Kutay Uğurlu - 2232841
Arda Yazıcı - 2232981

EE430 Term Project Part 1

In the first part of our term project, we have developed an algorithm for computing and then displaying the Short-Time Fourier Transform (STFT) of a time-domain signal. STFT is some kind of a Fourier Transform which is utilized to specify the sinusoidal frequency content of local parts of signals.

There are three separate parts in this project. These parts are Data Acquisition, Data Generation and Spectrogram, respectively. In this report document, the aforementioned parts are explained in a detailed fashion.

Data Acquisition:

Sound data from a microphone:

RECORD AUDIO:
The user can uncomment the below 4 lines of code to record their voice.

```
F_s = 44.1e3
recObj = audiorecorder(F_s,8,1);
recordblocking(recObj, total_length);
Signal = transpose(getaudiodata(recObj));
t = linspace(0,total_length,F_s*total_length);
```

Sound data from a file:

LOAD AUDIO DATA FROM FILE:
The user can uncomment the below 4 lines of load the audio data from a specified file path.

```
filename = 'handle.wav';
[y,F_s] = audioread(filename);
total_length = length(y)/F_s;
t = linspace(0,total_length,F_s*total_length);
Signal = y';
```

MATLAB function “audioread” cannot be input the sampling frequency as a parameter, instead sampling frequency is set to be the output of the function.

Data Generation:

There are several different MATLAB scripts that generate time-domain samples representing the required functions. The functions return both the signals and vectors for time axes. The vector that represents the time axis is generated by sampling the time axis of length total_length uniformly by the number of F_s*total_length, where the total_length represents the length of the signal in seconds and F_s represents the sampling frequency in Hz.

The explanation and the code for these functions are presented below as follows.

Sinusoidal signal:

```
function [signal, t] = get_cosine(total_length, F_s, frequency, amplitude, phase)
t = linspace(0,total_length,total_length*F_s);
signal = amplitude * cos(2*pi*frequency*t+phase);
end
```

Windowed sinusoidal:

```
function [signal, t] = get_linear_chirp(duration, F_s, frequency_init, amplitude, phase, bandwidth,
shift)
t = linspace(0,duration,duration*F_s);
t = t + shift; % Assumed user inputs t0 to produce s(t-t0)
signal = amplitude * cos(2*pi*(frequency_init*t + bandwidth/(2*duration)*t.^2) + phase);
end
```

Rectangle windowed linear chirp:

```
function [signal, t, w] = get_windowed_s(total_length, F_s, frequency, amplitude, phase,
window_name, starting_time, window_length)

%Built-in window uses sample number. To convert seconds to sample number we
%must use
T =  1 / F_s;
t = [0 : T : total_length];
t_w = [starting_time : T : starting_time + window_length];
w = transpose(window(str2func(window_name), window_length*F_s + 1)); %To maintain the
given sampling period, needed to consider the outermost samples
signal = amplitude * cos(2*pi*frequency*t + phase);
intersection = intersect(t,t_w);
idx_start_signal = find(t == intersection(1));
idx_end_signal = find(t == intersection(end));
idx_start_window = find(t_w == intersection(1));
idx_end_window = find(t_w == intersection(end));
signal = signal(idx_start_signal:idx_end_signal) .* w(idx_start_window:idx_end_window);
w = w(idx_start_window:idx_end_window);
t = t(idx_start_signal:idx_end_signal);
end
```

Square wave:

```
function [signal, t] = get_square(total_length, F_s, frequency, amplitude, phase,
duty_cycle_in_percent)
t = linspace(0,total_length,total_length*F_s);
signal = amplitude * square(2*pi*frequency*t+phase,duty_cycle_in_percent);
end
```

Sawtooth wave:

```
function [signal, t] = get_sawtooth(total_length, F_s, frequency, amplitude, phase, width)
t = linspace(0,total_length,total_length*F_s);
signal = amplitude * sawtooth(2*pi*frequency*t+phase,width);
end
```

Signal involving multiple components:

```
function [signal, t] = get_multiple(number_of_components,total_length, F_s, frequency, amplitude,
phase)
t = linspace(0,total_length,total_length*F_s);
signal = 0;
for i = 1:number_of_components
    temp = get_cosine(total_length,F_s,frequency(i),amplitude(i),phase(i));
    signal = signal + temp;
end
end
```

The Short-Time Fourier transform (STFT):

In this part of the term project, to realize the STFT operation, we first determined the number of time slices for the STFT window to iterate through. Until the STFT window attains the last slice, the function takes the FFT of the element-wise product of that slice of the signal with the window. When the STFT window arrives at the end of the signal, we have developed our MATLAB code such that it takes the FFT of the product of the window with the last window-length samples of the function.

```
function [STFT, number_of_slices] = st_ft(F_s ,signal, window, stride_in_sec)
window = transpose(window);
window_length_in_sample = length(window);
stride_in_sample = stride_in_sec * F_s;
N  = length(signal);
number_of_slices = ceil((N - window_length_in_sample)/stride_in_sample) + 1; % total number of
slides including semi-full one
STFT = zeros(window_length_in_sample, number_of_slices);
for i = 1:number_of_slices-1
    STFT(:,i) = transpose(fftshift(fft(signal((i-1)*stride_in_sample+1:(i-1)*
stride_in_sample+window_length_in_sample).*window)));
end
STFT(:,number_of_slices) = transpose(fftshift(fft(signal(end-window_length_in_sample+1:end))));
end
```

Up to this point, we have demonstrated the MATLAB scripts that generate the required signals and we have also shown the script for the STFT operation. In the following part of the report, we will be providing the information to the user so that he/she can generate the required signals, read the parts of the user's audio files or record the sound via microphone. In addition, the user will also be capable of investigating the signal properties via time plots and spectrograms.

Now, we will demonstrate the plots of the generated signals as follows.

