Middle East Technical University

Electrical-Electronics Engineering Department

EE430 Digital Signal Processing Term Project (2017-2018 Fall Semester)

Group 9

Mehmet Mert Beşe 2093482

Berk İskender 2031920

Project Phase 1 – Implementation of a program that computes and displays the Short-Time Fourier Transform (STFT) of a time domain signal

In the first part of the project, we were required to implement a function that computes and displays the spectrogram of an input signal. Also, we were required to implement data acquisition and data generation modules which acquire or provide the required signal to our spectrogram implementation. This steps are implemented with a user interface that completes steps according to the user inputs.

Data Acquisition: Two types of acquisition exist. The first type is the voice recording. In order to record voice, user should select the option '0' and sampling rate of the recorded signal. Finally, program asks the user to select the length of the signal as input and recording starts and finishes. Total sample amount of the acquired signal is equal to recording period times sampling rate.

Second type is sound input from an audio file. In order to take input from a sound file, user has to select option '1' and provide the file name as in the format of 'sound_file.mp3' or 'sound_file.wav'. 'audioread' function of MATLAB takes this string as input and provides the sound signal and sampling rate of the signal.

Data Generation: As a third choice, user can generate three types of signals and their combinations by selecting the option '3'. These types are continuous sinusoidal, windowed sinusoidal and linear chirp sinusoidal signals. User has to select options 0, 1, 2, 3 for respective types.

For sinusoidal, length of the signal, amplitude, frequency and phase should be provided as inputs.

For windowed sinusoidal, length of the signal, amplitude, frequency, phase and starting time should be provided as inputs.

For linear chirp sinusoidal, length of the signal, amplitude, frequency, phase, starting time and bandwidth (which determines the sweeping range of the chirp sinusoidal in the period of generation) should be provided as inputs.

For multiple signal generation, steps listed above should be followed for each signal after providing the total amount of signals included in the audio signal.

How does spectrogram work?

After we generate or acquire our signals , these signals are windowed by 8 window types such as Hamming ,Tukey, rectangular, Hann, Triangular, Gaussian and Blackman. Then, FFT's of resulting windows are calculated for each time block. After that, signal is represented in a graph where x -axis determines the time, y-axis determines the frequency and colors determine the intensity of the frequency component in logarithmic scale. Input audio signal is plotted on the spectrogram as windowed blocks for the range of frequencies which is determined by the sampling rate of the signal. While plotting the spectrogram, user has to select the length of the window in terms of seconds and amount of overlap for each window, again in seconds. Comments about the selection of these parameters are included in the following section. Also, in order to avoid displaying the negative frequency components of the real signals on the spectrogram (since it computes DFT, they show up at the end of the frequency axis in a symmetrical manner) we truncated the complete spectrogram after the radian value of pi.

Comments on spectrogram computation and its utilization:

Before skipping to the works related to phase 2 of the project, we want to include details about how to select overlap amount and the purpose of windowing while using spectrogram function that we have constructed.

- While obtaining the spectrogram of a signal, there is a tradeoff between spectral leakage and spectral resolution. Rectangular window has higher side lobes compared to other windowing methods. A window that results in less spectral leakage has smaller side lobes in its Fourier transform. However, it distorts the input signal unavoidably because of the increased main lobe width while decreasing the magnitude of the side lobes in its Fourier transform. Increased main lobe width limits the spectral resolution per window length since we use convolution in the frequency domain while windowing.
- To increase the resolution, window length can be increased. Being an almost optimal value, we tried to use %50 overlapping per window while displaying our results in the further parts of the project.
- As you increase the window size, you will have a fine frequency resolution, however, a coarse time resolution because wide windows have a long time duration but a narrow frequency bandwidth.

Project Phase 2

1) Reducing the Number of Time Samples:

In this part of the project, we are asked to reduce the total amount of time samples in a periodical manner (downsampling) with different ratios. (e.g. discard one of each N time domain samples) To perform this operation, we used built-in functions of MATLAB which are “downsample” and “decimate”.

Effects of increasing downsampling rate: As downsampling factor increases and exceeds the value of pi/W (W is the highest frequency present in the signal) aliasing starts to occur. As a result of this fact, starting from the higher frequencies, signal gets aliased (distorted). When we apply this effect on linear chirp with sufficiently high bandwidth (m value in the chirp) or any signal that includes higher frequencies, signal has doubled higher frequency components due to aliasing. This effect can be easily seen in linear chirp case. As we increase the ratio, high frequency parts starts to get repeated in the spectrogram which means that the power of those corresponding frequencies are doubled.

As the downsampling rate incerases, signal becomes shorter and its speed gets multiplied by the downsampled factor when played using the sampling frequency rate. However, if the playing frequency is fs*1/downsampling rate, speed remains constant while doing playback.

Moreover, same effects can be observed in the recorded sound signal and input data file when we apply downsampling and decimation separately. As we apply downsampling, frequencies present in those signals are stretched towards the higher values (proportional to the downsampling factor). When this factor times maximum frequency of the signal exceeds the half of the sampling frequency, aliasing starts to occur. Decimation nearly eliminates all of the aliasing effect in those two cases too.

Avoiding aliasing using decimation: In order to avoid this aliasing effect, decimation should be applied on the signal instead of downsampling. Decimation initially filters the signal (low pass filtering) with a pi/M cutoff frequency in order to avoid aliasing after stretching the frequency domain response due to downsampling. This technique may also result in loss of some high frequency components in the filtering stage while avoiding aliasing. However, losing such components is better than experiencing aliasing on them. Results obtained using decimation are added in order to display the behavior explained above.

Applying fractional downsampling factors: To use fractional downsampling factors, upsampling should be done first. For instance, if the signal is to be "downsampled" by a factor of 2.5, it should be upsampled by 2 firstly and downsampled by 5 afterwards. While doing so, we need to use a "low pass filter" after upsampling in order to eliminate frequencies higher than the W/L, where W is the bandlimit of the input signal. By doing so, there will not be repetitions at the output of the downsampler. This can be also achieved by applying decimation on the upsampled signal. By comparing both results of 2.5 factor downsampling, one can easily see that decimation provides a nearly ideal result. As an interpolator, zero order hold is used while performing upsampling.

Listening experience for produced signals (Audio comparison): For the sound file inputs, as we increase the downsampling rate, quality of the sound decreases. Also, aliasing starts to occur for smaller rates for signals that include higher frequencies. When decimation is used to avoid this effect, a muffled sound is heard with lower frequencies because it suppresses higher frequencies as explained and some frequencies are completely eliminated afterwards. Aliasing effect can be easily heard when listening to linear chirp cases.

Results using decimation in time-domain:

Results for 3 different signals are listed below:

a) Linear Chirp signal, starting from 0 to 10 kHz, 5 seconds long: While obtaining spectrograms, Hamming window is used. Overlap amount is 50% in the figures for each window.
   - Time Domain signals for different downsampling rates:

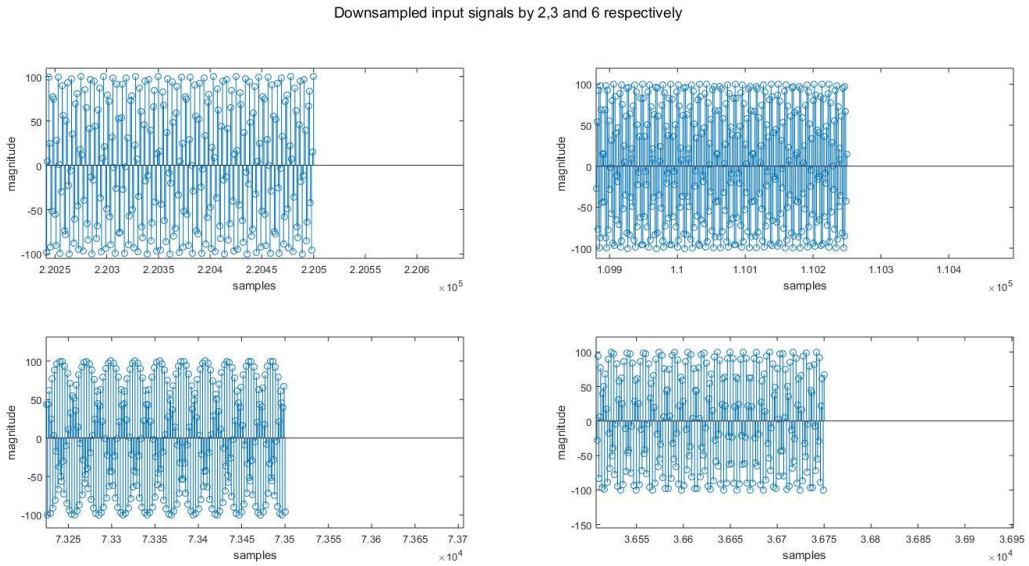

Figure 1: Fractions of time domain linear chirp signals (ends of the signals are extracted as fractions for better visualization)

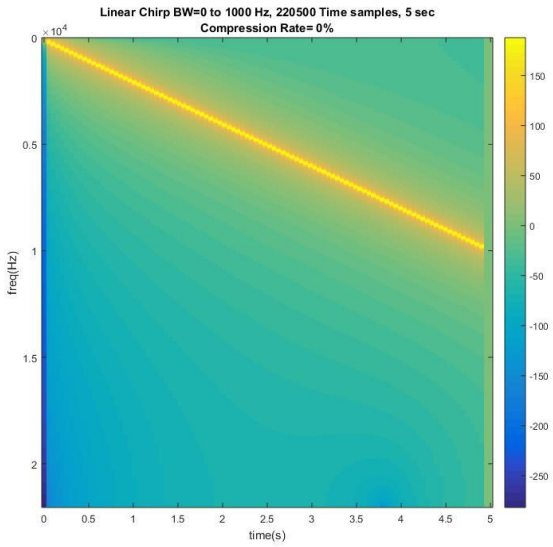

Figure 2: Spectrogram of the original linear chirp signal

- **Results obtained using downsampling:**

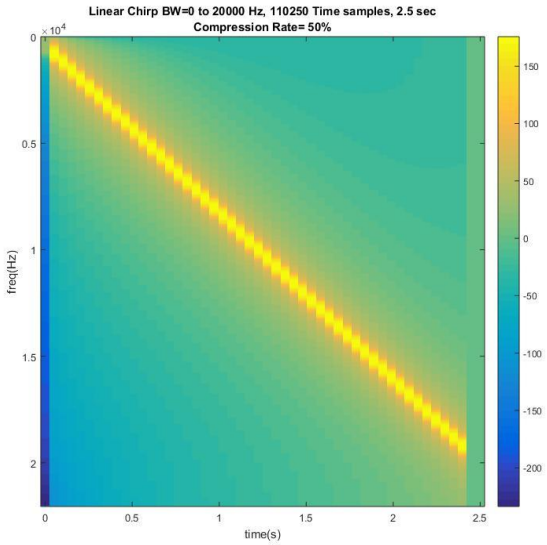

Figure 3: Spectrogram of the linear chirp signal downsampled by 2 (Frequency streches to 20 kHz which is equal to 10 kHz*downsampling rate)

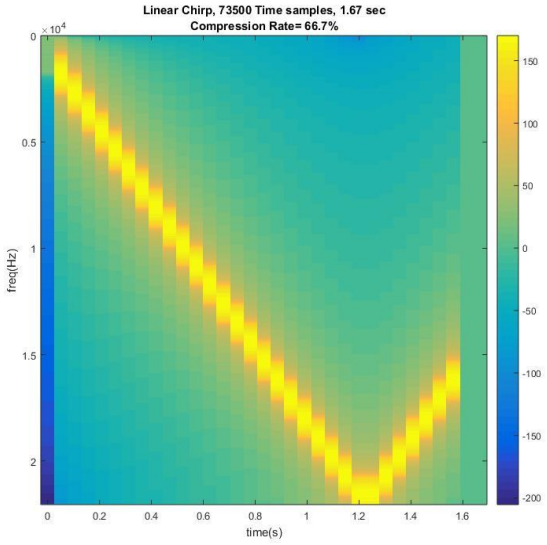

Figure 4: Spectrogram of the linear chirp signal downsampled by 3 (aliasing occurs for frequencies larger than fs/2)

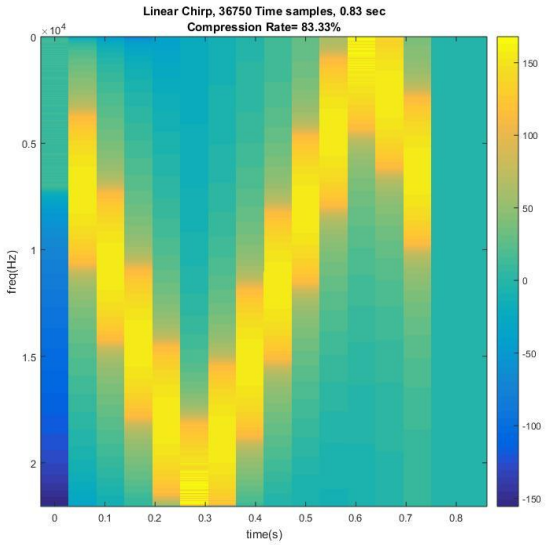

Figure 5: Spectrogram of the linear chirp signal downsampled by 6 (aliasing occurs for frequencies larger than fs/2)

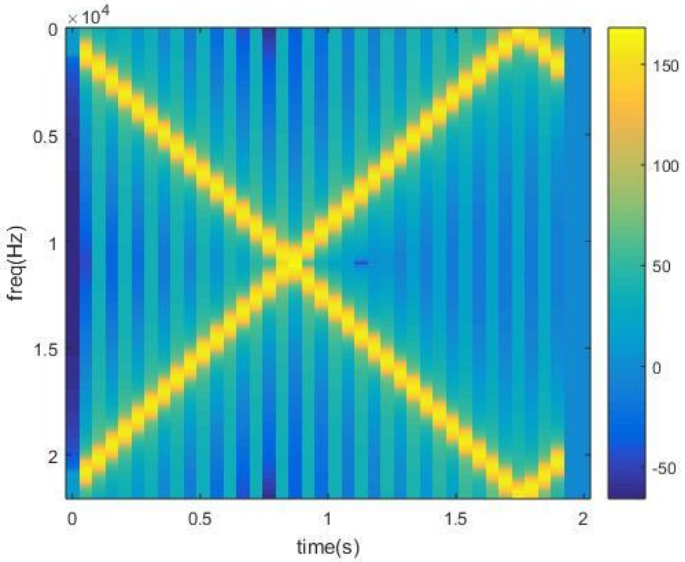

Figure 6: Spectrogram of the linear chirp signal downsampled by 2.5

As it can be seen in the figure, repetitions occur in the frequency domain because a LPF after the upsampling process is not used. Low pass filtered result is shown in the next part.

