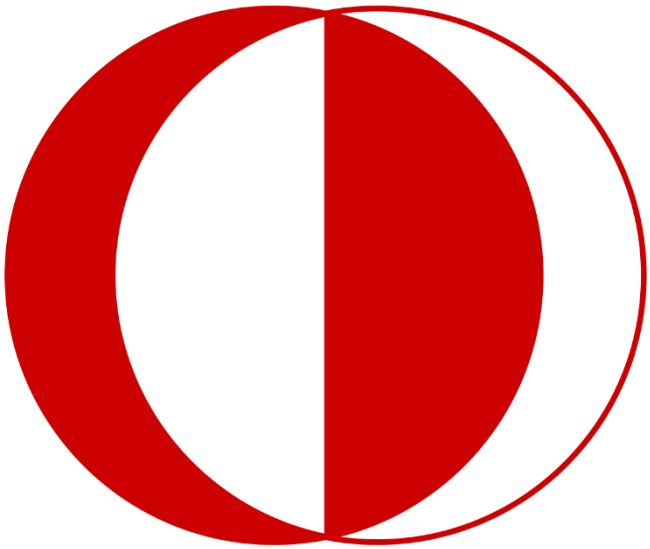


MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL AND ELECTRONICS ENGINEERING

EE430 Digital Signal Processing, 2020-2021 Fall

Term Project

Part I

Halil Çağrı Bilgi

2231470

Project Group 65

Table of Contents

Introduction

In this term project, I have had a chance to implement what I have learned in EE 430 DSP course. However, since we didn't cover the STFT topic in class I read the related chapters from our textbook as recommended in the project description file. After the readings, I implemented the spectrogram function by myself. One can see the myspectrogram function at the end of the report. Since the main focus of the project is the spectrogram function I will only provide the code for that function.

Throughout this report, I will be telling how the subsystems (data acquisition, data generation, and plots) of my app works, and also I will guide the reader through how to use the GUI.

Data Acquisition

In this part, there are 2 different types of data acquisition. The first one is the sound data recorded with a microphone and the second one is any sound data with '.mp3' and '.wav' extensions. First we will start with sound data from a microphone.

1. Sound data from a microphone

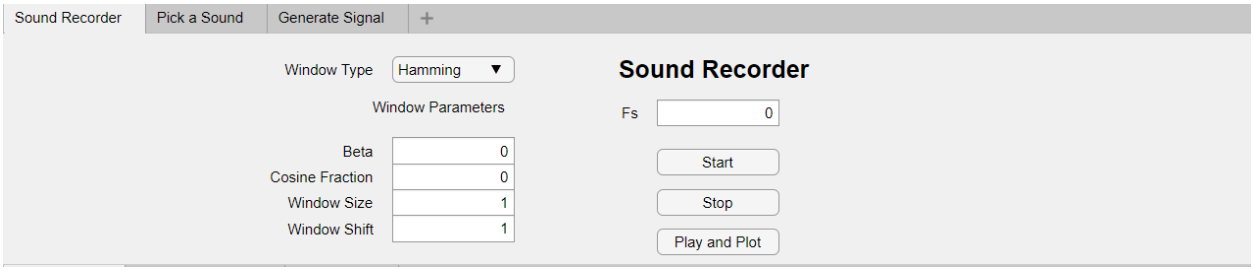


Figure 1

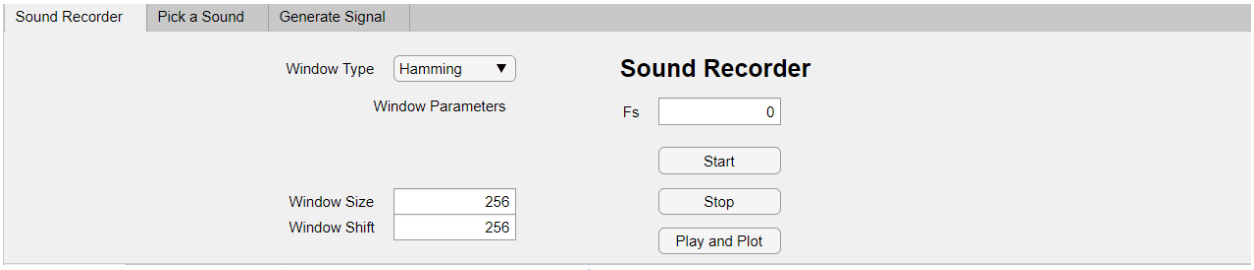


Figure 2

‘Figure 1‘ above is the view of my app on the app designer page and the ‘Figure 2‘ is the view when the app runs. I included both figures because depending on the Window Type choice, irrelevant parameters are hidden from the user. This means only the parameters that can be seen will be changed.