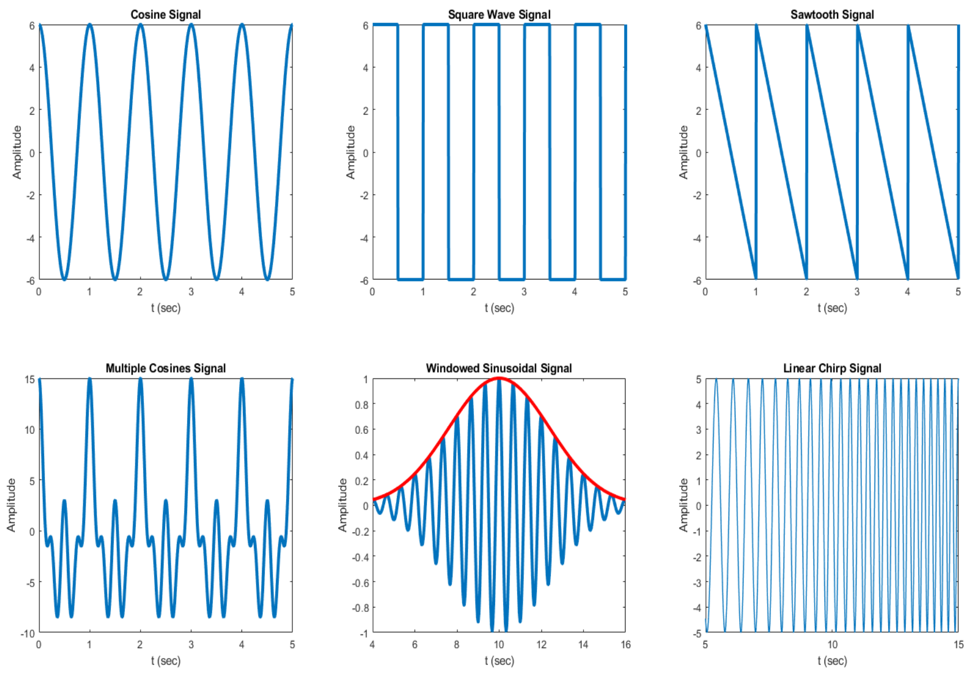

Figure 1 The plots of the Generated Signals

Spectrogram:

In the previous part, we have fulfilled the STFT operation for the generated signals. The output of the above STFT function is a matrix whose columns store the FFT values of the element-wise product of the window and the respective time slice. In order to determine the frequency intervals where we want to compute the frequency content, we have specified an input parameter for the frequency range. The Spectrogram function iterates through the columns of the STFT matrix and takes the sum of the absolute square of the frequency spectrum slices generated based on the frequency range input. The symmetry around the 0 Hz axis is due to the fact that the magnitudes of FFTs of the real signals, that we generated, are even. Besides, it needs to be mentioned that we could have truncated the rows corresponding negative frequencies, but we have decided to keep it as it is.

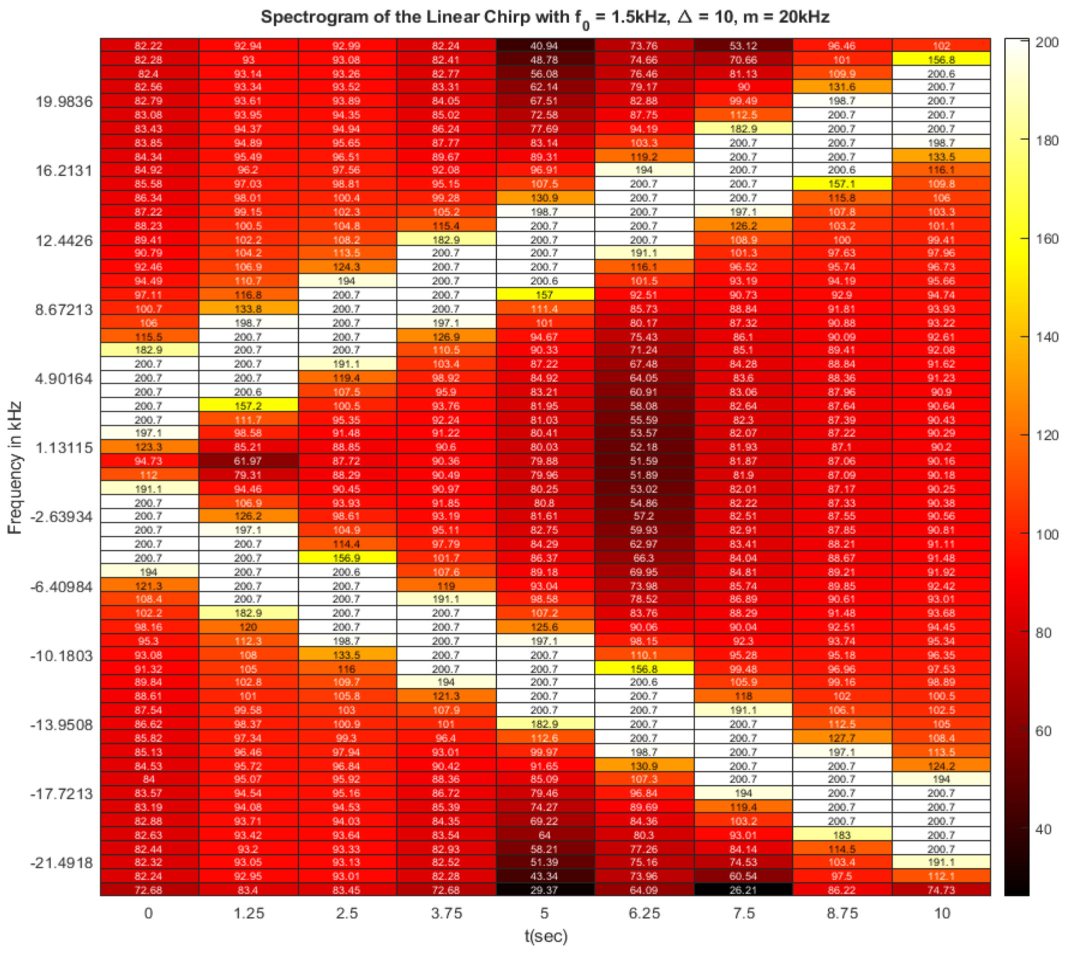

Figure 2 The Spectrogram of Linear Chirp

The above spectrogram in Figure 2 was obtained by the linear chirp function that we have generated with the parameters specified in the title of the figure. As we have expected, since the frequency of linear chirp is a linear function of time, we have observed a linearly increasing fashion in the power spectrum.

In addition to this, it is expected to investigate the portion of the spectrum that carries the highest power. For instance, while we observe that the interval between 2.64 kHz – 6.5 kHz carries the highest power value for the time t=1.25 seconds, we notice that the interval between 15 kHz – 19 kHz carries the highest power value for the time t=8.75 seconds.