Results obtained using decimation:

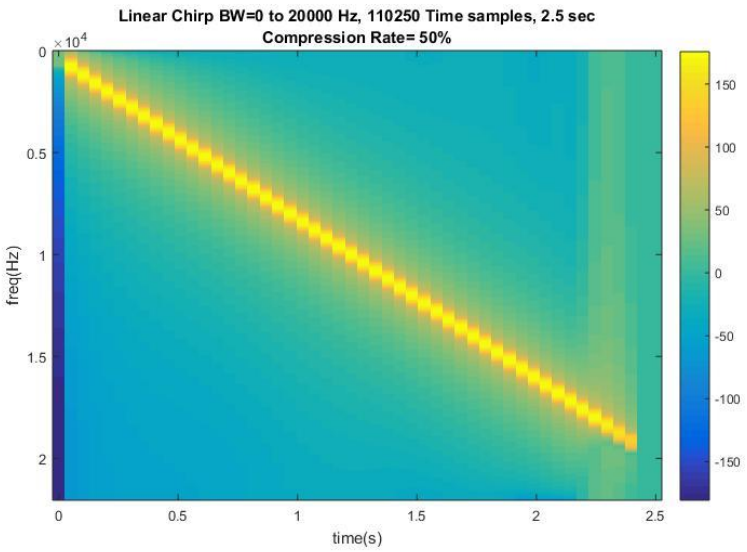

Figure 7: Spectrogram of the linear chirp signal decimated by 2

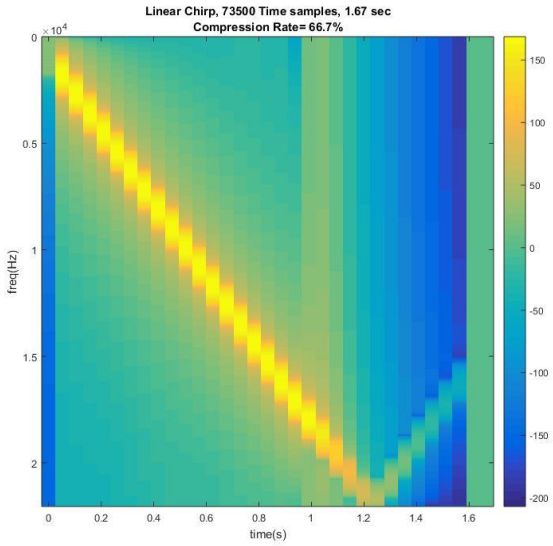

Figure 8: Spectrogram of the linear chirp signal decimated by 3 (frequencies which cause aliasing are suppressed by LPF)

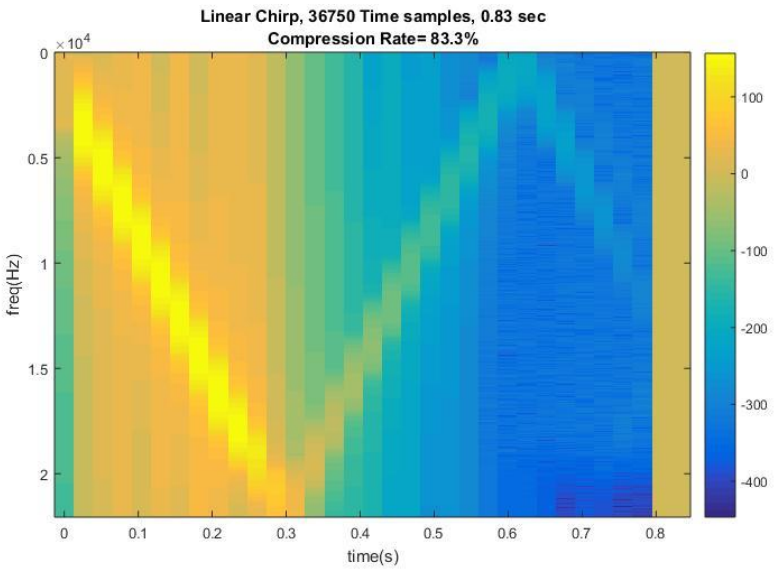

Figure 9: Spectrogram of the linear chirp signal decimated by 6 (frequencies which cause aliasing are suppressed by LPF)

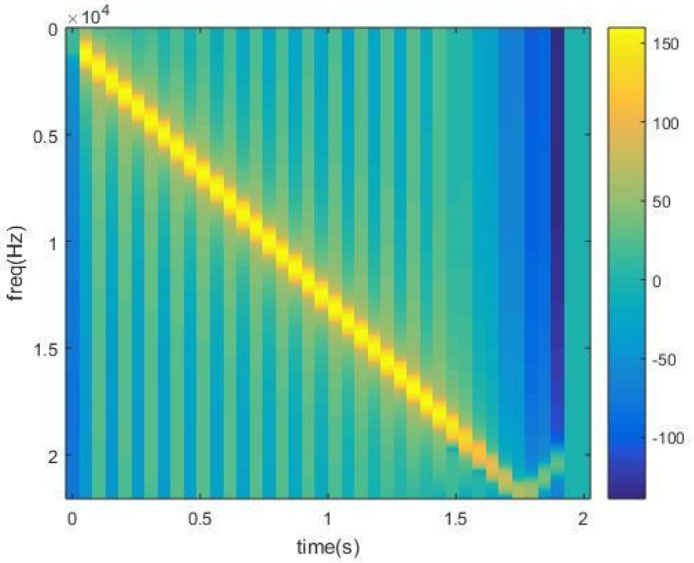

Figure 10: Spectrogram of the linear chirp signal decimated by 2.5 (frequencies which cause aliasing are suppressed by LPF)

b) Sound file input:

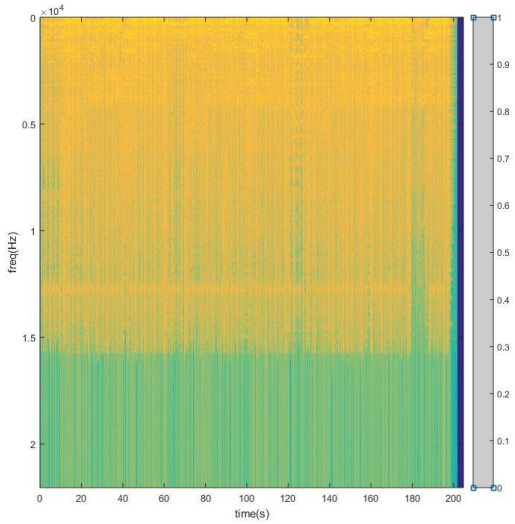

Figure 11: Spectrogram of the sound file input signal

- **Decimation applied on a sound file input:**

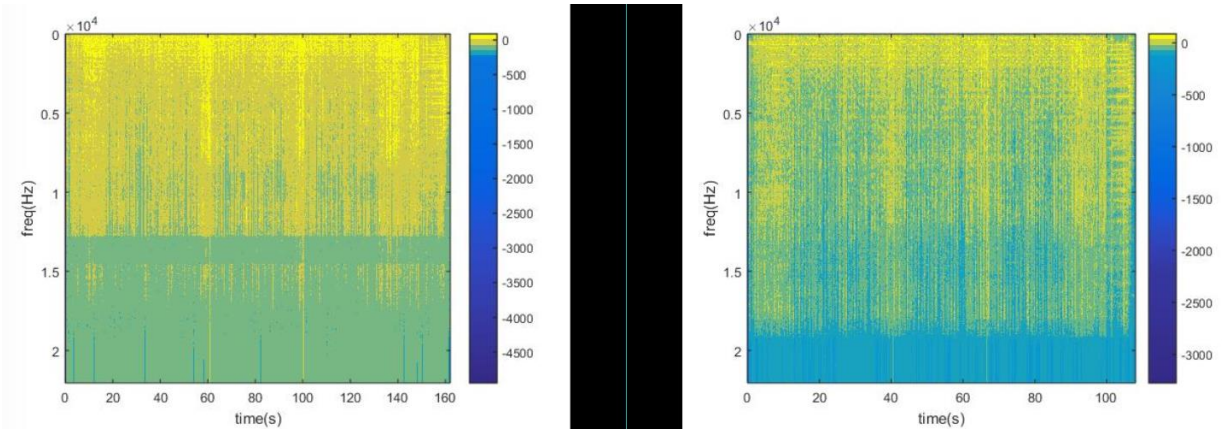

Figure 12: Spectrogram of the sound file input signal decimated by 2 and 3 respectively (Higher frequencies are suppressed by LPF)

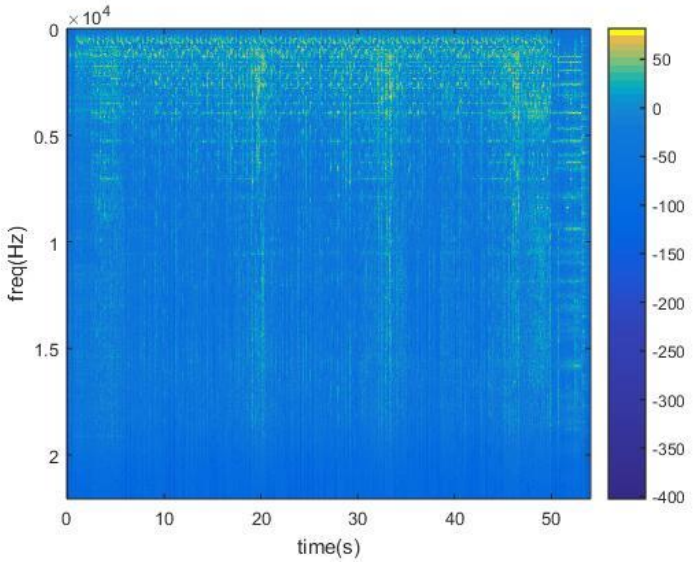

Figure 13: Spectrogram of the sound file input signal decimated by 6

- **Downsampling applied on a sound file input:**

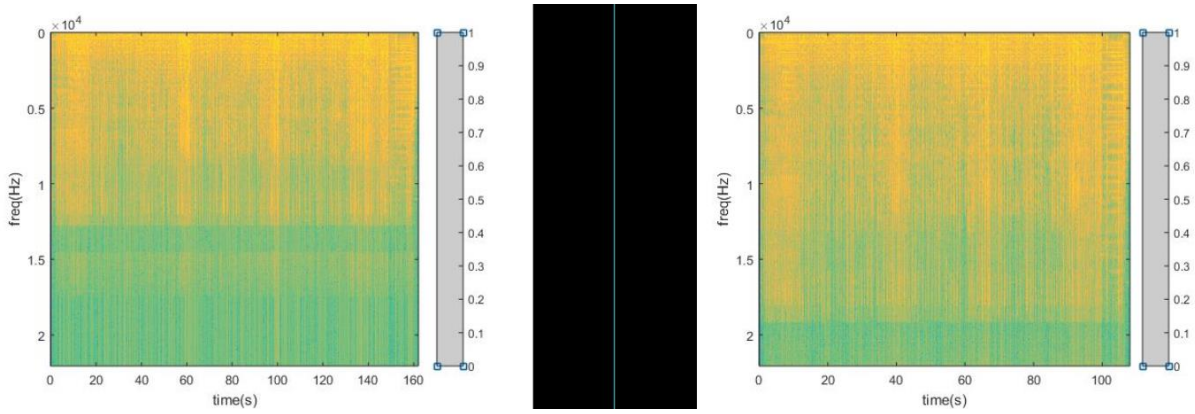

Figure 14: Spectrogram of the sound file input signal downsampled by 2 and 3 respectively

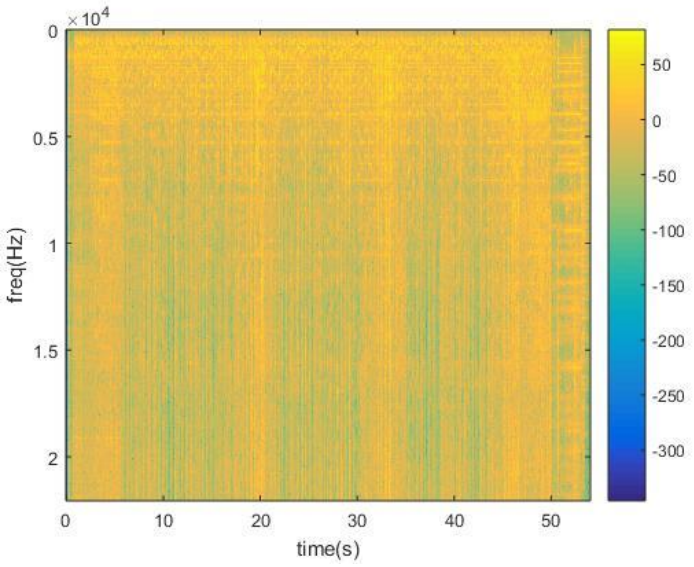

Figure 15: Spectrogram of the sound file input signal downsampled by 6 (Aliasing is quite observable)