Before the recording, all the parameters that can be seen on the screen must be provided by the user. The user can change the Window Type that will be used in the spectrogram calculations and also the sampling frequency must be provided.

Then, when the user press the **Start** button a message box will pop up and inform the user that the recording has started. I used the `audiorecorder` object and it’s functions in this part. In the same way, when the user press the **Stop** button another message box will pop up and inform the user that the recording has stopped. The user can listen to the recorded sound and see the time, frequency, and spectrogram plots inside the GUI.

2. Sound data from a file

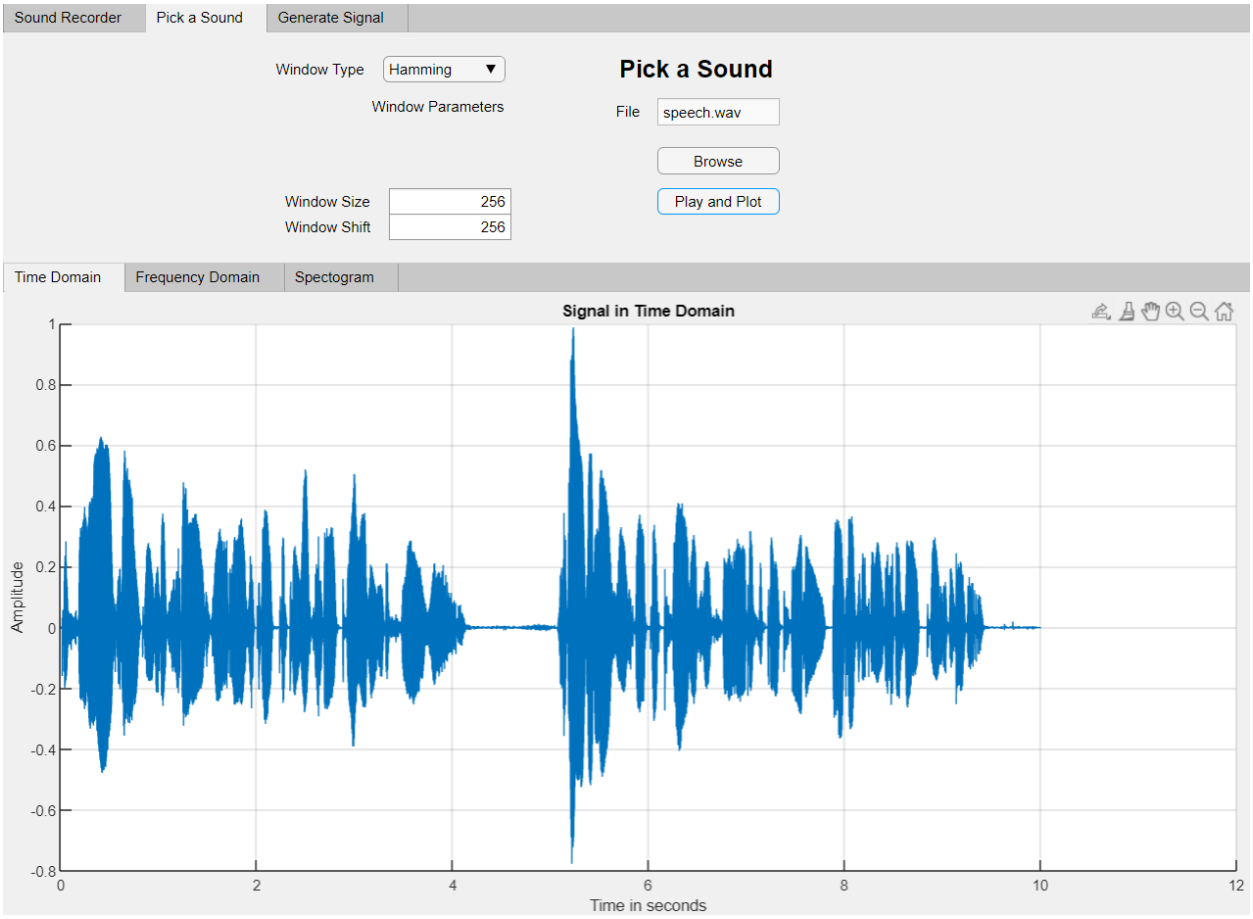


Figure 3

One can see the sound data from a file interface in the above figure. As in the recording part, the irrelevant parameters are hidden from the user.

After filling in the required parameters the user has to press the **Browse** button. This button will open an interface that allows the user to choose a sound file with a ‘.mp3‘ or ‘.wav‘ extension. Then, pressing the Play and Plot button will play the chosen sound file and also it will provide time, frequency, and spectrogram plots of that sound file. Each