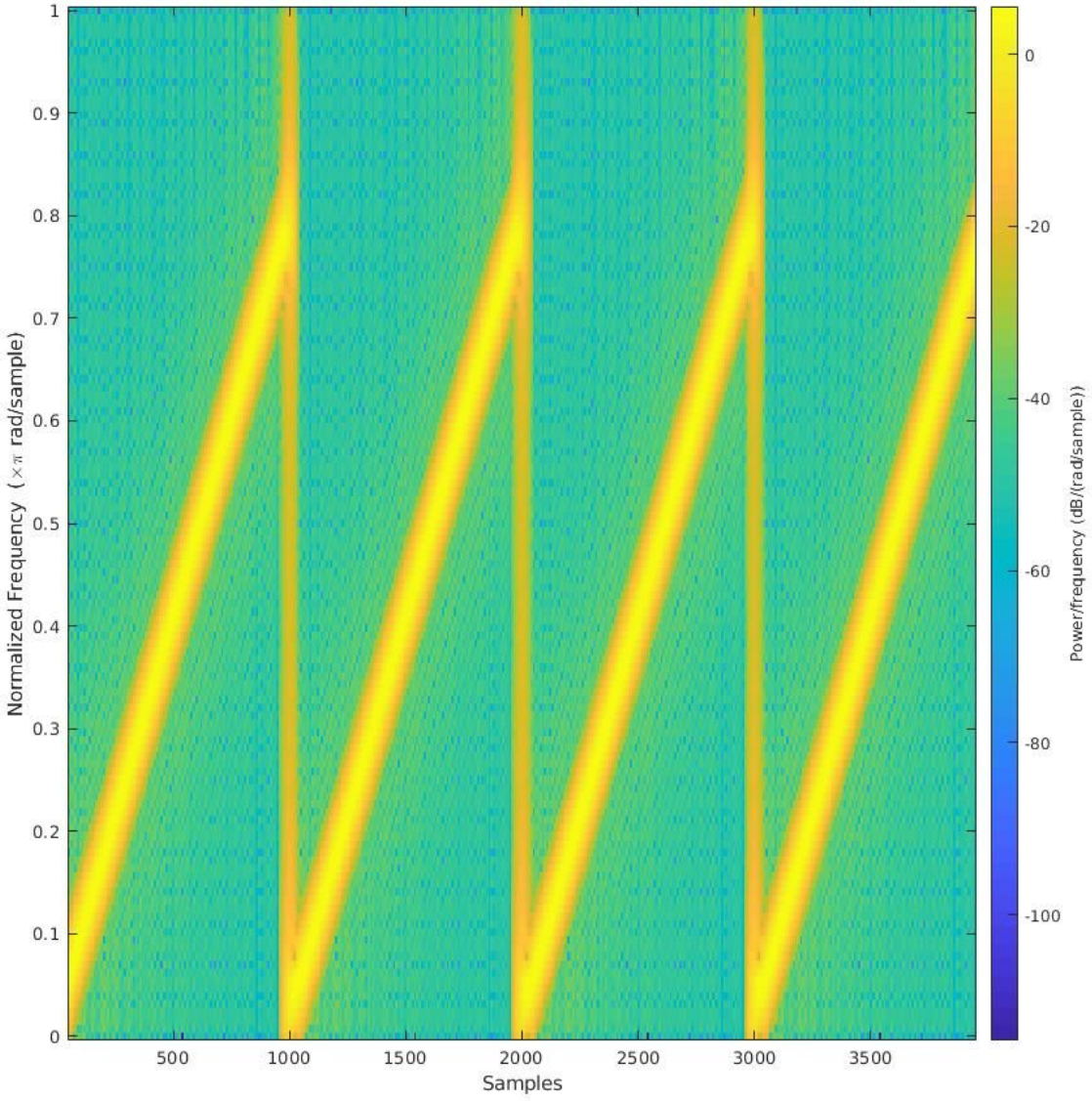
double-fft

double-conj-fft

- As is clearly seen, in the double-fft image, running fft on the fft of the same image flips it around the the rows and columns one by one.
- In order to get the same image back , we can take the conjugate of the matrix obtained by applying 1 fft ( conjugate of the image in the frequency domain); and then taking fft again on it, shown in the above image- double-conj-fft.