Guidance for The User:

```
USER SET TESTS
In this section of the script, the user can modify the code given below to create signals of specified parameters, later
these signals will be summed up and shown in time and spectrogram plots.
To eliminate a signal, the user should set its amplitude to 0. That signal is not going to be shown in the output plots.

Note: The comments regarding each line of the below code are colored in red.

F_s = 15e3;  Sampling Frequency
total_length = 15;  Total length of the resultant signal in seconds starting from 0

cosine_signal_count = 2; The number of components of the resultant signal
cosine_amplitudes = [5 10];  Amplitudes for cosine signals, 1-length vector can be used to set a single cosine
cosine_frequencies = [1000 5000] ;  Frequencies for cosine signals, 1-length vector can be used to set a single cosine
cosine_phases = [0 0];  Phases for cosine signals, 1-length vector can be used to set a single cosine

square_amplitude = 1;  Amplitude for square wave signal
square_frequency = 1.5e3 ;  Frequency for square wave signal
square_phase = 0;  Phase for square wave signal
square_duty_cycle = 80;  Duty cycle for square wave in percent

sawtooth_amplitude = 1;  Amplitude for sawtooth wave signal
sawtooth_frequency = 2e3 ;  Frequency for sawtooth wave signal
sawtooth_phase = 0;  Phase for sawtooth wave signal
sawtooth_width = 0;  Width for sawtooth wave signal

windowed_s_amplitude = 5;  Amplitude for windowed_s signal
windowed_s_frequency = 1;  Frequency for windowed_s signal
windowed_s_phase = 0;  Phase for windowed_s wave signal
windowed_s_window_name = '@rectwin';  Window name for windowed_s signal in type string
windowed_s_window_length = 3;  Window length in sample
windowed_s_starting_time = 12;  Starting time of the first sample relative to zero initial time

linear_chirp_amplitude = 1;  Amplitude for linear_chirp signal
linear_chirp_frequency_init = 1;  Frequency init for linear_chirp signal
linear_chirp_phase = 0;  Phase for linear_chirp wave signal
linear_chirp_bandwidth = 5e3;  Bandwidth for linear_chirp signal
```

If the user does not desire any kind of signal that is demonstrated in the above script, all they need to do is to set that specified signal's amplitude to 0.

In addition, if one wants to have a single cosine, the only thing that needs to be done is to set the number of components to 1, and then set the other relevant input parameters using a vector containing one element.

The next step that the user should complete is to set the total length and the sampling frequency of the resultant signal. Afterwards, the user should determine the intervals of the individual components of the resultant signal.

```
SET INTERVALS FOR EACH SIGNAL

cos_start = 0 ; cos_end = 3;
square_start = 3; square_end = 6;
sawtooth_start = 6; sawtooth_end = 9;
% Windowed_s start and end points are specified with window_length and window_starting_time parameters.
linear_chirp_start = 9; linear_chirp_end = 12;

[Cosines, t_Cosines] = get_multiple(cosine_signal_count, cos_end-cos_start,F_s,cosine_frequencies,
cosine_amplitudes,cosine_phases);
[Square, t_Square] = get_square(square_end-square_start,F_s,square_frequency,square_amplitude,
square_phase,square_duty_cycle);
[Sawtooth, t_Sawtooth] = get_sawtooth(sawtooth_end-sawtooth_start,F_s,sawtooth_frequency, sawtooth_amplitude,
sawtooth_phase,sawtooth_width);
[Windowed_s, t_Windowed_s, ~] = get_windowed_s(windowed_s_window_length, F_s,windowed_s_frequency,
windowed_s_amplitude, windowed_s_phase,windowed_s_window_name,0,windowed_s_window_length);
[Linear_chirp, t_Chirp] = get_linear_chirp(linear_chirp_end-linear_chirp_start,F_s,linear_chirp_frequency_init,
linear_chirp_amplitude,linear_chirp_phase, linear_chirp_bandwidth,linear_chirp_start);
```

Once these fundamental parameters of the individual signals and their relative positions in the time axis are set, the user can conveniently run the script provided for the user. Besides, the above code is yielded with the name “User_Set_Run.m” in the codes we have submitted.

We have already prepared the demonstration using the above script. Now, we will be showing the sample output generated by the above script.