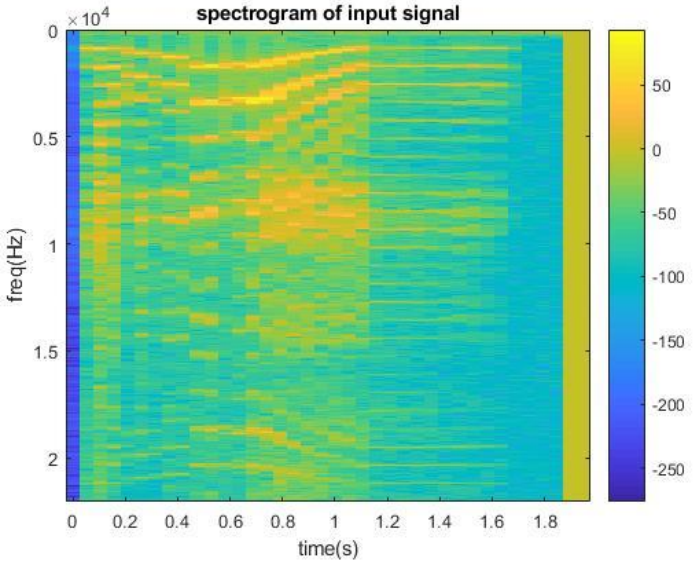

Figure 16: Spectrogram of the sound file input signal downsampled by 2.5 (A different sound file input is used)

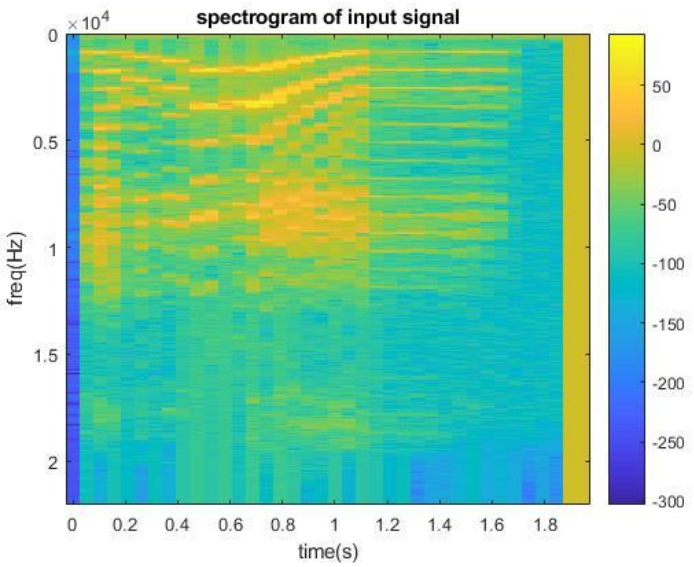

Figure 17: Spectrogram of the sound file input signal decimated by 2.5 (A different sound file input is used) (Lower magnitude high frequency components)

c) Recorded signal input:

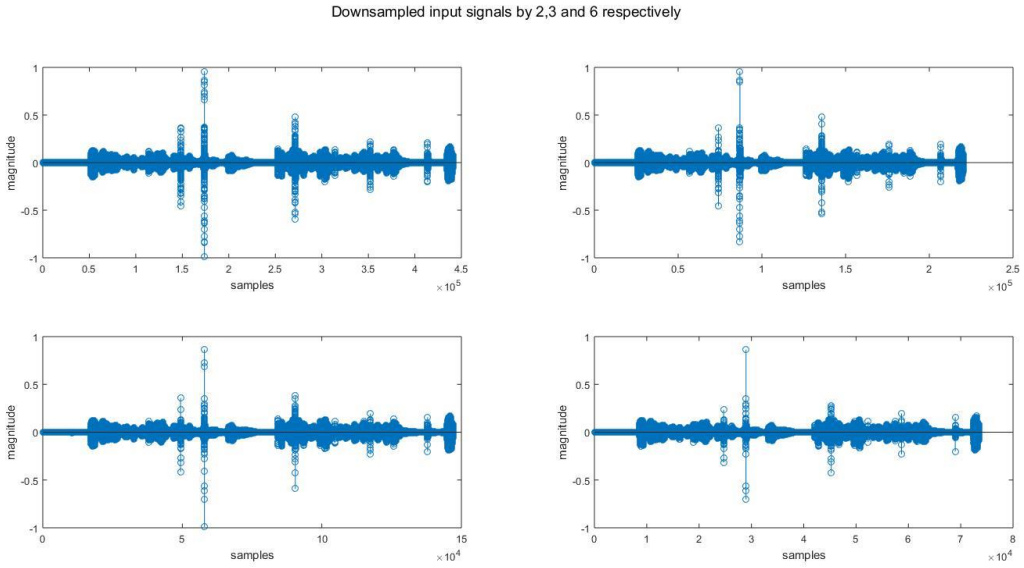

Figure 18: Time domain representations of the original, downsampled by 2, 3, 6 respectively

- **Recorded signal downsampled:**

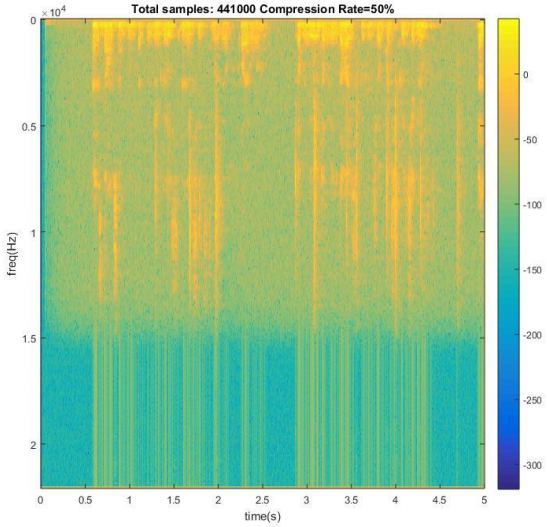

Figure 19: Spectrogram of the recorded signal downsampled by 2

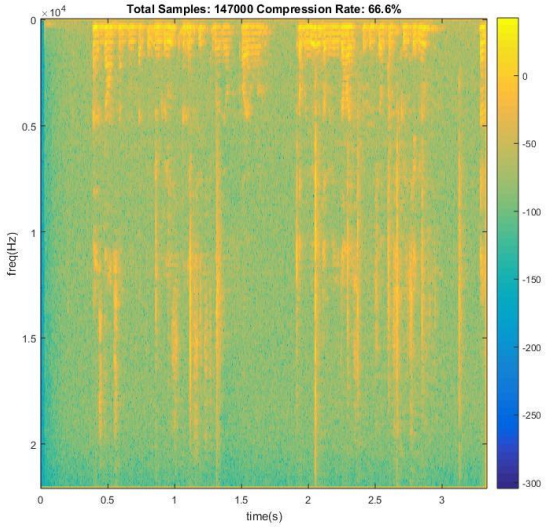

Figure 20: Spectrogram of the recorded signal downsampled by 3

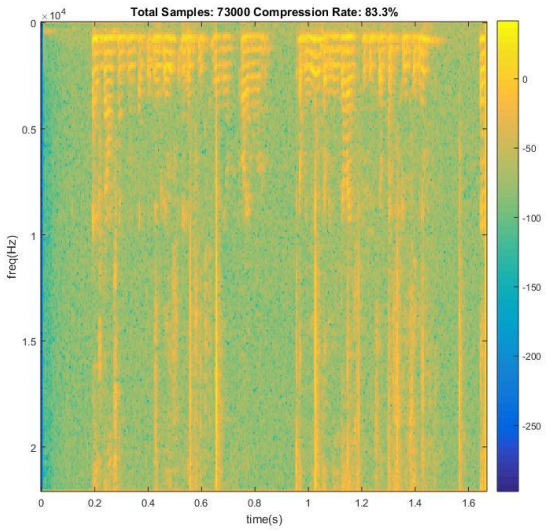

Figure 21: Spectrogram of the recorded signal downsampled by 6

- **Recorded signal decimated:**

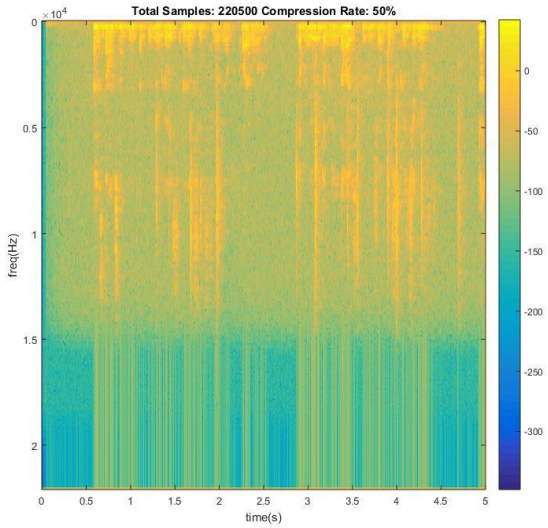

Figure 22: Spectrogram of the recorded signal decimated by 2

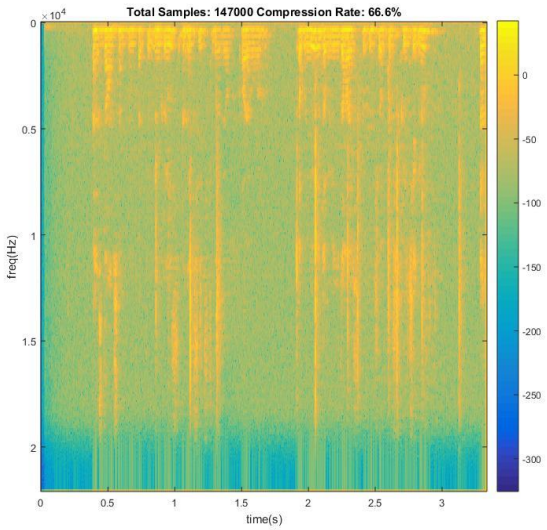

Figure 23: Spectrogram of the recorded signal decimated by 3

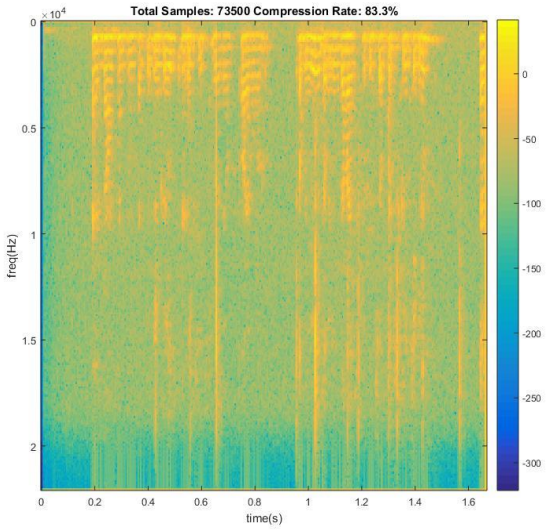

Figure 24: Spectrogram of the recorded signal decimated by 6

2) Reducing the Bits Per Sample on Time Domain:

- Quality of sound signal decreases each time we decrease the bits per sample because quantization error increases.
- For a 48000 sample/second sampling rate, bit rate is 48000(samples/sec)*8(bits/sample)/1024(kbit) =375 kbps.
- If we tried to increase the precision of sampling (such as trying to sample amplitude of signals with floating point precision), more bits per sample should have been stored.
- Quality could be increased by using a non-uniform quantizer by applying Lloyd-Max algorithm , which sets the quantization regions by the distribution of samples. It guesses an initial set of

uniform quantization points and it calculates decision thresholds and then, new representative quantization levels by making use of samples' probabilty function. And this process continues until no further distortion reduction is possible. Also, A-law and mu-law can be used as companding algorithms to quantize the signals non-uniformly. These algorithms provide extra resolution for low magnitude parts of the signals which is especially useful for speech signals, decreasing the total error.

Results:

For a sinusoidal signal with the formula y=10sin(2π2t) :

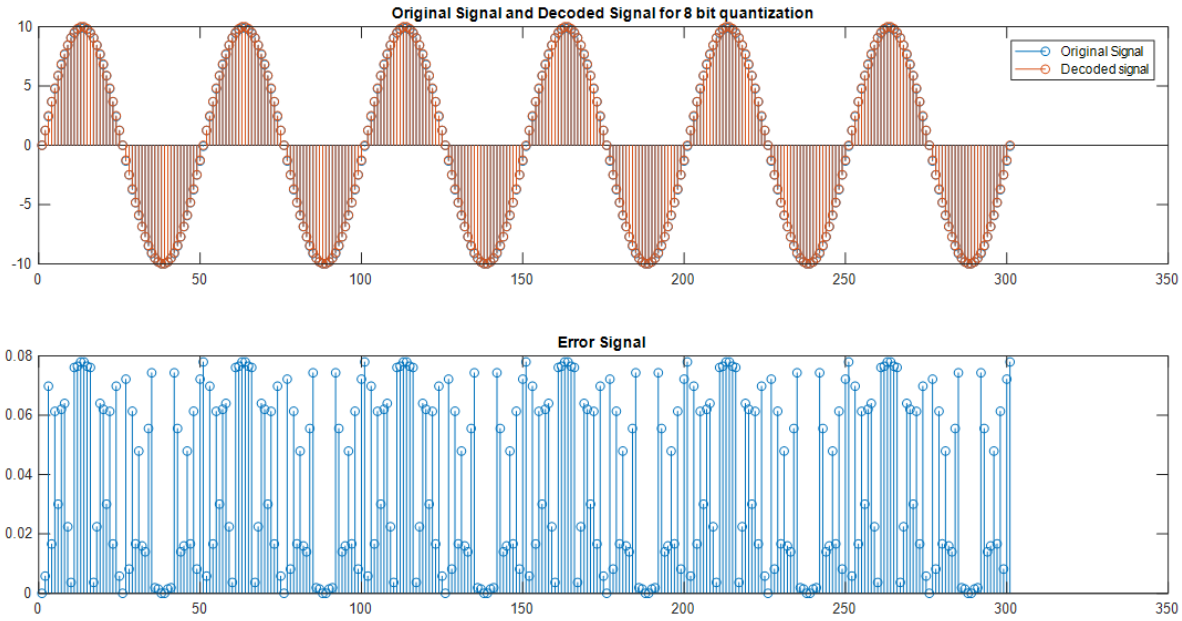

Figure 25: Original, decoded and error signals for 8 bit quantization for sine

Mean of error signal for 8 bit quantization for sine is 0.0389.