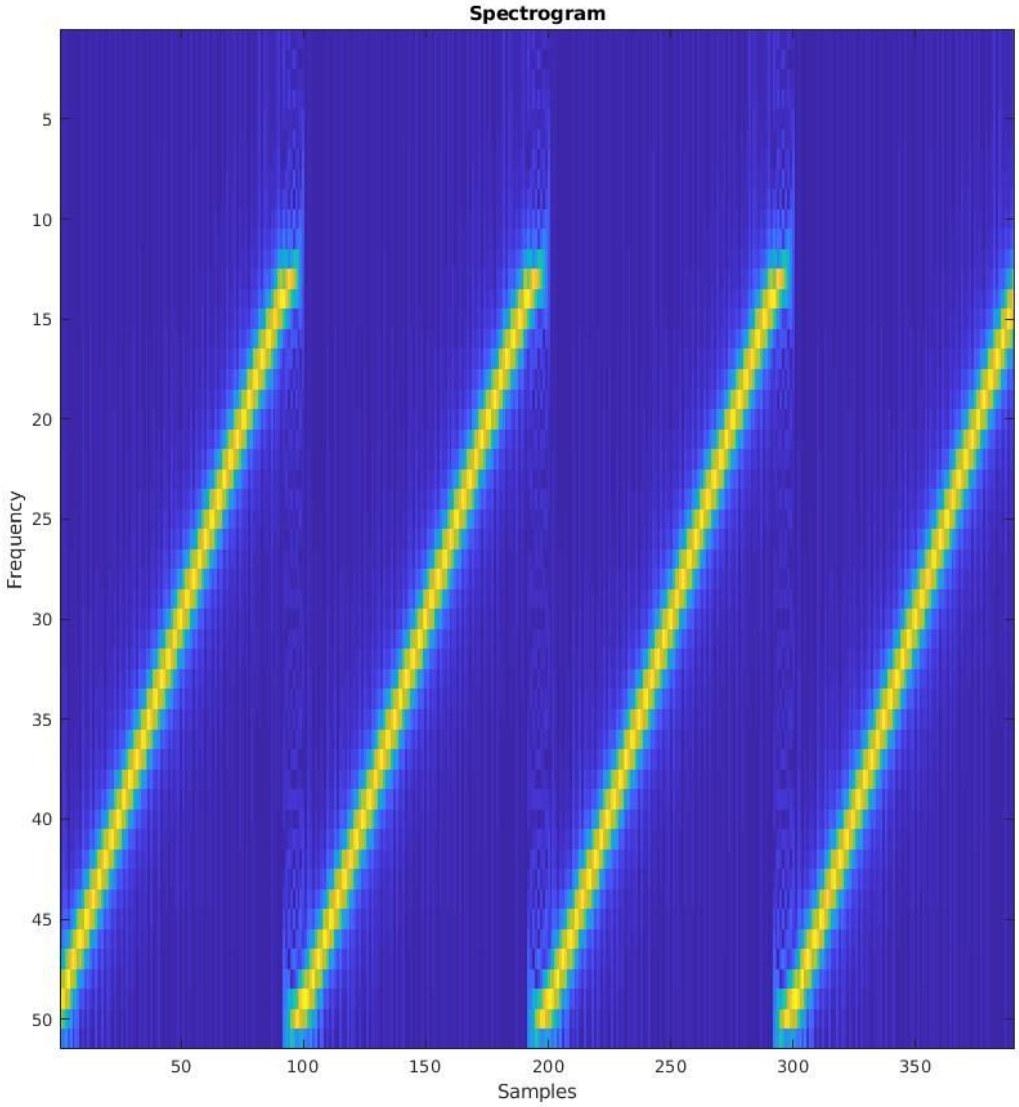
# Question 7:

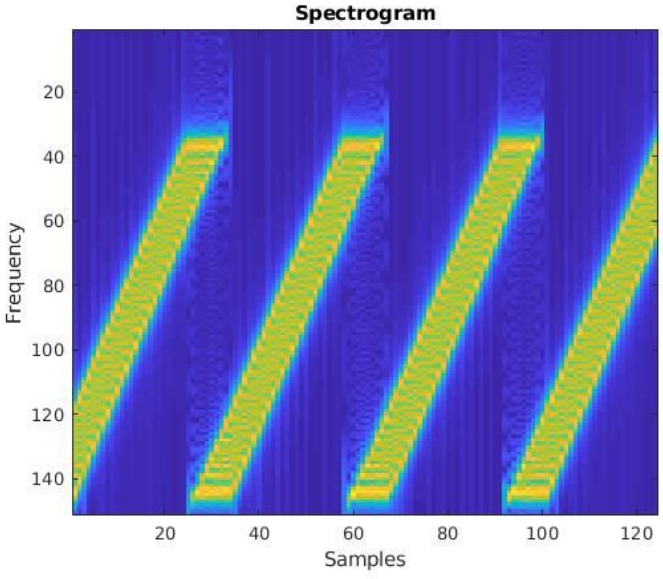
**1**. Spectrogram output from inbuilt function ( window size =100 and overlap =10):