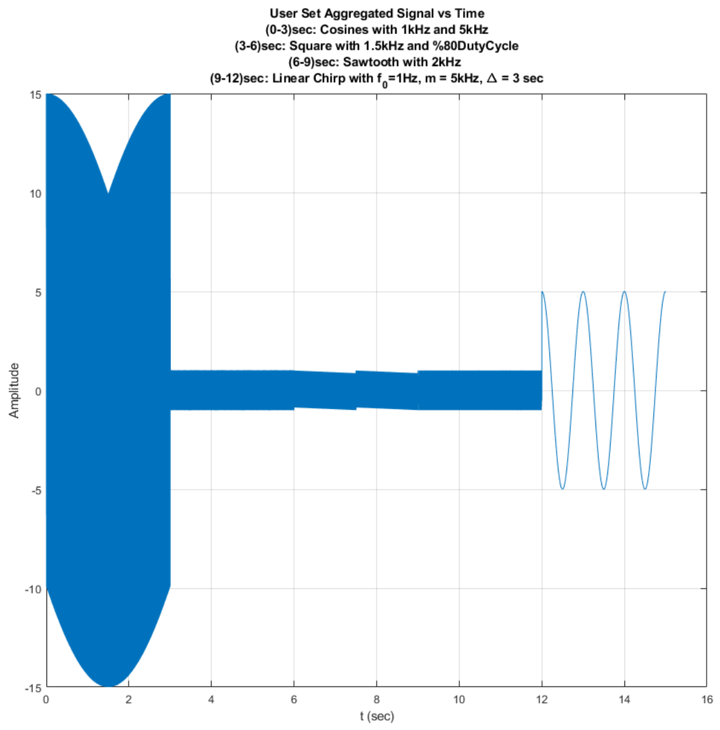

Figure 3 The Time Plot of the Sample Signal

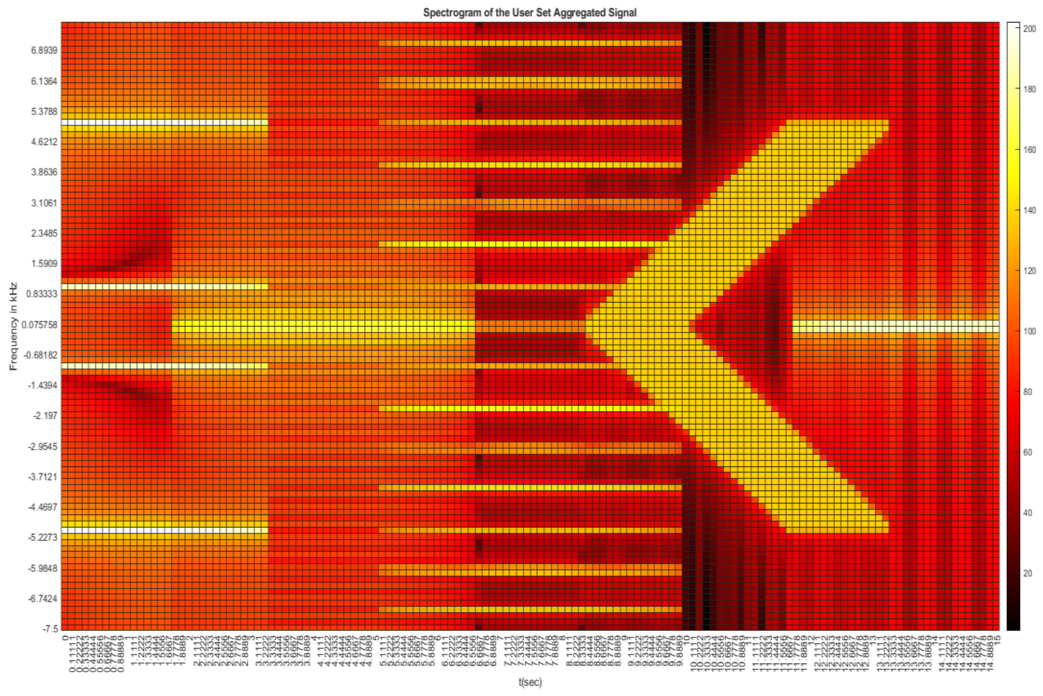

Figure 4 The Spectrogram for the Sample Signal

As it can be observed from the first quarter of the spectrogram, the frequency content that carries the highest power accumulated around 1 kHz and 5 kHz. This is an expected result for the sum of two cosine signals carrying the aforesaid frequencies. In the segments containing the FFT of the sawtooth signal, we have investigated almost uniformly spaced multiples of the fundamental frequency. In the time interval where the samples of window signal coincide with samples from both sawtooth and linear chirp signals, we have observed the frequency content from both signals.

Discussion:

In this part of the report, we want to address the questions that are asked in the last part of our project description document. The following figures represent the spectrograms of a signal whose first 8 seconds is a linear chirp function and the rest is a multiple cosine signal with 4 kHz and 20 kHz.