plot can be seen in the separate tabs. The user can also see the effects of different window types by changing the window type after loading the file and pressing the **Play and Plot** button again.

I have used the `audioread` function in this part. That function allowed me to retrieve the sampling frequency information while loading the sound file.

Data Generation

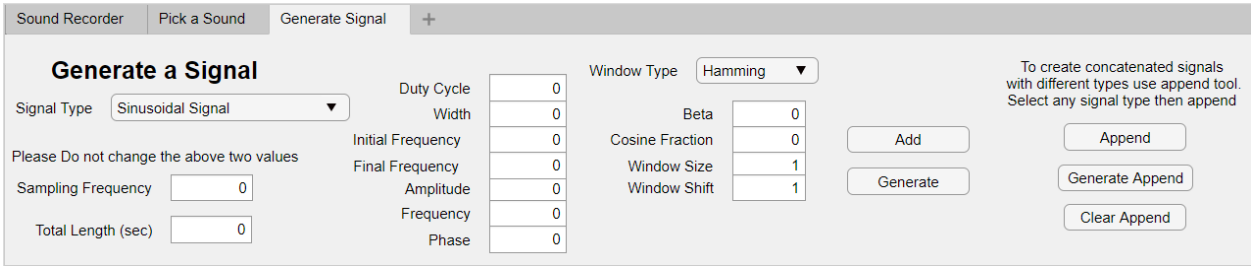


Figure 4

In the above figure the reader can see the data generation panel. There are a lot of parameters shown here but as in the previous parts when app is running only the related parameters are shown depending on the chosen Signal and the Window Type.

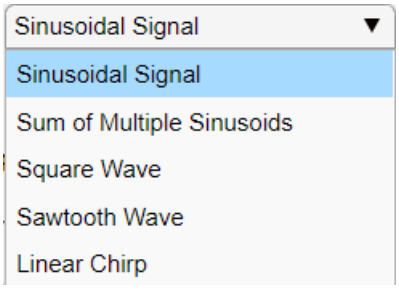


Figure 5

Signal Types that can be generated

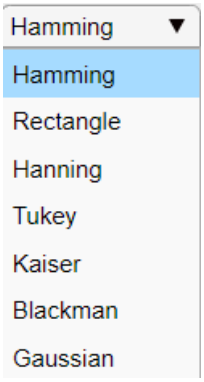


Figure 6

Window Types that can be generated

In the above two figures, one can see the different signal types and the window types that can be generated using the GUI. I will tell how each type of signal is generated.

As I said, the user can generate 6 different types of signals but in figure 5 only the five of them are shown. Because the 6th type of signal is generated by concatenating any combination of the 5 different signal types shown in Figure 5.