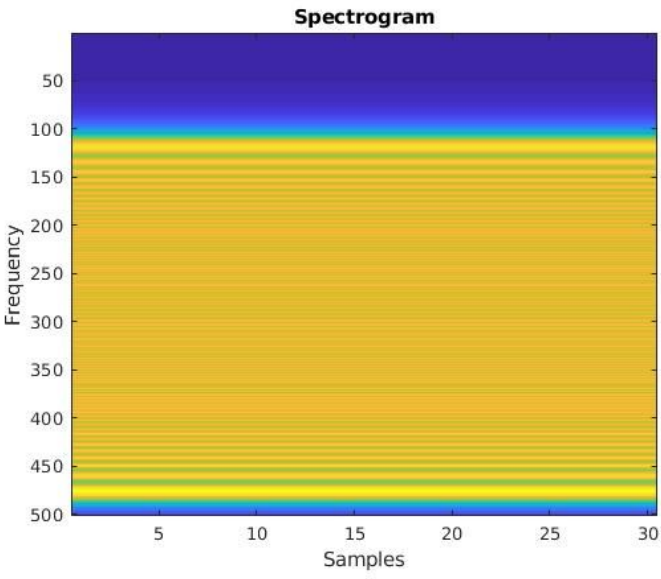



Outputs from my function( in order : window size and stride length ) :

1. 100 , 10
2. 300, 30
3. 1000, 100


Spectrogram
Frequency
Samples
5
10
15
20
25
30
35
40
45
50
50
100
150
200
250
300
350


Spectrogram
Frequency
Samples
20
40
60
80
100
120
140
20
40
60
80
100
120


Spectrogram
Frequency
Samples
50
100
150
200
250
300
350
400
450
500
5
10
15
20
25
30

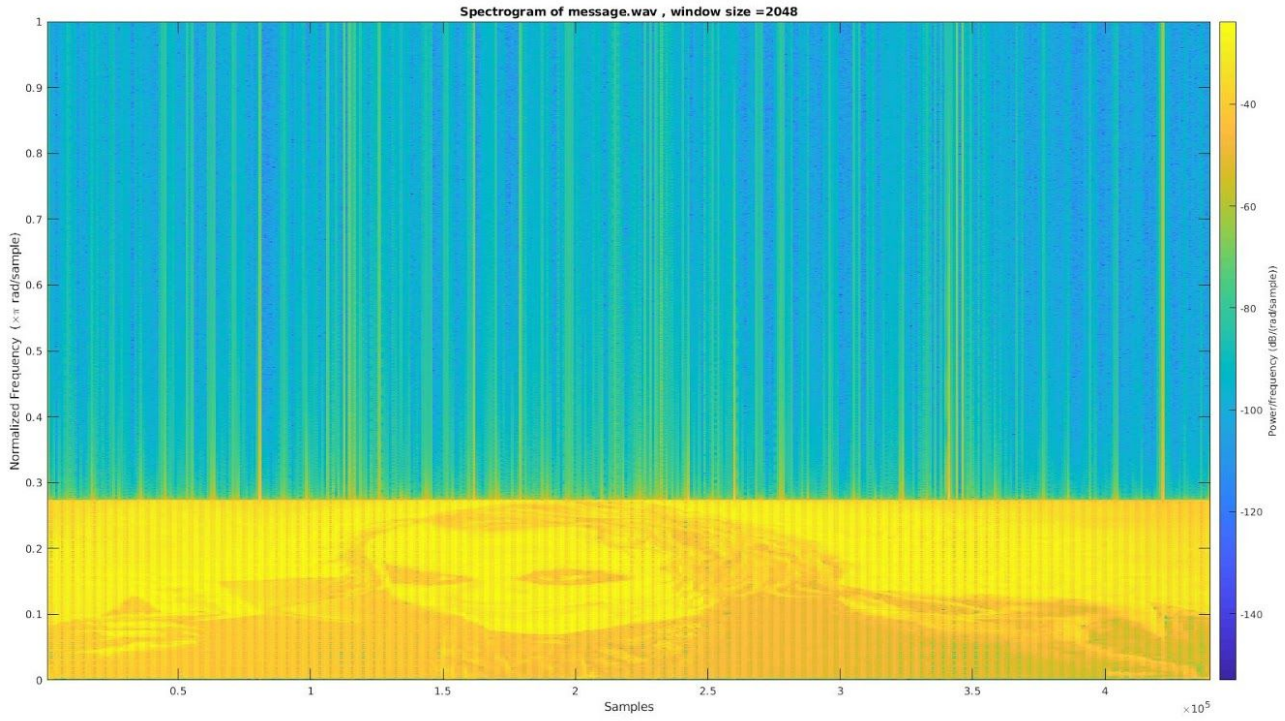
Observations :

- Long window size -> More dft points/samples -> more frequency resolution
  Long window size -> less precision in time, transitions can be missed

- Small window size -> More time slices -> precise locations of transitions
  Small window size -> less dft points, poor frequency resolution.

**2**.

On taking Short time fourier transform and observing the spectrogram(window size =2048), we see the joker's image.



**3.** *(q7c.m)* The user is asked to enter roll number. Corresponding DTMF dial tone is created and spectrogram is returned. This is one such example of my roll number(20171062) :