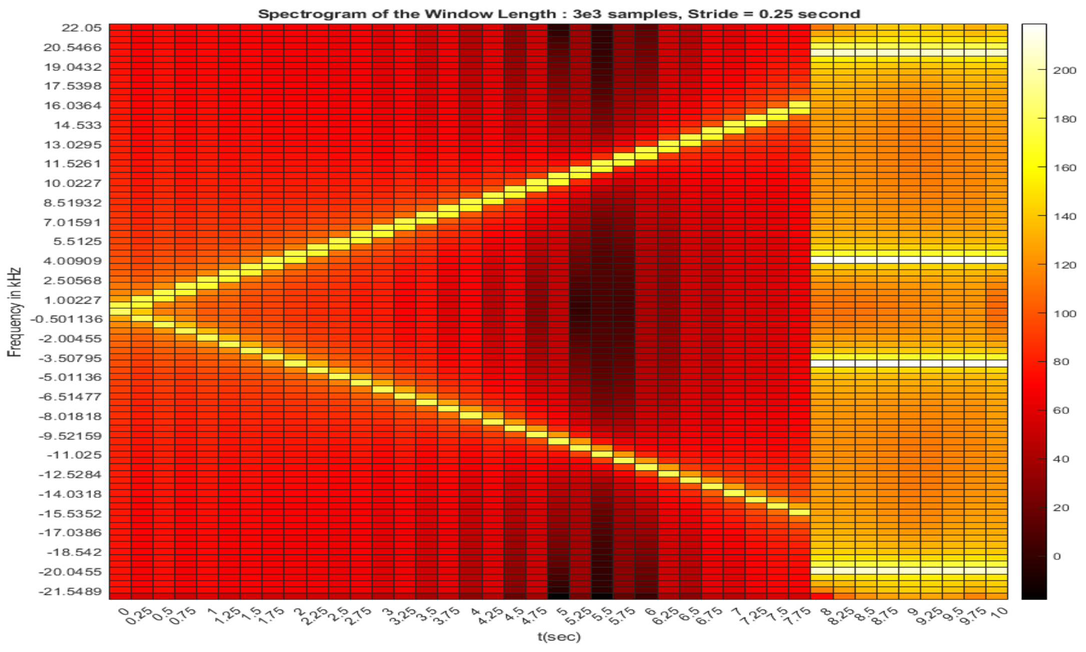

Figure 5 Spectrogram with 3000 samples and 0.25 sec stride

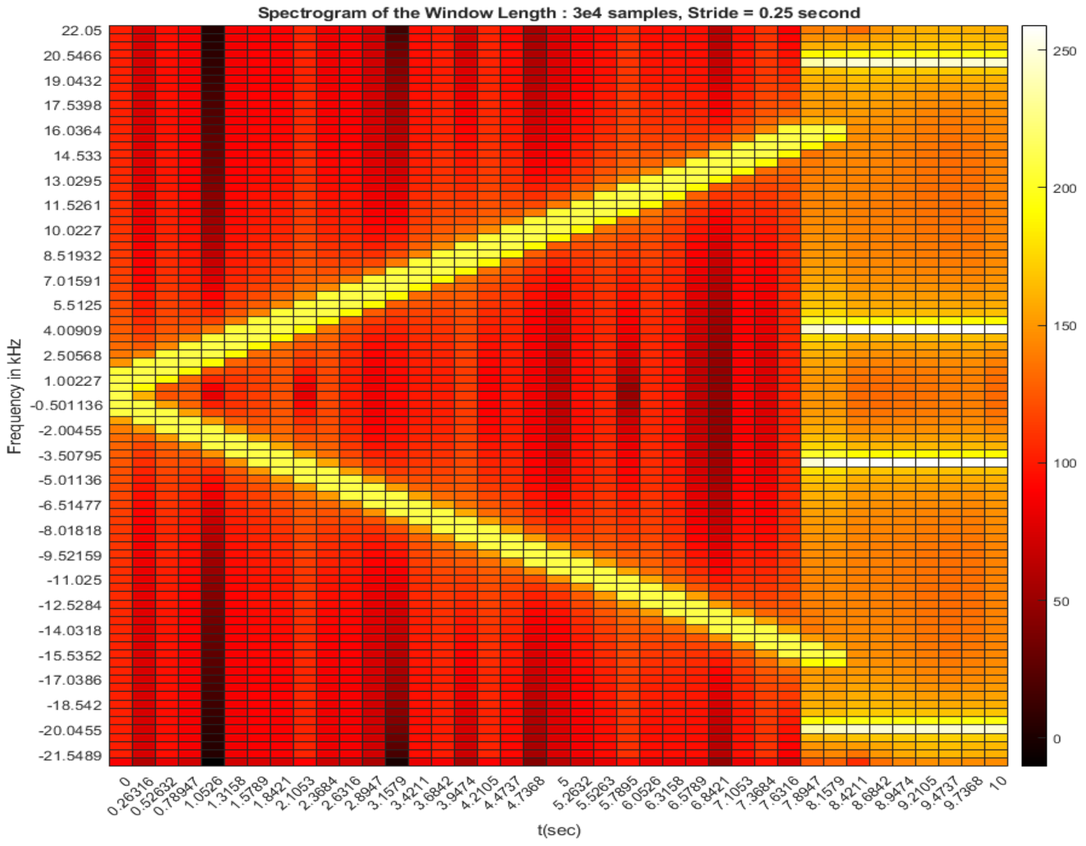

Figure 6 Spectrogram with 30000 samples and 0.25 sec stride

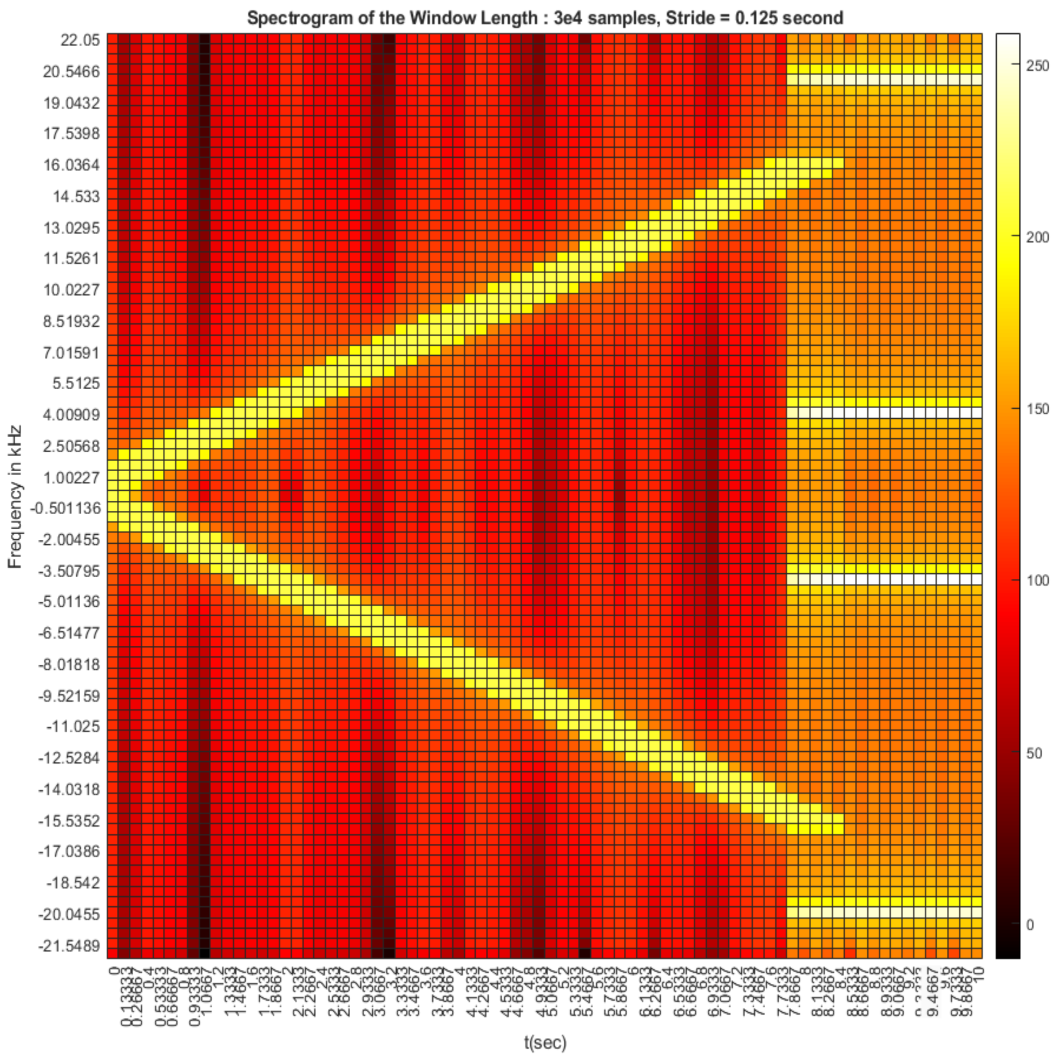

Figure 7 Spectrogram with 30000 samples and 0.125 sec stride

Now, we will make comparison between Figures 5 and 6 to address how window length affects the STFT, hence the spectrogram. Since a window higher in length can capture more time content from a given signal, the STFT in figure 6 was able to capture the increasing frequency of linear chirp, and this resulted in a wider band of high-power carrying spectrum for a given time instance. We can investigate this effect from the spectrograms in Figures 5 and 6 by looking at the width of the edges of V-shaped high-power content. Besides, from the comparison between Figures 6 and 7 having different strides, it can be stated that with smaller shifts, we are able to investigate a higher number of slices. Moreover, there is the following trade-off such that smaller stride is better in terms of resolution; however, it is worse in the sense that it requires more arithmetic operations and hence more computational power. On the other hand, there is also an upper bound for this shift value. If the shift value exceeds the length of the window, some samples do not make any contribution to the STFT sum. Therefore, some information embedded in the signal is lost.

Before we conclude our report, we want to command on the effect of window type on the spectrogram by comparing three different window types. The first one is the regular rectangular window.