- **Sinusoidal Signal**

Figure 7

- **Square Wave**

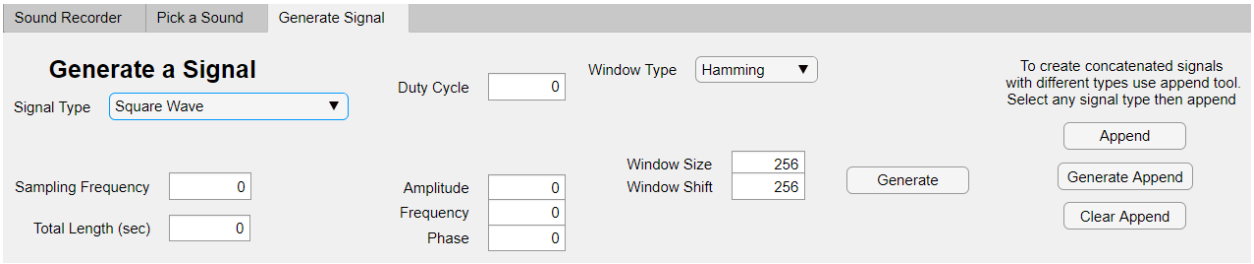


Figure 8

- **Sawtooth Wave**

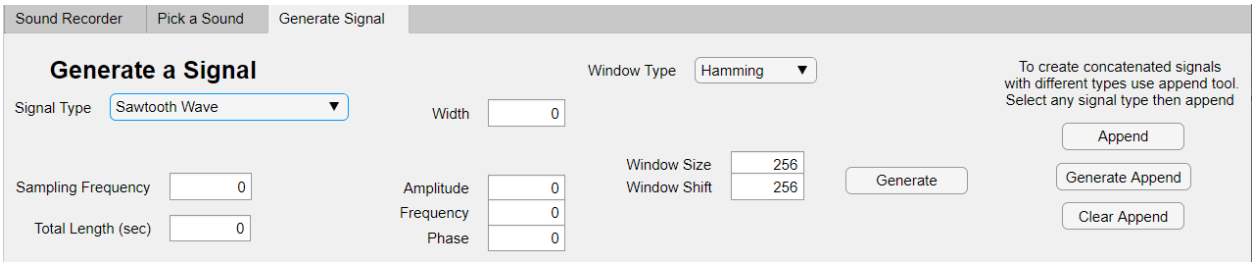


Figure 9

- **Linear Chirp**

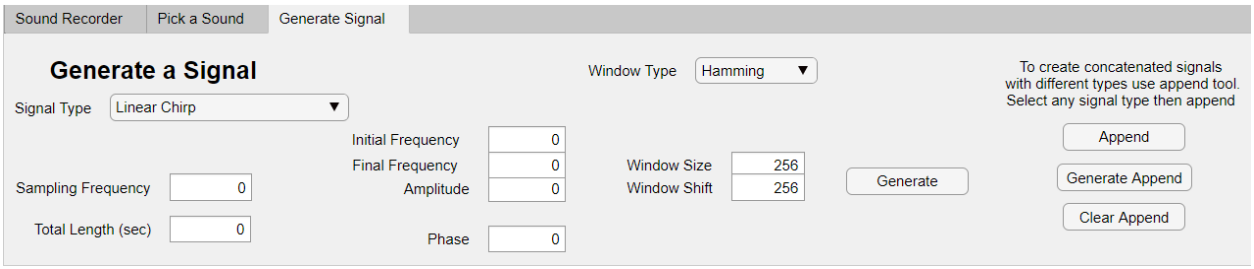


Figure 10

In the Figures 7,8,9 and 10 we can see the panel for each type of signal and the related parameters that has to be filled. After filling in the parameters, the user has to press the **Generate** button. The generated signals will be shown in time, frequency and spectrogram plots. The generater functions of each signal can be found in the .mlapp file as a private function.

- **Sum of Multiple Sinusoids**

Figure 11

In order to generate the sum of sinusoidal signals where each of the sinusoids could have different amplitude, frequency, or phase, the user must fill the Sampling Frequency, Total Length, Amplitude, Frequency, and Phase parameters. Then, the user has to press the **Add** button. The generated sinusoid is stored in the background. After the first sinusoid that is added, the Sampling Frequency and the Total length of the remaining signals shouldn't be changed but Amplitude, frequency, and phase could be changed. So after stacking up different sinusoids to generate the plots user must be press to **Generate** button. Time, frequency, and spectrogram plots will be generated.