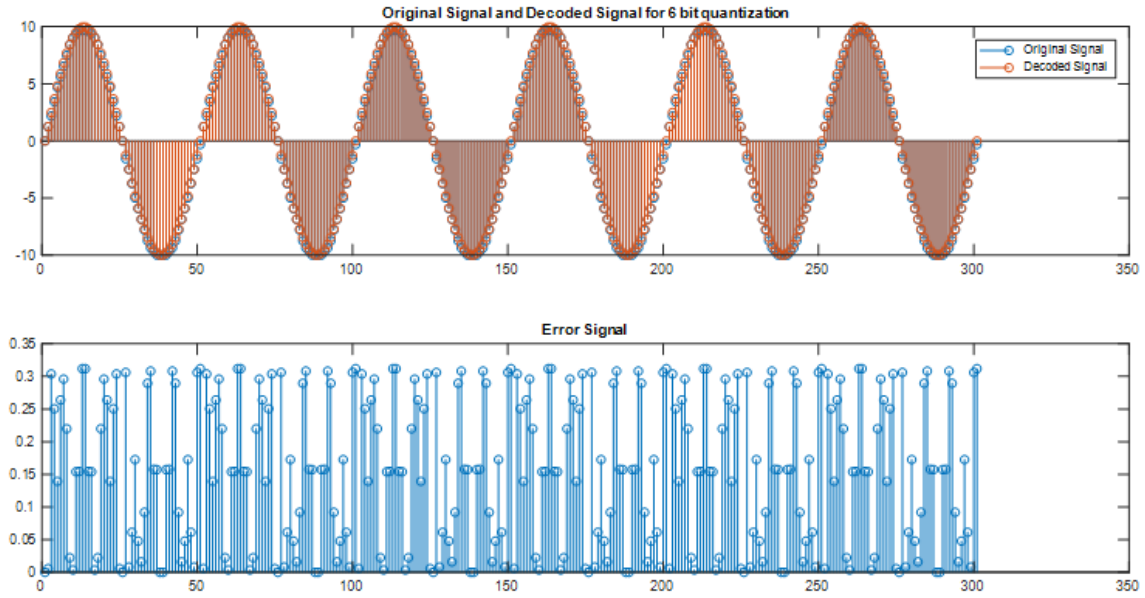

Figure 26: Original, decoded and error signals for 6 bit quantization for sine

Mean of error signal for 6 bit quantization is 0.1554 for sinusoidal.

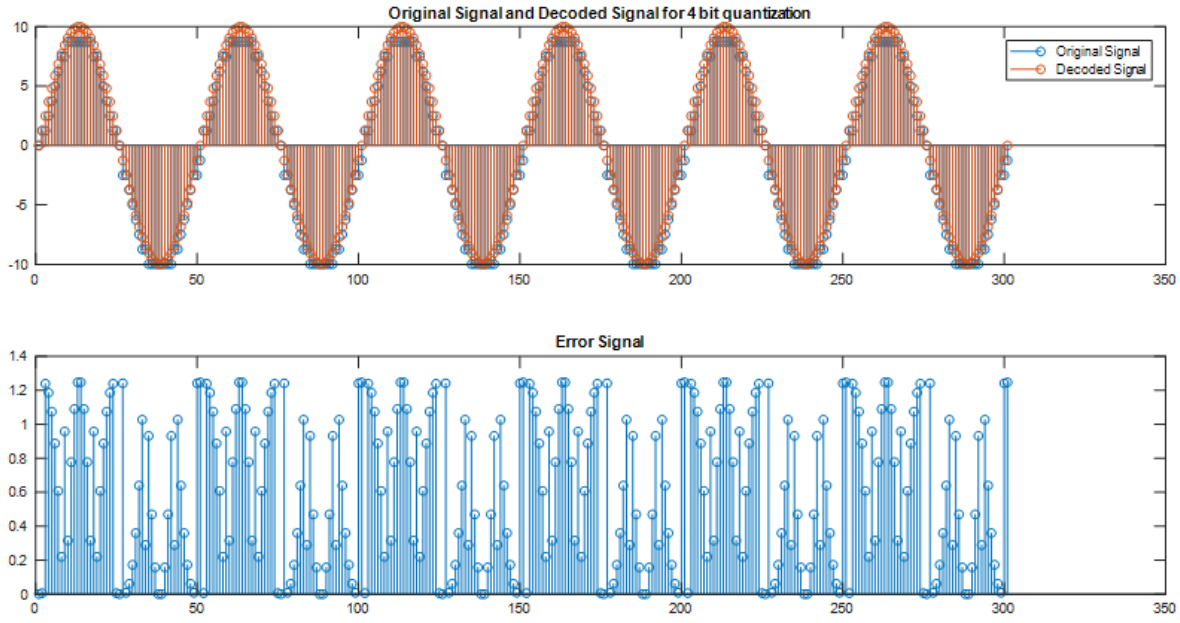

Figure 27: Original, decoded and error signals for 4 bit quantization for sine

Mean of error signal for 6 bit quantization is 0. 6217 for sinusoidal.

For a song, which we have taken 300 samples from, which is sampled with 44100 samples/second:

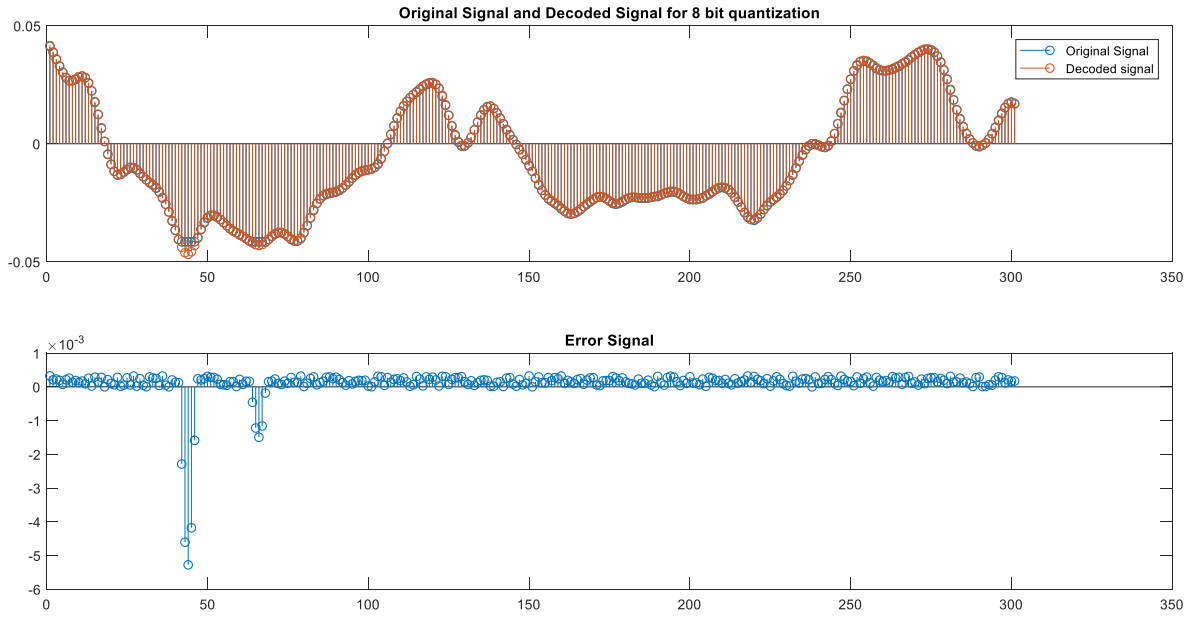

Figure 28 : Original, decoded and error signals for 8 bit quantization

Mean of power of error signal for 8 bit quantization is 3.01e-07.

Mean of absolute values of error signal for 8 bit quantization is 2.39e-04.

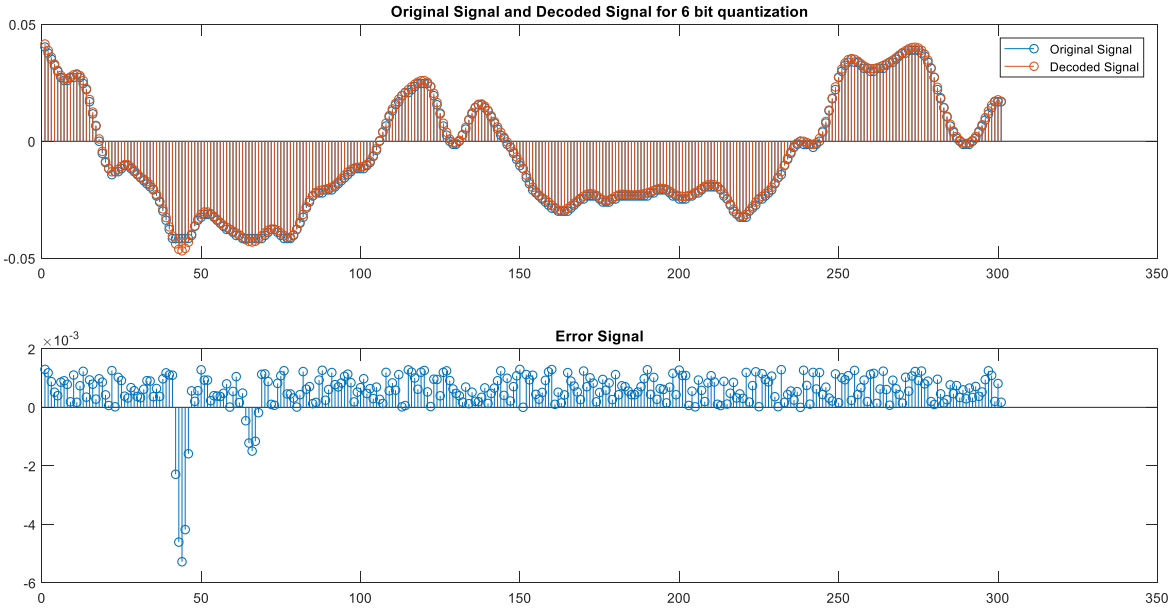

Figure 29 : Original, decoded and error signals for 6 bit quantization

Mean of power of error signal for 6 bit quantization is 8.24e-07.

Mean of absolute values of error signal for 6 bit quantization is 7.12e-04.

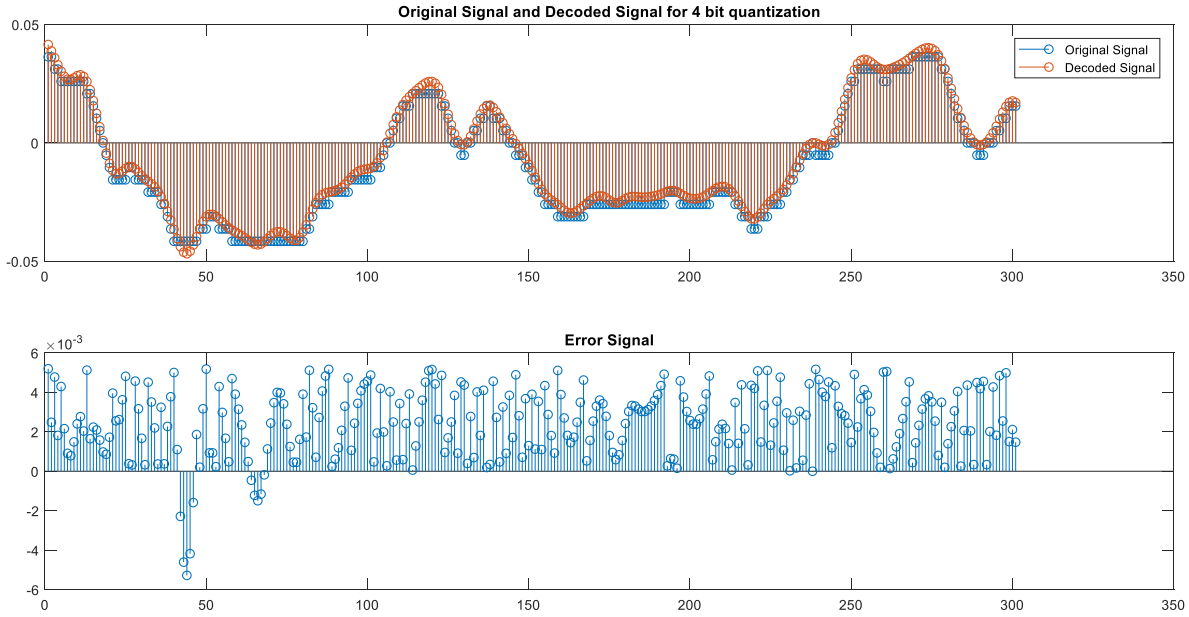

Figure 30: Original , decoded and error signals for 4 bit quantization

Mean of power of error signal for 4 bit quantization is 8.99e-06.

Mean of absolute values of error signal for 4 bit quantization is 2.6e-03.