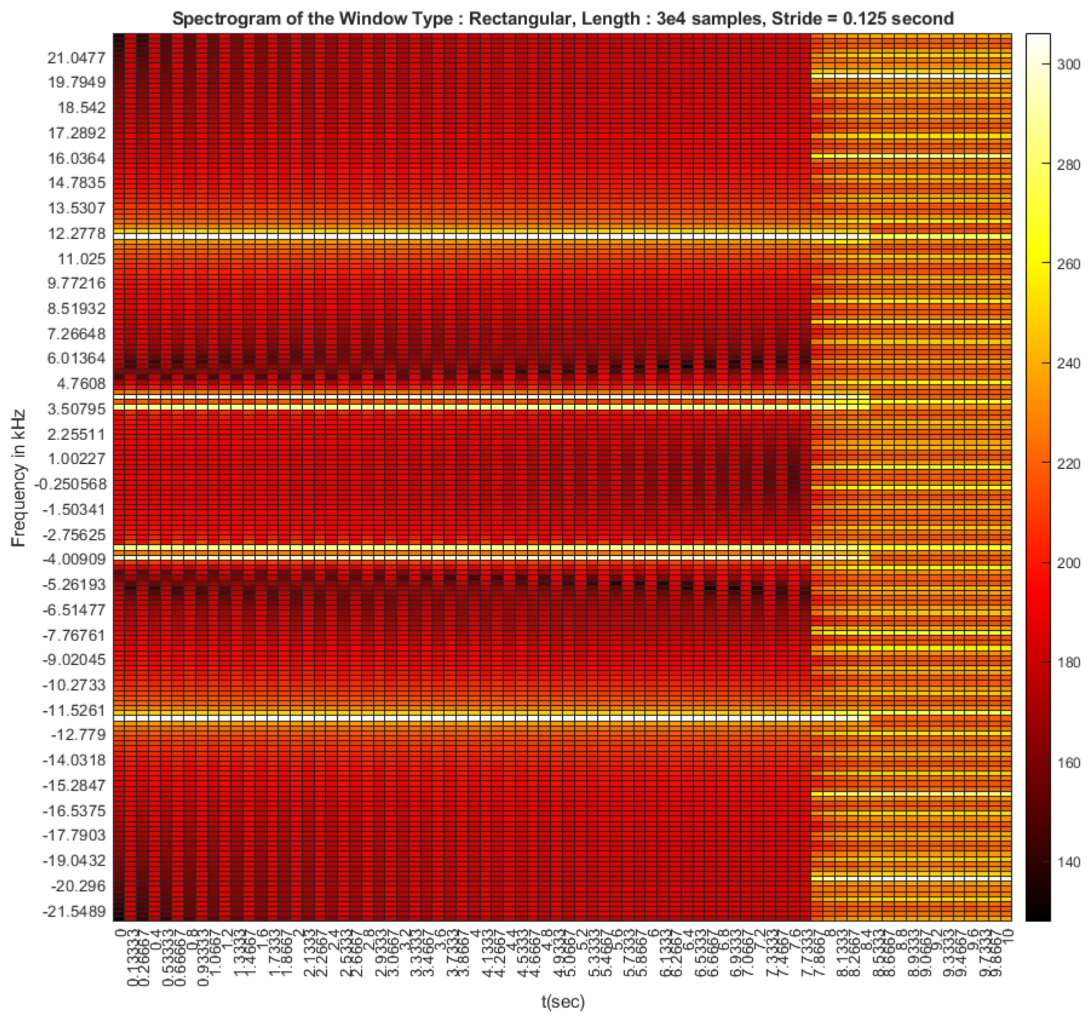

Figure 8 The Spectrogram with Rectangular Window

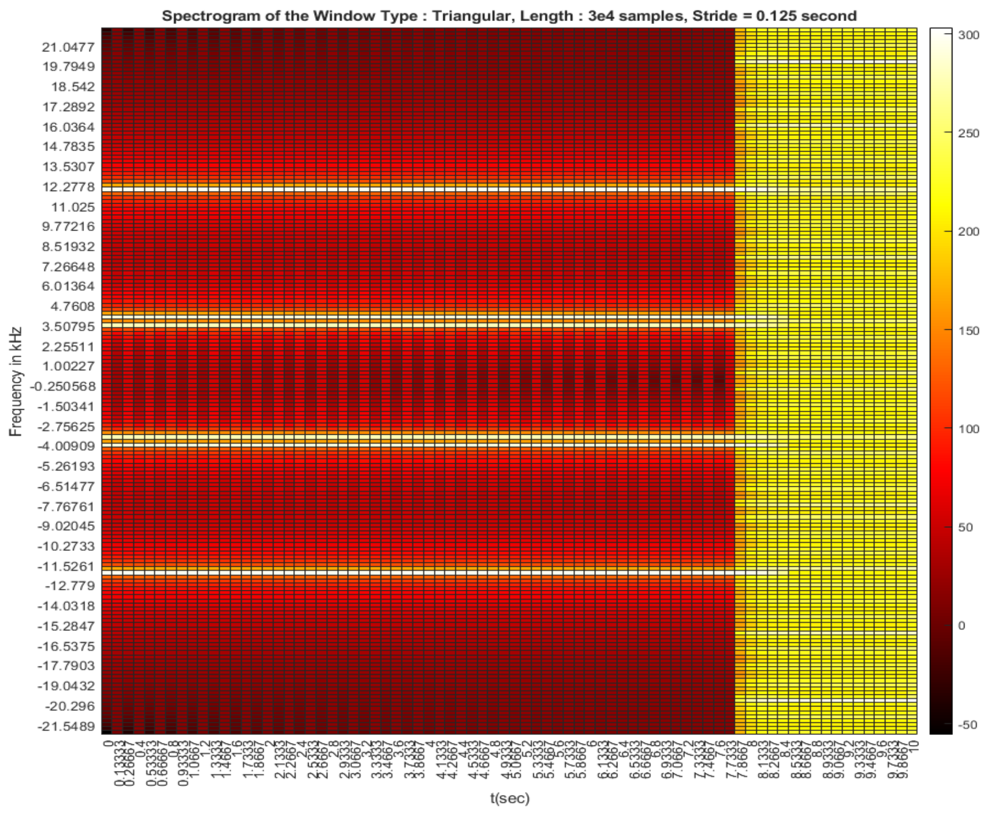

Figure 9 The Spectrogram with Triangular Window

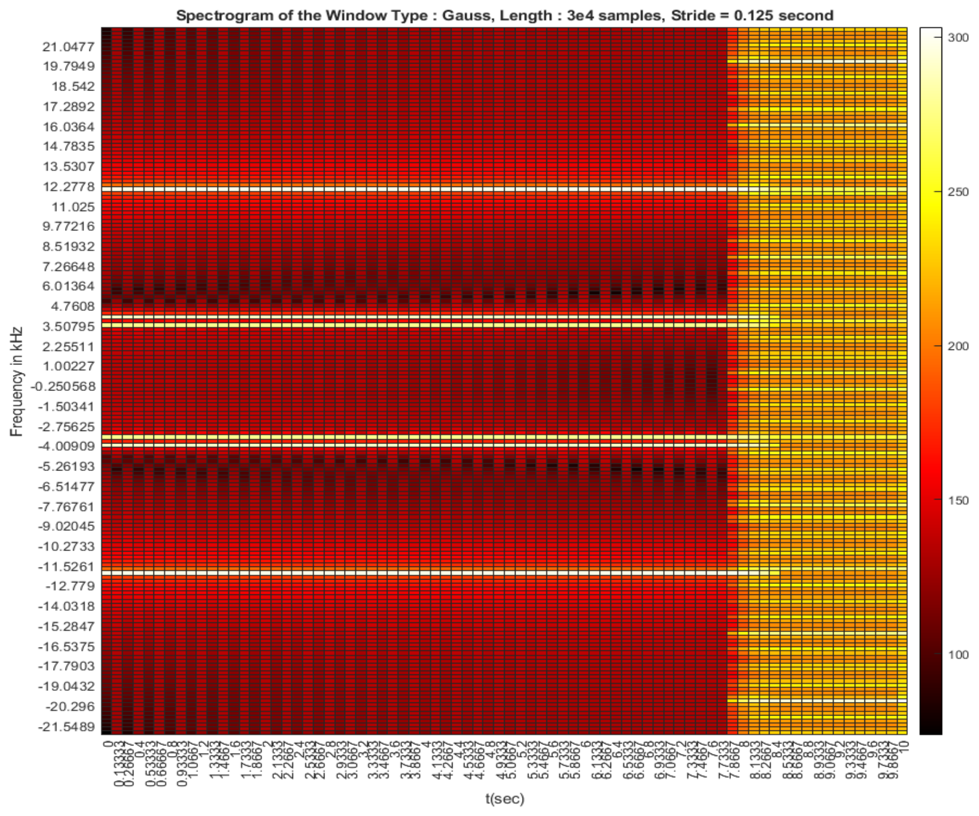

Figure 10 The Spectrogram with Gauss Window

It is known that, unlike the rectangular window, the Gaussian and the triangular windows introduce new frequency components on the time slices that the STFT window selects. Due to this fact, high power frequency content seems to be distributed slightly more homogenously around the frequency band including high power values. In Figure 9, we have noticed that the power spectrum is distributed more homogenously in the approximately last two seconds of the given time interval of the spectrogram.