- **Concatenated Signals**

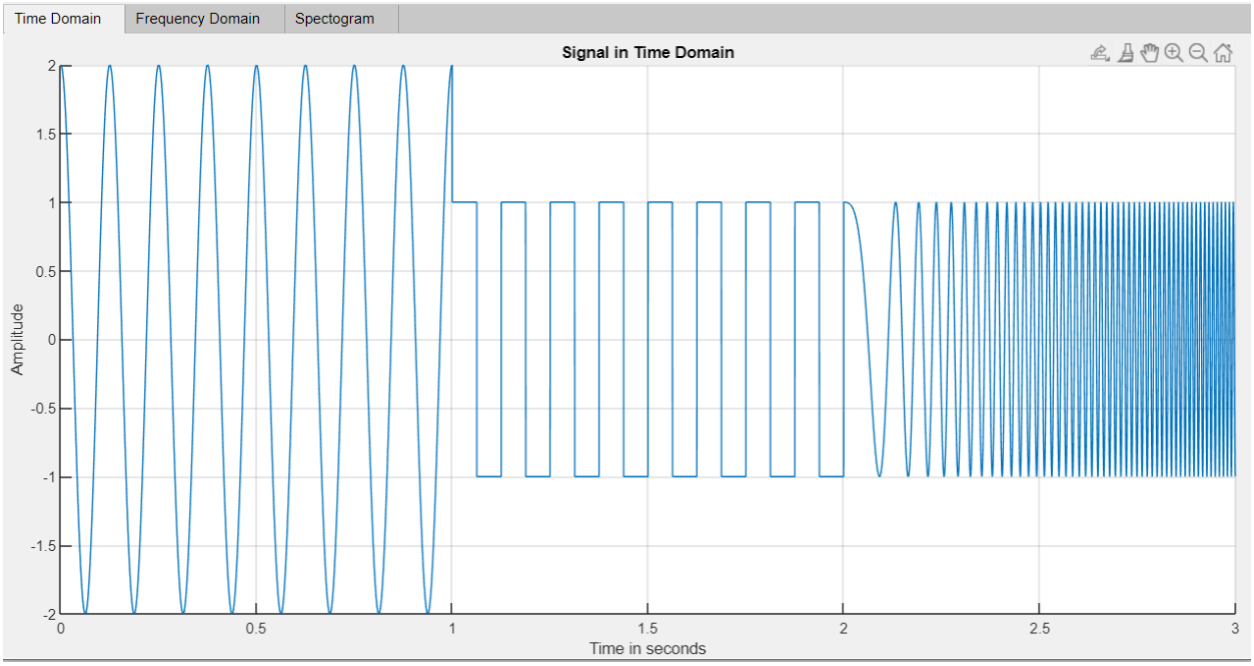


Figure 11

To generate concatenated the user can select any type of signal from the drop-down menu. The related parameters for selected signals have to be filled and then the user has to press the **Append** button. The selected signal is stored in the background and the upcoming signals will be concatenated to each other. After the last appended signal, the user has to press the **Generate Append** button to plot generate Concatenated Signals.

In Figure 11, the reader can see the one concatenated signal example which consists of 3 different types of signals such as sinusoid, square wave and a linear chirp.