- As the results show, decreasing the bit numbers for quantization increases the quantization error magnitudes which results in a worse and "quantized" hearing experience.

Resulting number of stored bits for the 5 seconds of the same signal is:

For 8 bit quantization : 48000*5*8= 1920000 bits

For 6 bit quantization : 48000*5*6= 1440000 bits

For 4 bit quantization : 48000*5*4= 960000 bits

3) Transform Coding:

Using DFT: In this part, quantization is applied to frequency domain coefficients. As indicated in the description of the project, input signal is divided into N partitions and Fast Fourier Transform (FFT) is applied for each partition. After that, coefficients of these partitions are sorted in ascending order and a threshold is determined for them using the total length of partitions. If we call the total length as N and if we want a threshold that makes 90% of the coefficients zero, threshold is decided as the N*(90/100)th element of ascending order sorted coefficient matrix. After deciding on the threshold, elements of the unsorted matrix which are smaller than this threshold are made zero for each partition. Next step is taking inverse fast fourier transform of each partition and concatenating them in the order of how we partitioned the complete signal.

Using DCT: Similar to the procedure for DFT, input signal is again divided into N partitions and instead of FFT, DCT-2 is applied for each block and blocks are thresholded according to the desired rate of compression. This is done in a similar way with the previous part. All components are sorted and respective component which has a magnitude larger than required percentage of remaining signal components is selected as the threshold for that block. Then, inverse DCT is computed and concatenation is done.

Using MDCT: Explanation and figures related to this part are added at the end of this section.

Comments and results:

We can see that DCT has better quality when compared with DFT. This is because of the reason that in DCT low frequency components are emphasized and as we are speech processing it gives much better results. For high compression rates, it is very audible. In the following figures, you can see the error signals of DFT and DCT . DCT provides better energy compaction. **For your convenience, sound signal is multiplied by 10^8 to see the difference of error signals easier.**

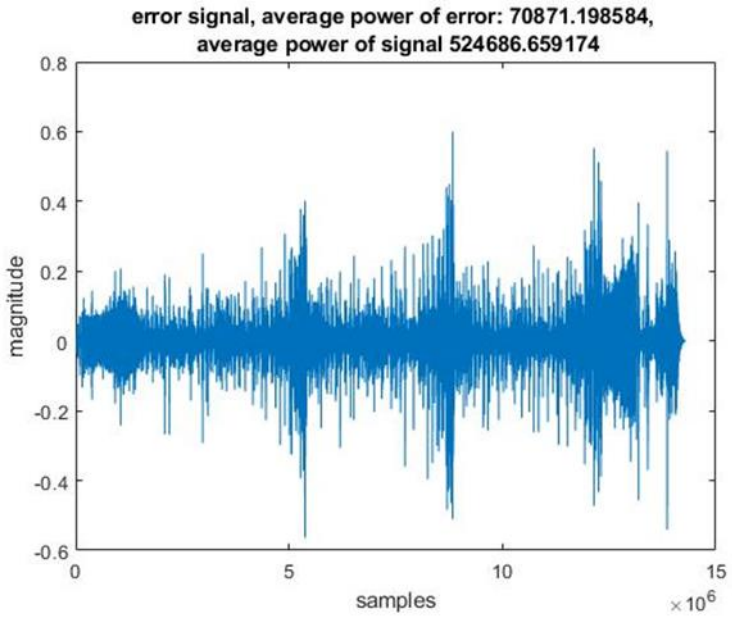

Figure 31 : Error signal of a sound signal compressed with DCT(Blocksize=429 , 99% compression)

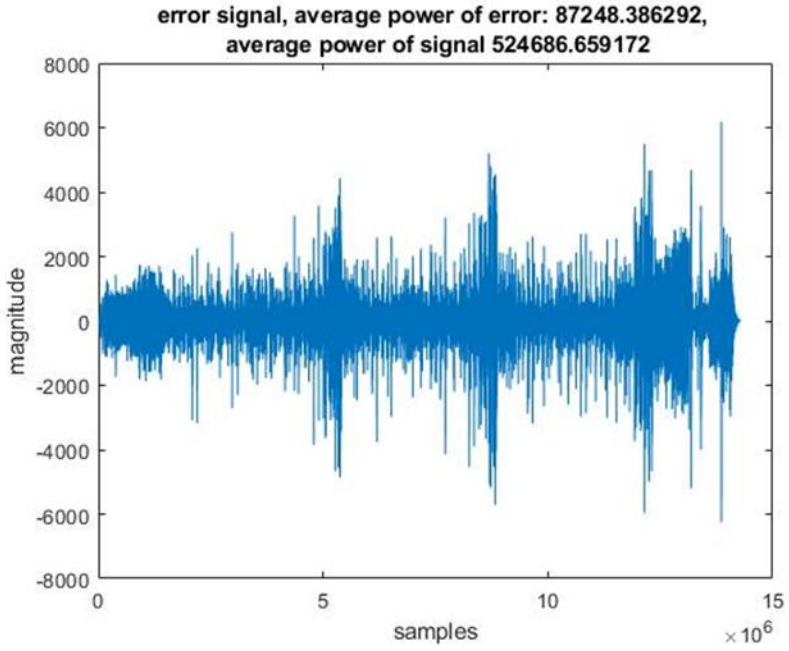

Figure 32 : Error signal of a sound signal compressed with DFT(Blocksize=429 , 99% compression)

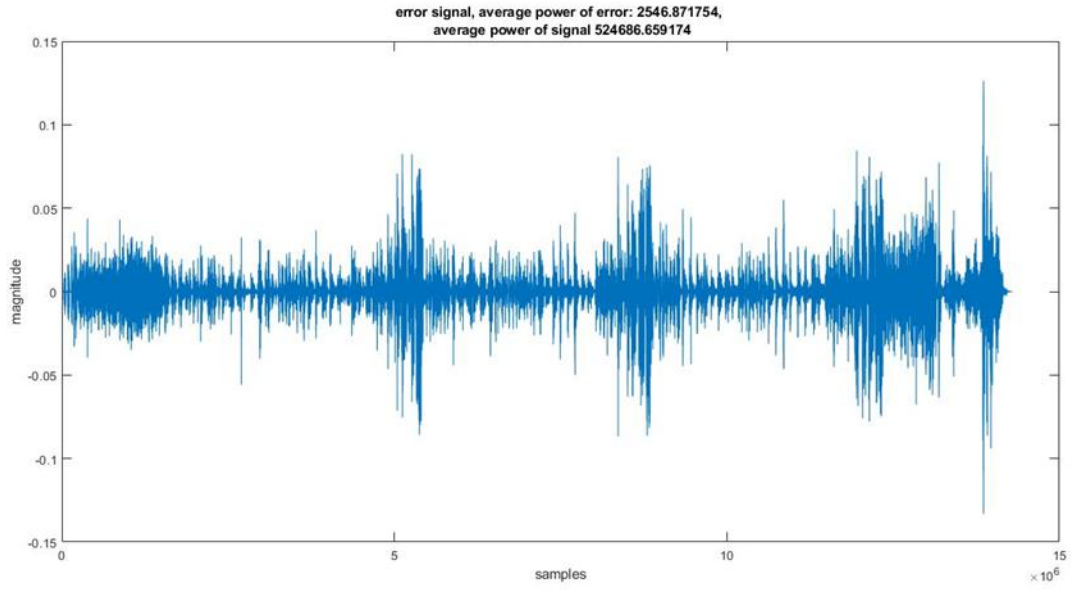

Figure 33 : Error signal of a sound signal compressed with DCT(Blocksize=429 , 90% compression)

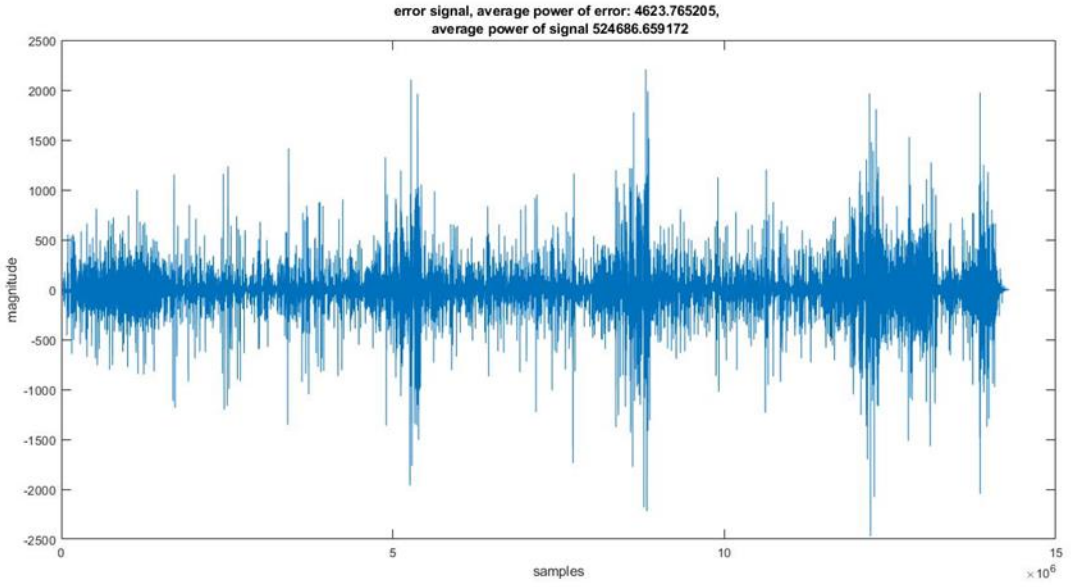

Figure 34 : Error signal of a sound signal compressed with DFT(Blocksize=429 , 90% compression)

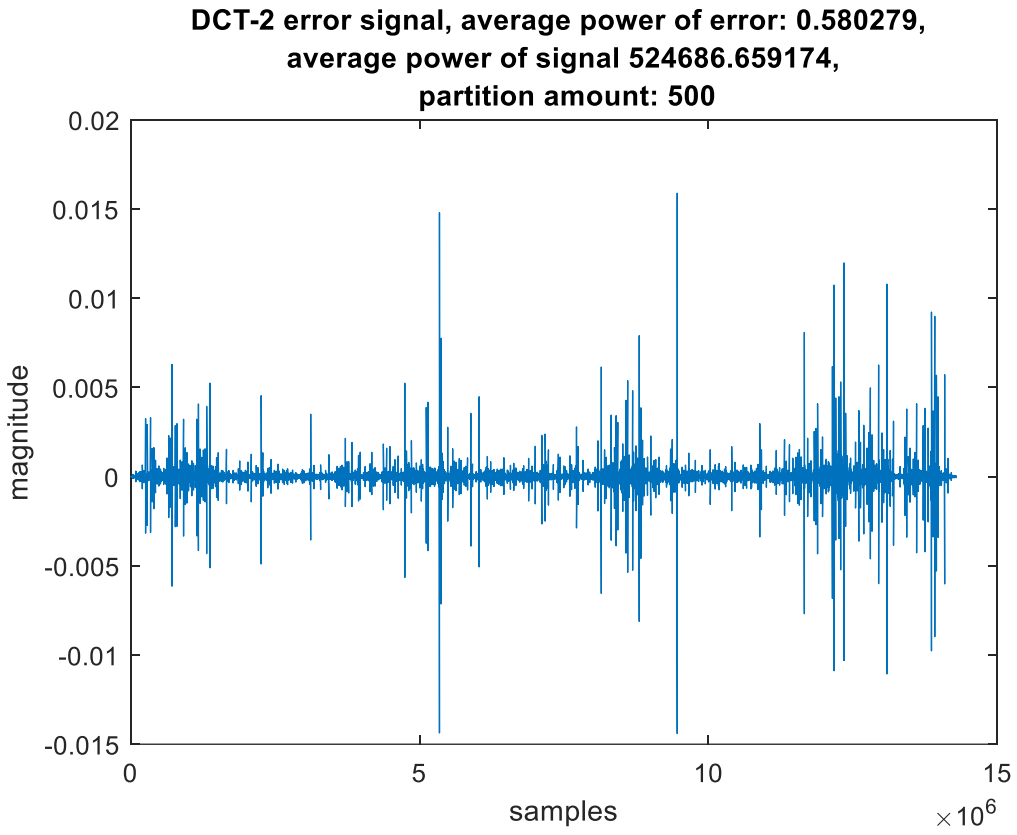

Figure 35: Error signal of a sound signal compressed with DCT(Blocksize=429 , 70% compression)

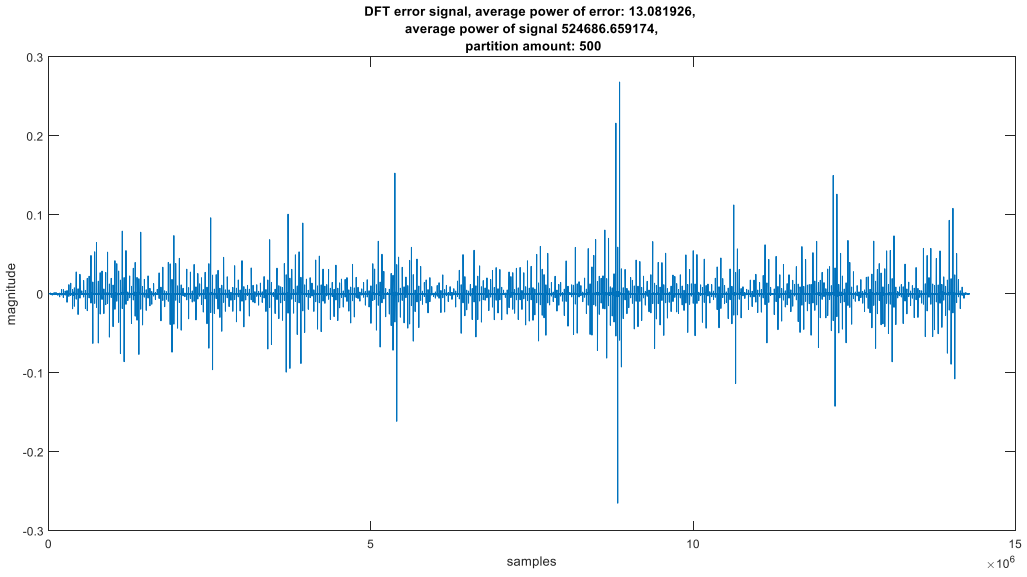

Figure 36 : Error signal of a sound signal compressed with DFT(Blocksize=429 , 70% compression)

For 70% compression rate, signal is compressed without almost any errors.