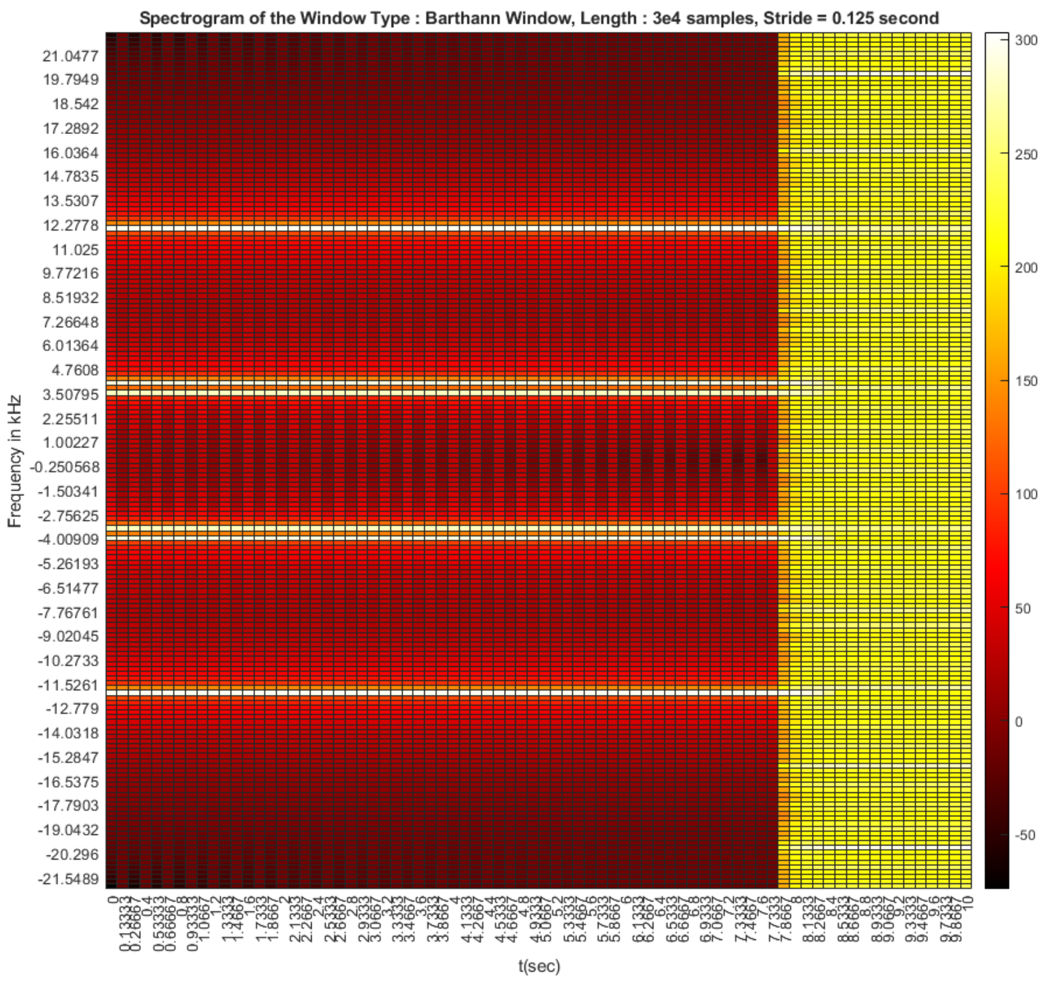

Figure 11 The Spectrogram with Bartlett-Hann Window

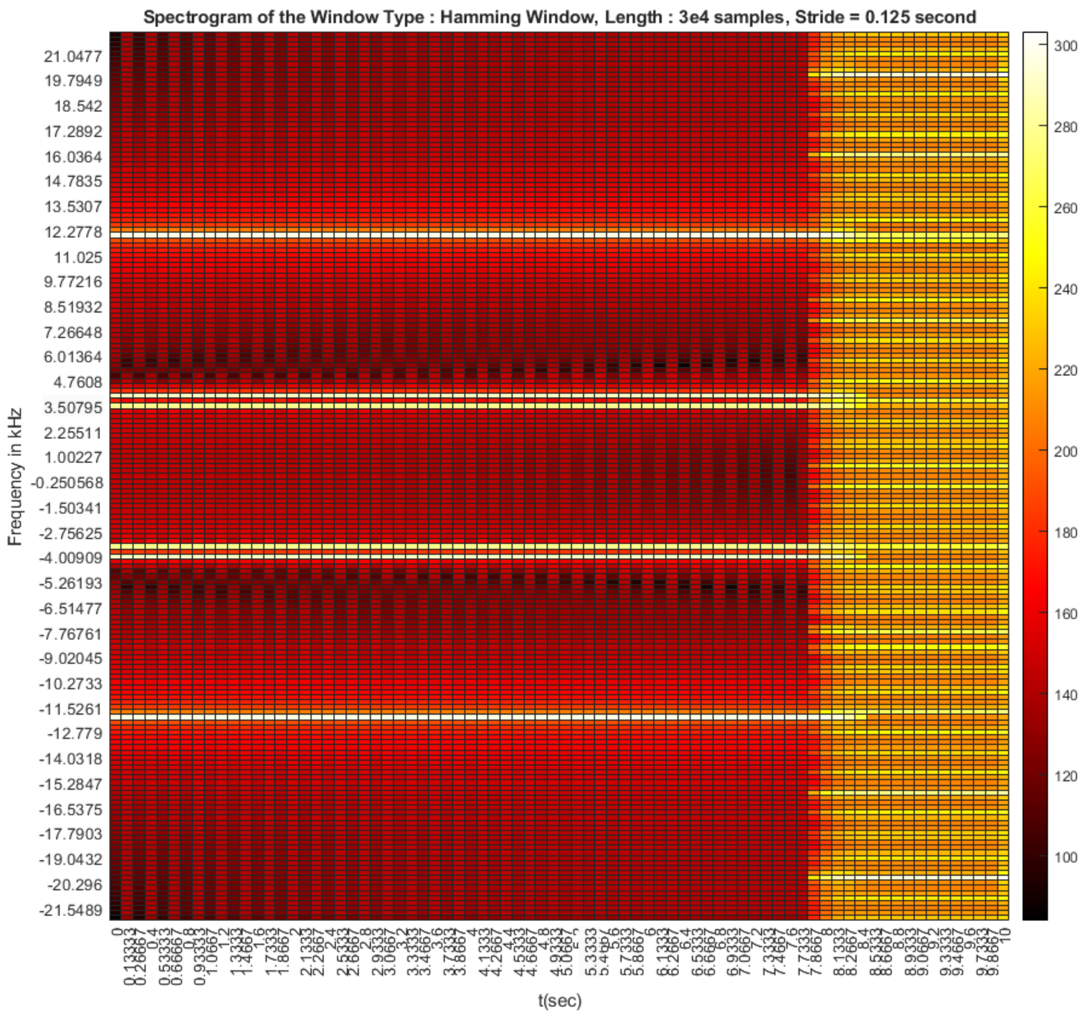

Figure 12 The Spectrogram with Hamming Window

When we compare the spectrograms with Bartlett-Hann and Hamming windows, we have noticed that the former one was relatively more successful in terms of detecting the frequencies being close to DC frequencies with the higher temporal resolution, compared to the spectrogram with Hamming window. It can also be observed that the STFT with a Hamming window was able to seize a similar temporal resolution when the window starts capturing the samples from the square wave.

In addition, when we compare Triangular and Hamming window with the remaining ones, the following can be mentioned. For the cases when Rectangular, Gaussian and Hamming windows are used in STFT, we investigated that these operations are more capable of capturing the multiples of the fundamental frequency of the square wave. However, in the other operations, we have observed a more homogenous distribution of the frequency content in the last two seconds of the signal where the square wave lies.

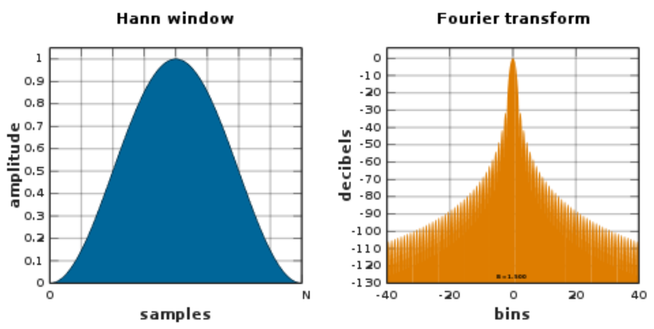

Figure 13 The Hann Window in Time and Frequency Domains

The elementwise multiplication of window and signal in STFT operation results in convolving their spectra in frequency domains. The reason why we observed the case that was mentioned on the previous page is the fact that window functions have different spectra.