Spectrogram

I implemented the spectrogram function inside the .mlapp file as a private function. The function name is myspectrogram. Before implementation I learn how to perform STFT with a specified window funtion. The user can choose the window type in each panel (in sound recorder, pick a sound file and data generation tabs) and the selected window type will be used in spectrogram function. I provide the myspectrogram function below.

```
function myspectrogram(app, signal, Fs, M, H, win_type, beta, r)
% This function calculates STFT of a given signal and than
% plots the spectrogram of that function.
Y = signal;
len = length(Y);
duration = len/Fs;
% make sure that we have a column vector
[row, column] = size(Y);
if column > row
Y = Y';
End

if strcmp(win_type, 'Hamming')
      Wn = hamming(M);
elseif strcmp(win_type, 'Rectangle')
      Wn = ones(M,1);
elseif strcmp(win_type, 'Hanning')
      Wn = hanning(M);
elseif strcmp(win_type, 'Kaiser')
      Wn = kaiser(M, beta);
elseif strcmp(win_type, 'Blackman')
      Wn = blackman(M);
elseif strcmp(win_type, 'Gaussian')
     Wn = gausswin(M);
elseif strcmp(win_type, 'Tukey')
     Wn = tukeywin(M, r);
end

num_iteration = length(Y)/H;
spec_matrix = []; % Each column of this matrix is a DFT vector of each partition.

for k = 1:num_iteration - 1

    if H*(k-1 )+ M > len
        partition = Y( (k-1)*H:end,1);
        if strcmp(win_type, 'Hamming')
            Wn = hamming(length(partition));
        elseif strcmp(win_type, 'Rectangle')
            Wn = ones(length(partition),1);
        elseif strcmp(win_type, 'Hanning')
            Wn = hanning(length(partition));
        elseif strcmp(win_type, 'Kaiser')
            Wn = kaiser(length(partition), beta);
        elseif strcmp(win_type, 'Blackman')
            Wn = blackman(length(partition));
        elseif strcmp(win_type, 'Gaussian')
            Wn = gausswin(length(partition));
        elseif strcmp(win_type, 'Tukey')
            Wn = tukeywin(length(partition), r);
        end
```

```
    else
        partition = Y( (k-1)*H +(1:M) ,1);
    end
    F = fft(partition .* Wn, M); % M point fft

    % Since we deal with real signals
    % We can use the symmetry property of DFT by taking only the first half

    Half_F = F(1:ceil(M/2));

    spec_matrix = [spec_matrix, Half_F]; % stack each DFT column side by side

end


[r, c] = size(spec_matrix);

% Since the highest frequency in DFT (pi) corresponds to Fs/2 arrange the
% frequency values according to that. in kHz
f = ((Fs/(2*r))/1000).*(1:r);

% normalize t values to seconds
t = (duration/(c-1)).*(0:c-1);


surf(app.UISpec,t, f, 10*log(abs(spec_matrix)), 'EdgeColor', 'none');
axis(app.UISpec, 'tight');
colorbar(app.UISpec, "east");
colormap(app.UISpec, 'parula' );
app.UISpec.View = [0, 90];

end
```

The input parameters for this function

- `app` This parameter allows to reach global parameters in the app designer app.
- `signal` The signal that the spectrogram of itself to be calculated
- `Fs` The sampling frequency
- `M` The window size
- `H` the window shift size
- `win_type` the type of window that is selected by the user
- `beta` a parameter that used in kaiser window function
- `r` a parameter that used in tukey window function

Myspectrogram function calculates and plots the spectrogram of a signal with the steps as provided below;

1. Make sure that the provided signal is a column vector
2. Generate a window from provided window type and window length
3. Define an iteration parameter that make sures we calculate STFT until the end of the signal
4. In each iteration
- Retreive the partition from original signal with length that is equal to window length.
- Apply fft function to the product of partition and window with DFT length equal to window length
- Take only the half of the calculated DFT coefficients since we deal with real signals. Use the symmetry property of DFT.
- Add this column vector to spec_matrix in each iteration
5. According to the size of spec_matrix generate f and t axis where each of them are normalized according to sampling frequency and the total length of the singal in seconds repectively.
6. Use surf function to create 3D spectrogram plot where z axis represents the calculated spec_matrixs in dB scale.