- We can see that , more high frequency components exist in the frequency response of DFT compared to DCT . This is because DFT is calculated with the extension of input signal to make it periodic and artificial discontinuities occur . This is not the case in DCT as DCT makes input signal periodic by adding its symmetric to its right and then shifts it. Therefore in DCT **, artificial high frequency components do not appear** and it enhances the quality. You can see it in figures below. This also **provides better energy compaction** for the DCT algorithm, decreasing the total energy of the resulting compressed signal since cosine has greater magnitudes for lower frequencies and magnitude of zero for pi/2 radians and even symmetrical extension of the input signal provides extra high frequency noise.

Figure 37: Comparison of extending methods for DFT and DCT

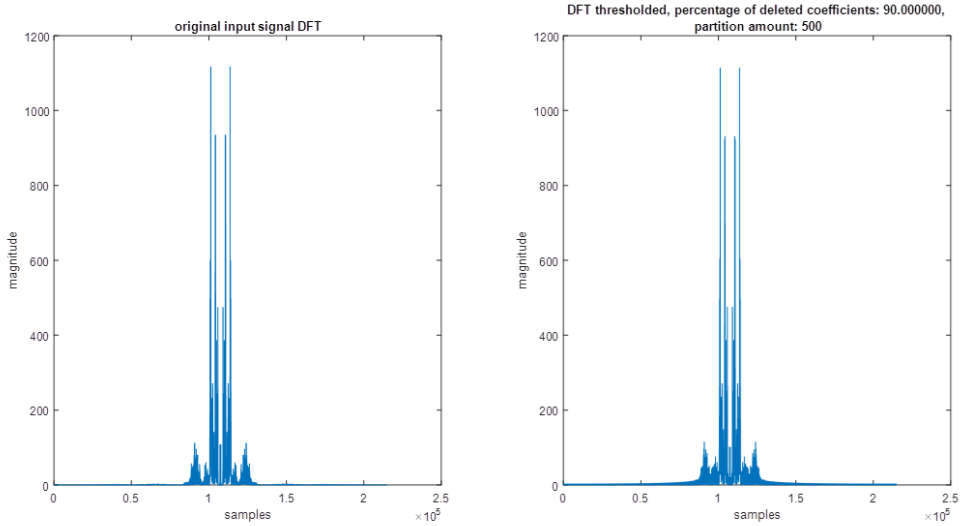

Figure 38 : Comparison of DFTs of original signal and DFT thresholded signal(%90 compression)

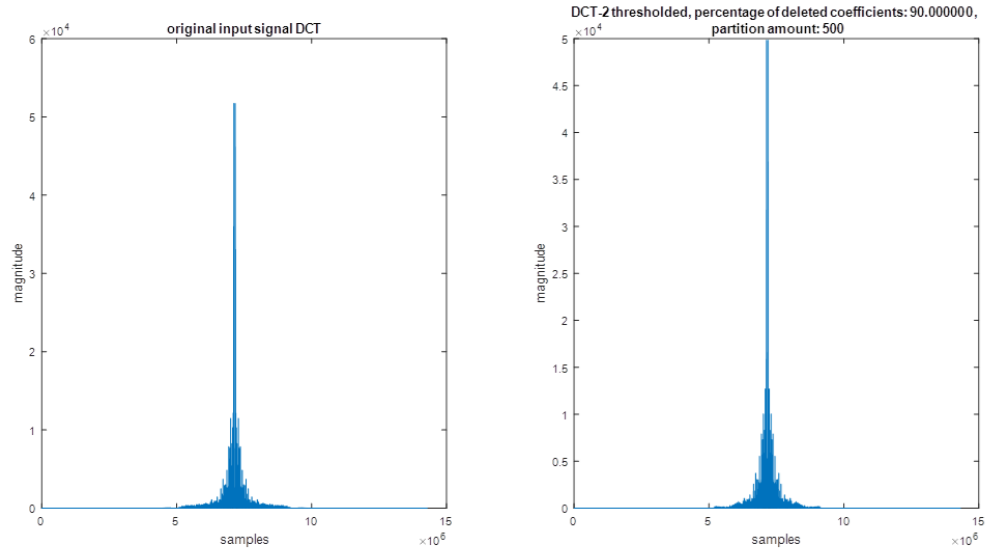

Figure 39 : Comparison of DFTs of original signal and DCT thresholded signal(%90 compression)

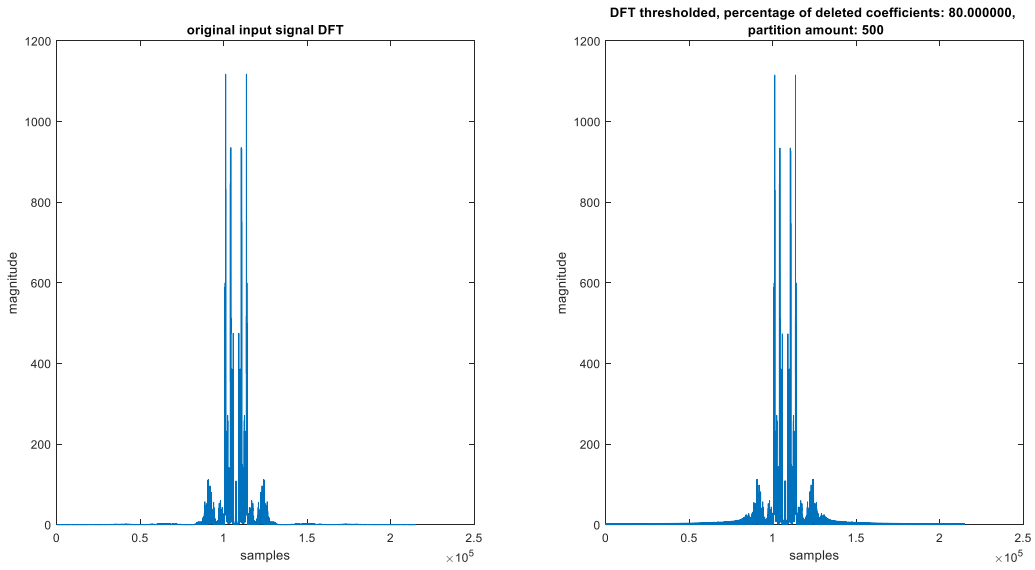

Figure 40: Comparison of DFTs of original signal and DFT thresholded signal(%70 compression)

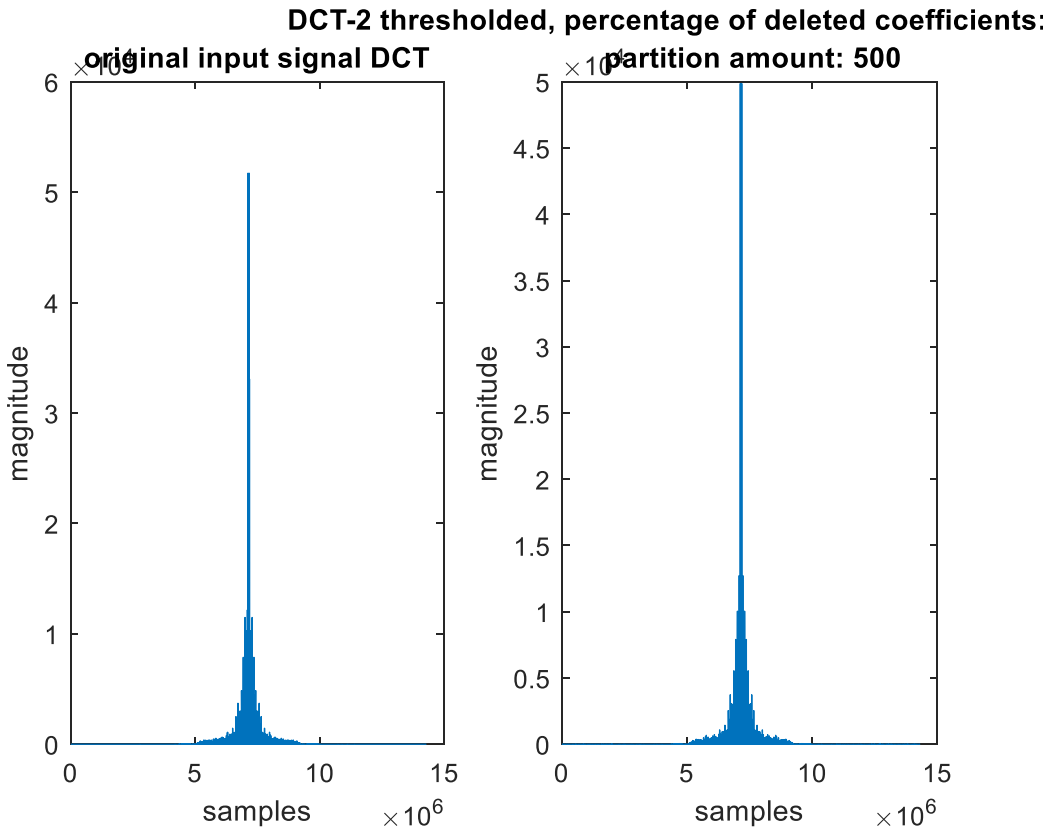

Figure 41 : Comparison of DFTs of original signal and DCT thresholded signal(%70 compression)

For %90 compression rate and an input signal's DFT which has 28252*501=14154252 coefficients , number of coefficients set to zero is 12861500 which is approximately 90% of the coefficients. It is nearly the same for DCT as compression rate is the same.

When we increase the blocksize from 429 to 859 (approximately 2 times), errors decrease for every compression rate and compression method (DFT & DCT-2). This makes sense because when we increase the blocksize, we apply compression on a larger portion of the audio signal which means that we take larger amounts of frequency components into account. Due to this fact, suppression of components are done in a more healthy way (components that really has smaller magnitudes among the signal are eliminated instead of components that have smaller magnitudes in an individual partition but spans along the signal with a greater magnitude).

Results for larger blocksize:

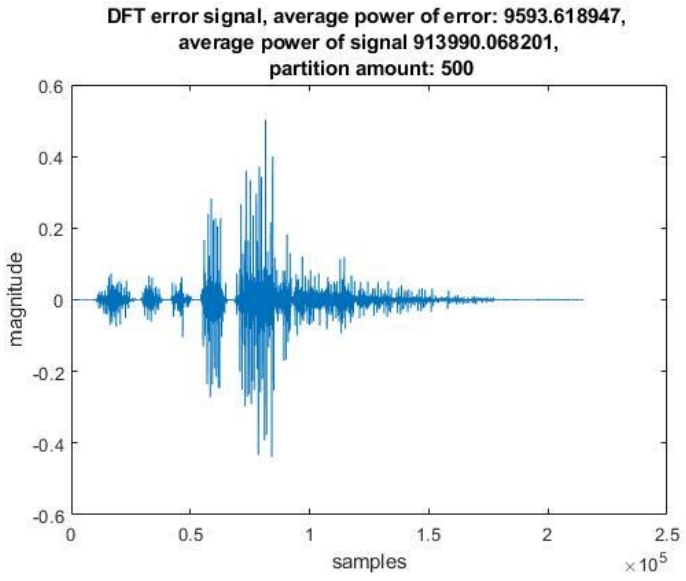

Figure 42 : Error signal of a sound signal compressed with DFT(Blocksize=429 , 90% compression)

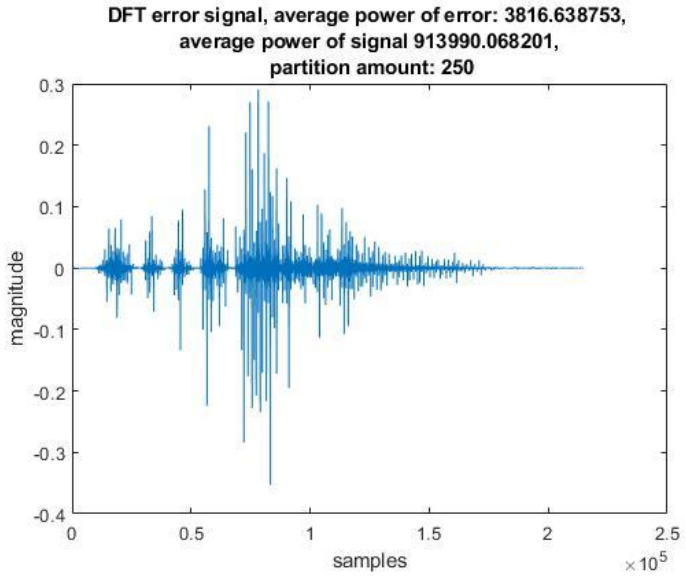

Figure 43 : Error signal of a sound signal compressed with DFT(Blocksize=859 , 90% compression)