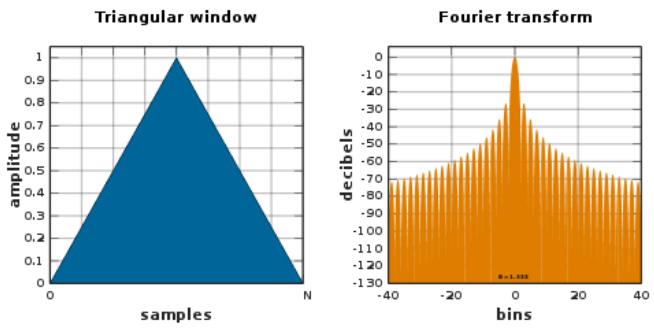

Figure 14 The Triangular Window in Time and Frequency Domains

As can be seen from Figures 13 and 14, the spectra shown in these figures are more pervasive which smooths the absolute magnitude of FFT coefficients resulting in a more homogenously distributed spectral power in the last two seconds of the signal. The reason why we do not see such behavior in the cases that other window types are utilized is the fact that the frequency content of these windows' spectra is concentrated around zero Hz.

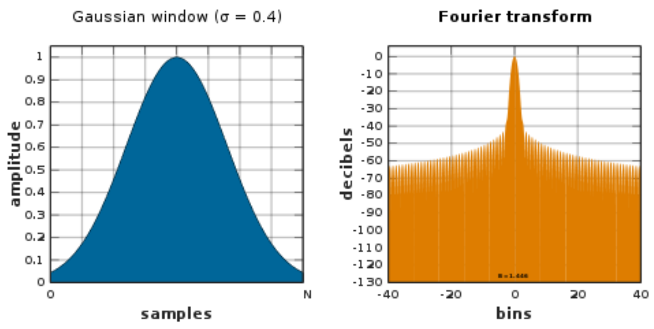

Figure 15 The Gaussian Window in Time and Frequency Domains

Conclusion:

In the first part of the term project, we have coded up signal generator functions, a MATLAB script that computes Short Time Fourier Transform of a given signal, and another MATLAB script that generates the spectrogram for the generated signals. This procedure helps the user understand the frequency content of a time-domain signal for different time intervals in a pictorial fashion. We have also coded up the script named "User_Set_Run.m" to ease the usage of the system we developed. Besides, the user manual is presented in this project report, including the instructions for reading audio data from a file and recording audio data via a microphone.