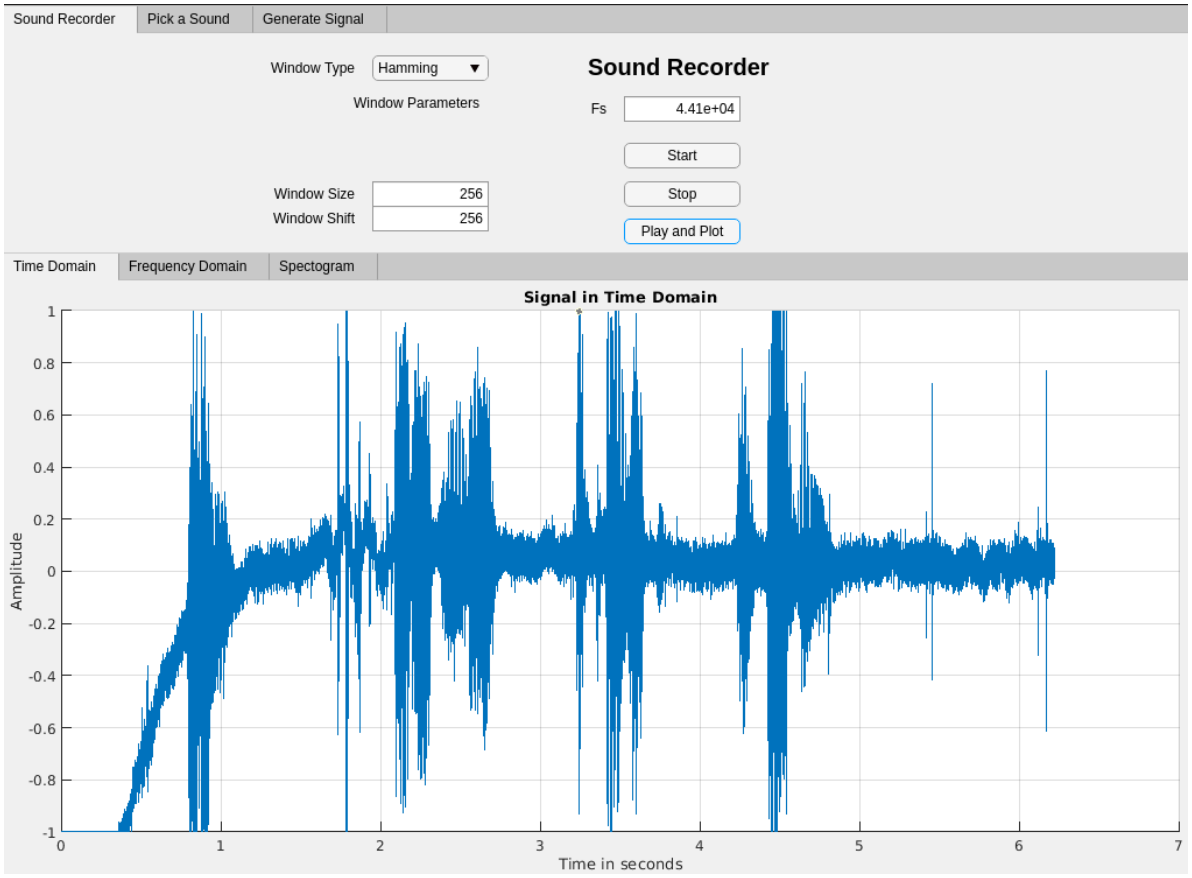
Conclusion

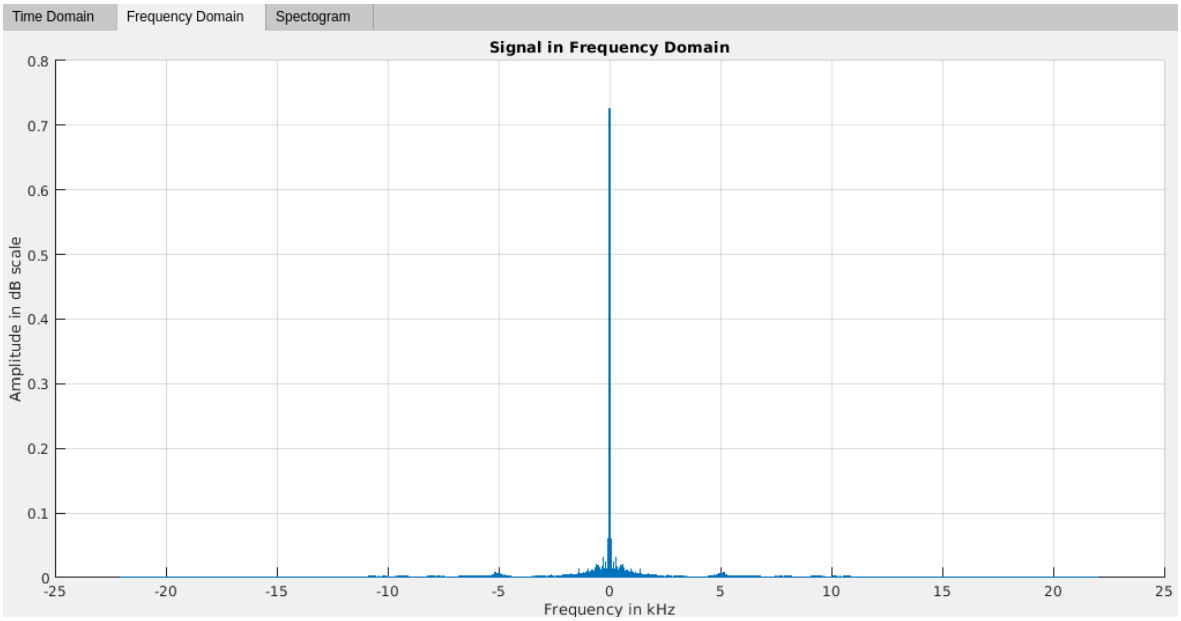
In this project, I learn the theoretical and practical implementation of STFT with different window sizes and shifts. Moreover, I understand the importance of spectrogram. Because in the time domain signal we couldn't see the frequency components on the other hand in the frequency domain we couldn't see the time domain information such that which frequency component occurs in which time instant. However, in the spectrogram, we can see both information in one plot. That ease the analysis of a signal.