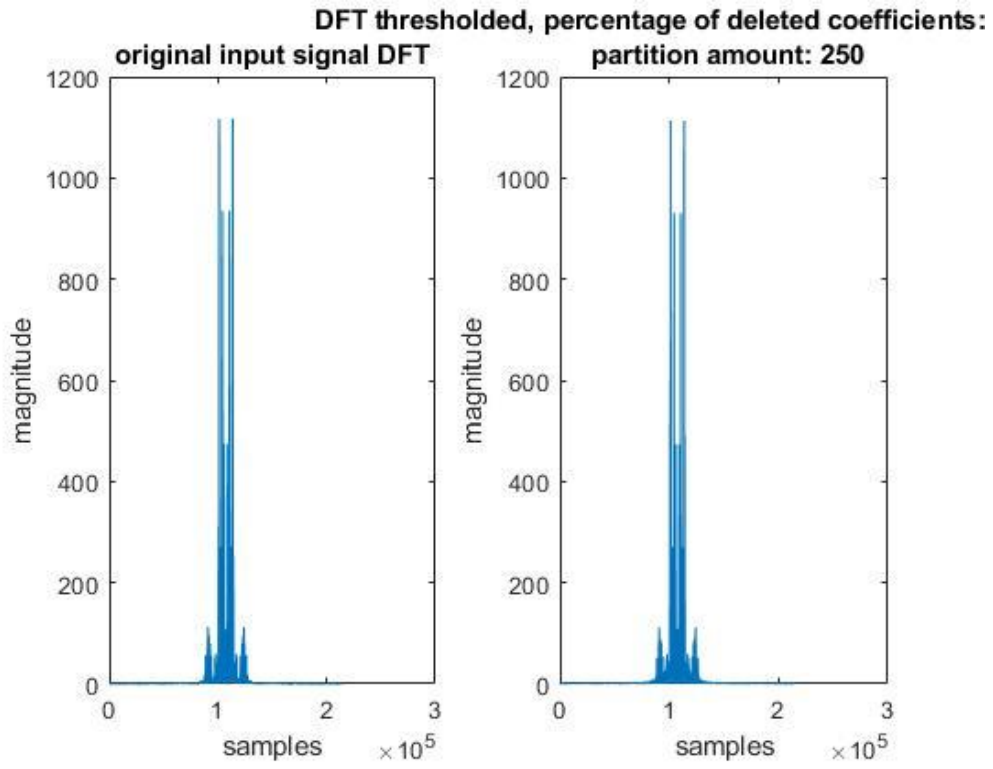

Figure 44: Comparison of DFTs of original signal and DFT thresholded signal, blocksize = 429

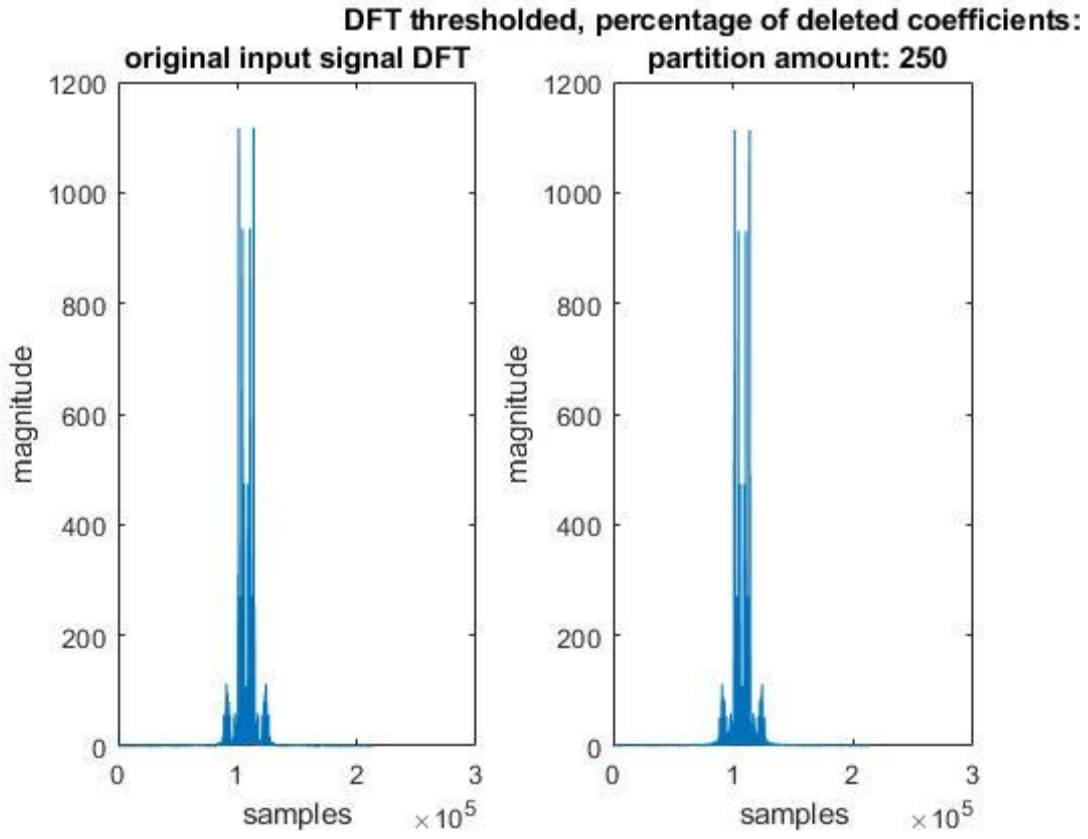

Figure 45: Comparison of DFTs of original signal and DFT thresholded signal, blocksize = 859

Bonus part – MDCT:

As indicated in the project description, MDCT implementation is a bonus part. In order to implement compression using the modified DCT, a different type of buffering should be implemented with each partitions overlapping with each other by 50%. After windowing, according to the formula,

$$y_n \triangleq \begin{cases} -x_{n+3N/2} - x_{3N/2-1-n}, & 0 \le n < N/2 \\ x_{n-N/2} - x_{3N/2-1-n}, & N/2 \le n < N. \end{cases}$$

Overlapped blocks are combined. Then, MDCT coefficients are calculated using DCT type 4. Thresholds for blocks are found and applied as in the previous cases after DCT-4. By using DCT-4 once more, Bn vectors are calculated. (DCT-4 also works as inverse transform at the same time) After that, N Bn vectors are combined as indicated in the figure and 2N alpha vectors are obtained. Lastly, same windows are applied to these alpha vectors and they are combined by summing alpha vectors of index n and a delayed version of n+N/2. If the windows provide the two conditions listed in the description appendix, then input signal is obtained at the output.

We followed those steps in the description and compressed the signal accordingly. DFT's of compressed signal and original one resembled to each other quite a lot, meaning that magnitudes

of frequencies are saved. However, when we use **windows as 1/sqrt(2),** we **deteriorated the phase information** of the signal, resulting in “high quality” voices with the wrong timing. This resulted in a high quality signal with slight timing errors. **Phase deterioration can be enhanced by using the sine** window included in the description. Results are listed below. We should note that MDCT provides greater similarity (after compression, nearly no additional noise occur on the signal as it can be seen in the figures below for rectangular window) for magnitude response compared to previous methods. This is due to the overlapping blocks. Using overlappings, **MDCT gets rid of the discontinuities** at the ends of the individual blocks which cause high frequency noise after compression.

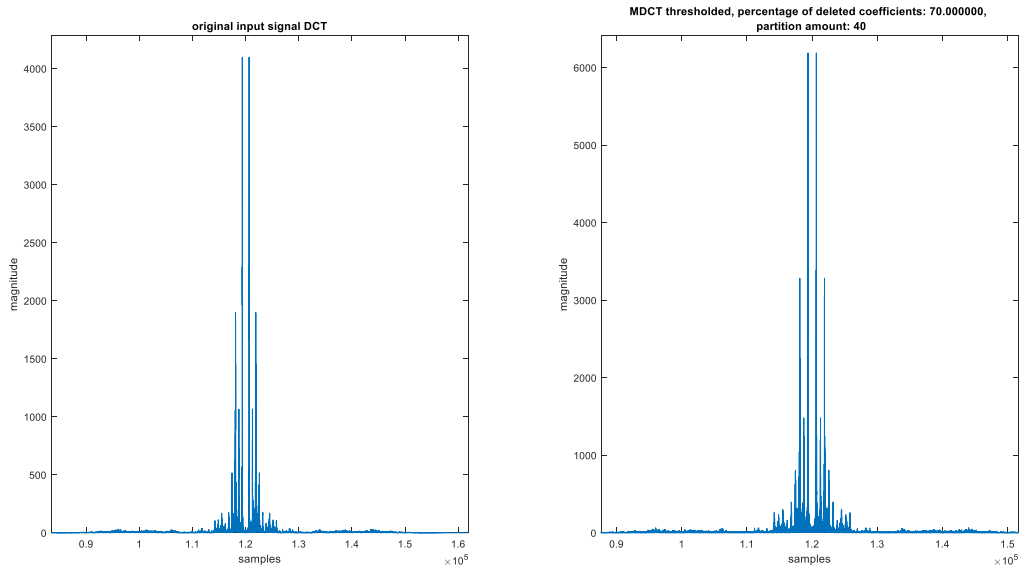

Figure: Magnitude response of the input and compressed recorded sounds for 5 sec using rectangular window (70% compression)

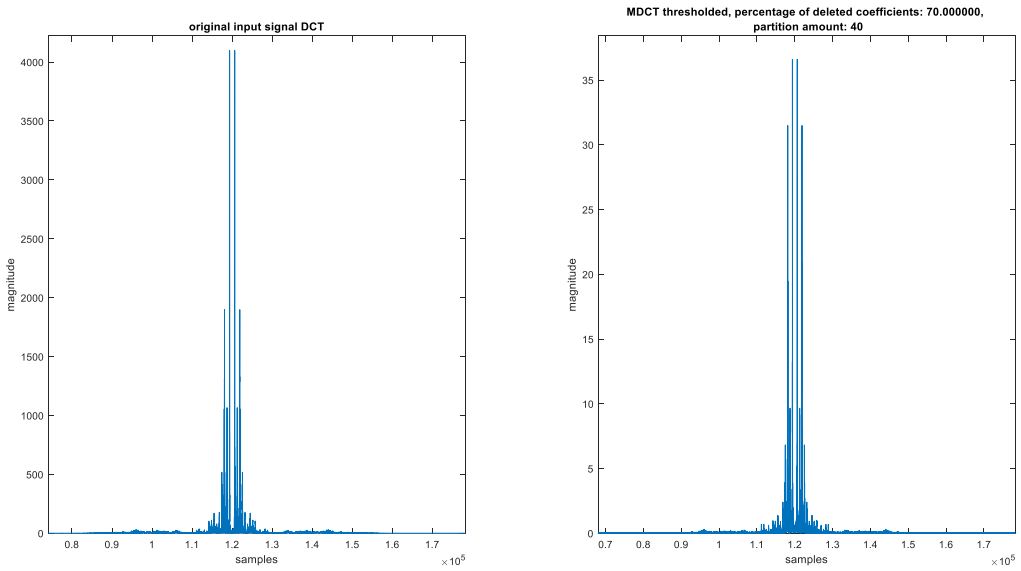

Figure: Magnitude response of the input and compressed recorded sounds for 5 sec using sine window

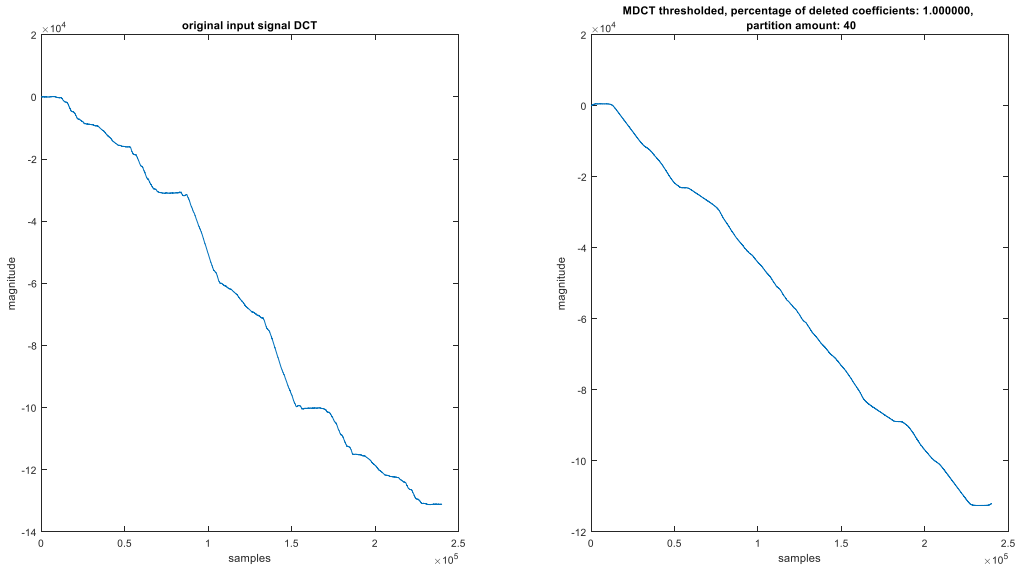

Figure: Phase of the input and compressed recorded sounds for 5 sec using sine window (phase is less deteriorated)

4) MPEG / Audio Encoder:

MPEG Layer 3 is an audio coding format for digital audio that quantizes the transform domain coefficients.It reduces the memory requirements to store a sound waveform. In this final part of the project, we were required to complete an unfinished code sequence which takes an input file which has a format ".wav", compresses and saves it in the format ".mp3". There are 3 main tasks that should be completed. First task is the prototype filter design.

a) Prototype Filter Design: MPEG/Audio encoding algorithm uses subband filtering including a bank of 32 subband filters with real coefficients. These subband filters result in 32 subband signals. These filters cover the entire spectrum from 0 to pi, each spanning pi/32. These filters are identical to each other, only difference between them is that the modulation with a cosine to obtain a shifted version in the frequency domain to span whole frequency range. So, all of them can be obtained from a prototype filter.

As indicated in the project description, this filter is an 512-tap finite impulse response filter, with the desired low pass response. According to the specifications, we have constructed the prototype filter using "firpm" command of MATLAB.

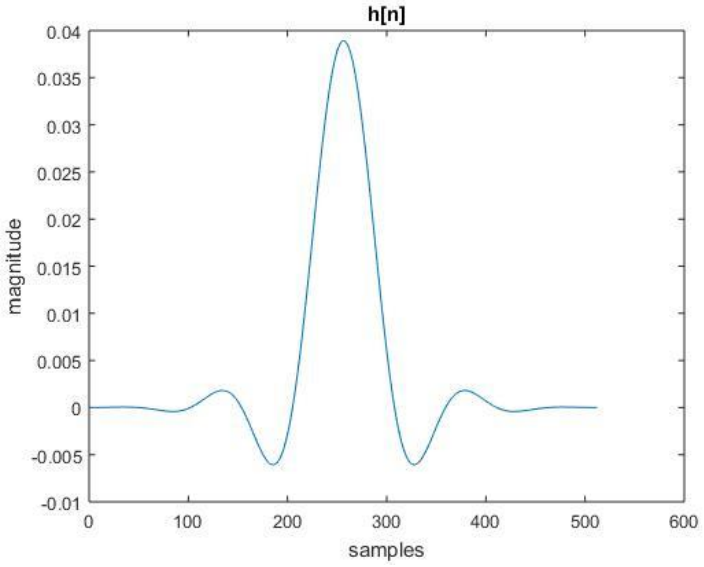

Figure 46: Time domain plot of the prototype filter

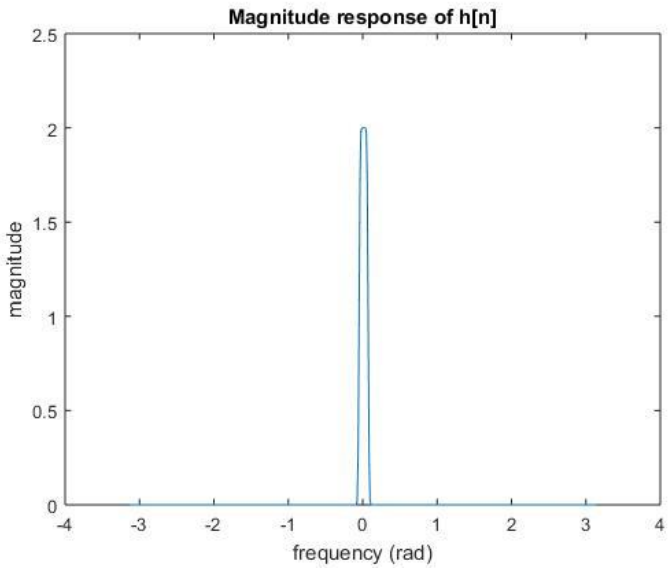

Figure 47: Magnitude response of the prototype filter

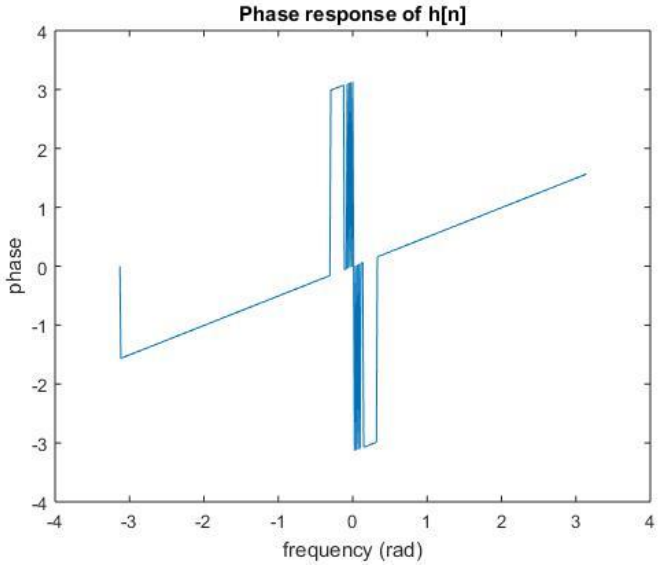

Figure 48: Phase response of the prototype filter

Prototype filter has linear phase characteristics as it can be seen in the figure.

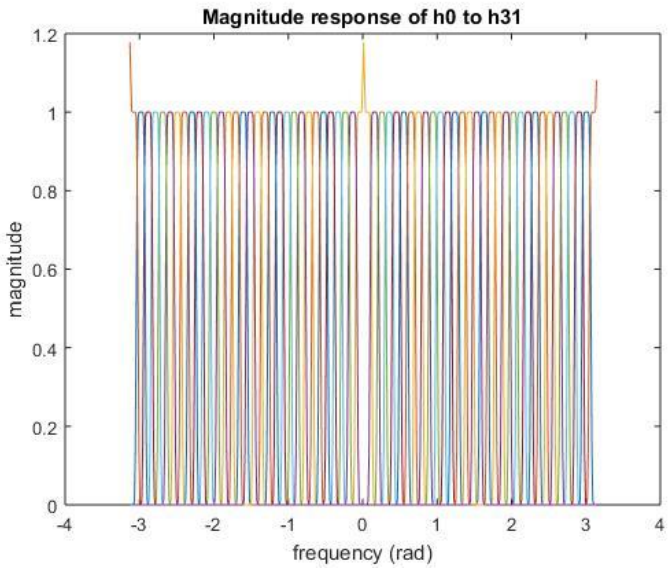

Figure 49: Magnitude response characteristics of H0 to H31 on a single plot

As it can be seen in the figures, frequency domain characteristics of H0 to H31 covers the entire spectrum since we multiply them with cosines that has frequencies increasing in the order of pi/32.

b) Subband Filtering: After applying 32 filters designed in the first part, resulting signals are downsampled by 32. However, applying straightforward convolution is inefficient in terms of computational concerns. Using the fact that $h_k[n]$, k=(0,..,31), are simply cosine modulated versions of each other and periodicity of cosine, a computationally efficient version of the operation is obtained by changing the summation index by 2Np+q where N is the order of downsampling and q=0,...,2N-1. Using 512-point circular buffer, subband filtered outputs $s_k[m]$ are obtained after downsampling.

c) Spectrum Estimation: MPEG encoder algorithm allocates different number of quantization bits for subbands, according to the masking levels of components. These levels are determined by the psychoacoustic model which assigns these bits to the regions with respect to their perceptual importances fot human ears. Subbands that are difficult to perceive are allocated very few bits (Higher frequency subbands). Human ear masks side frequency components near a strong component when it is heard according to the frequency and magnitude of the dominant component. If side components do not exceed the levels of the mask that ears apply, then they can not be heard. This masking gets stronger as the frequency of the strong component increase, meaning that we can not hear multiple high frequency components with different amplitudes in a high frequency subband whereas we can hear close components in the low frequency subbands, dividing the spectrum into approximately 24 critical bands. This leads to the corresponding bit allocation.

d) Quantization: Quantizer function that is required to be implemented quantizes the sample according to the required bit number obtained from the psychoacoustic model, quantization parameters and scaling factor which is used to obtain a the reconstructed signal without a magnitude deflection since spectrum estimation is done by scaling all subband signals to 96 dB

magnitude. Without this factor, magnitude information of subband signals are lost and compression results in equal magnitudes for all bands which results in a completely wrong signal.

Comparisons of the input and output signals:

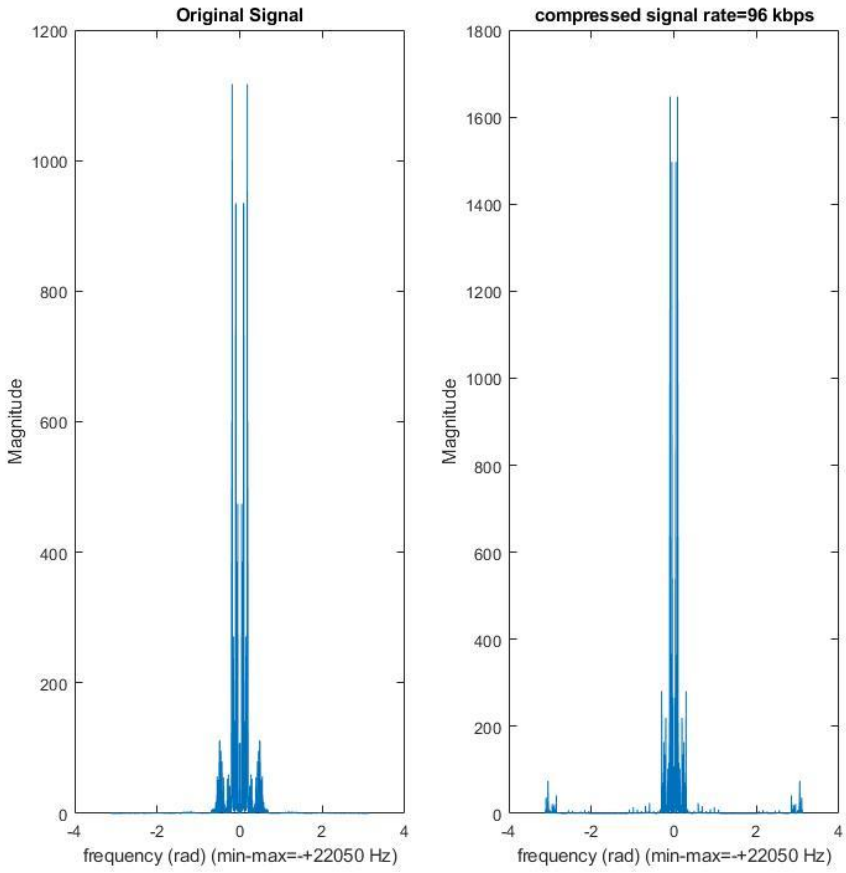

Figure 50: Magnitude responses (FFT's) of the original (.wav) and compressed (bitrate=96 kbps) (.mp3) signals

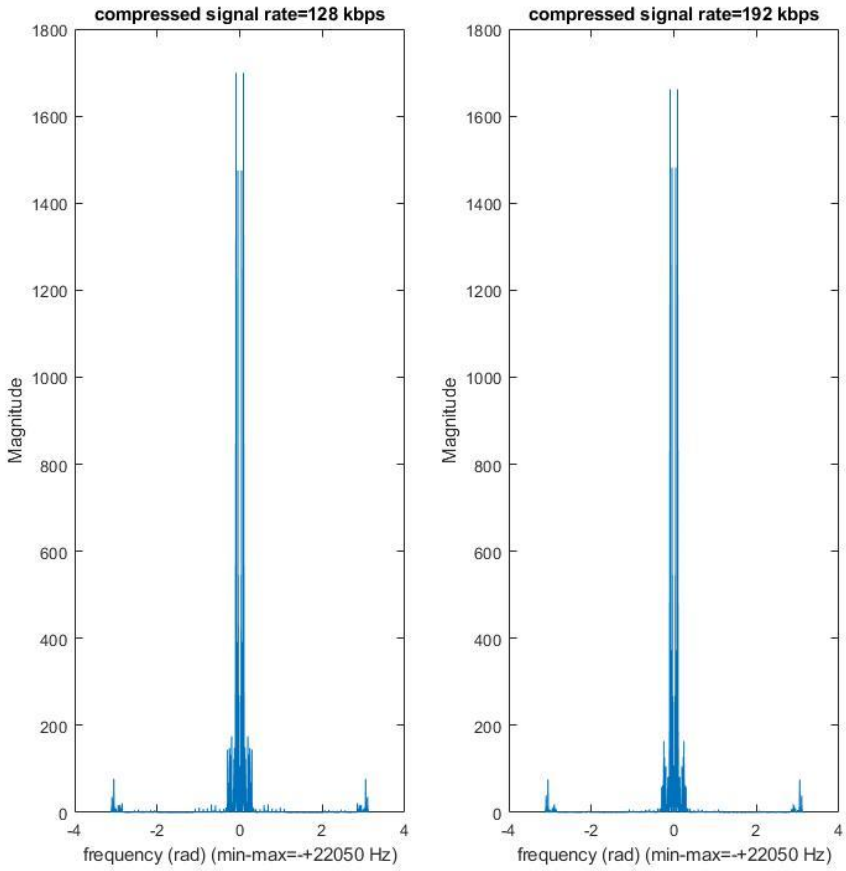

Figure 51: Magnitude responses (FFT's) of the compressed signals with bitrates of 128 and 192 kbps respectively

Comments:

- As it can also be seen in the magnitude responses of the compressed signals**, as the rate of bits per second increase in the compression algorithm, frequency characteristics of the compressed signal resembles more to the original one**. This is expected because as the rate of bits per second increase, number of bits used for quantization increase for samples which results in less quantization error.
- MPEG audio encoding also **adds high frequency noise** to the signal during compression. However, most of this noise is not audible and does not effect the hearing experience of the listener. However, **several high frequency peaks gets into the audible spectrum and they are audible.**

  When we listen the resulting signals, the quality enhancement can be observed. In the 96kbps case, several additional high frequency components, which can be observed in the FFT's above, can be heard as a robotic distortion. Magnitudes of these components decrease as we increase the kbps and mentioned effects almost vanishes when the rate is 192kbps.
- When compared with the previous methods, (Downsampling, time domain quantization, transform domain thresholding) this approach provides the best results. It applies frequency

domain threshold in a better manner than transform coding by taking hearing characteristics of human ear into account instead of crudely eliminating components according to their magnitudes. Also, it does reduction of the total number of bits used to store the data in a much more efficient way according to the resolution of human hearing system by taking the frequency band of the components into account.