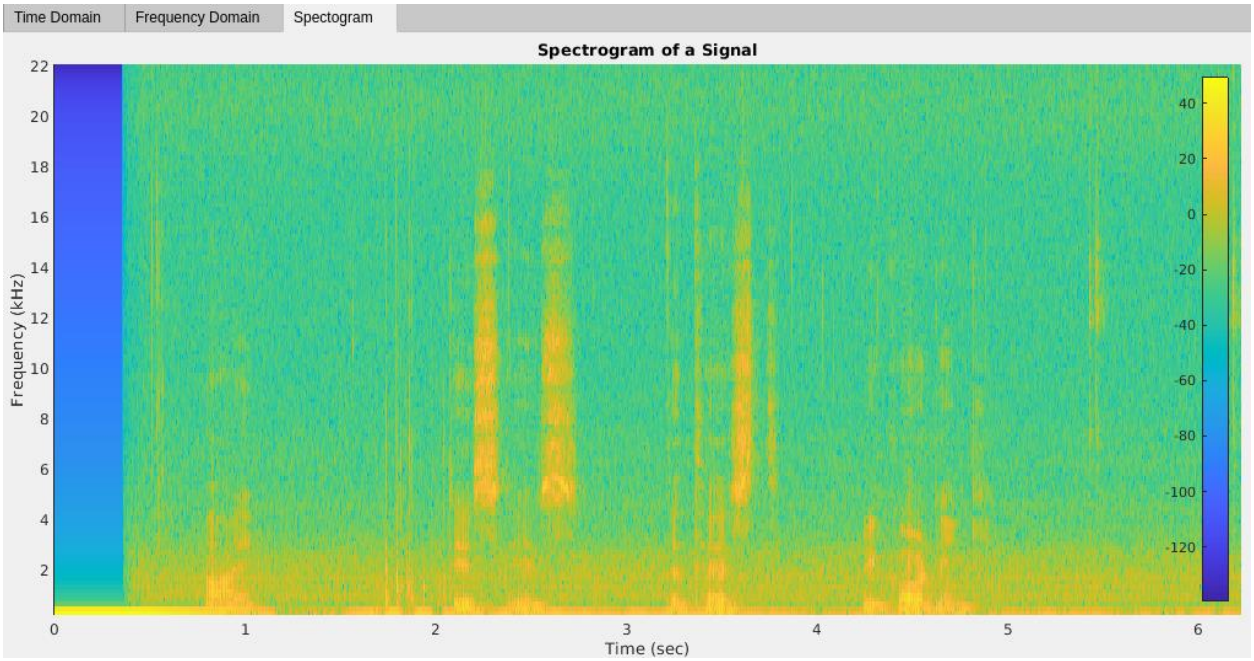
In addition to that, I also learn the effect of window size in the spectrogram plot. When the window length (also the dft length) increases we can identify the frequencies inside the signal more precisely but we lose the precision in the time axis. On the other hand, if we decrease the window length we can see the time axis in a high precision but again we lose precision in the frequency axis. This means depending on the window size, there is a trade-off between the time and frequency axis.

The project was beneficial for me, It gives me an insight that in the MATLAB environment I could build different apps if I needed to. In this part I will provide some test results for different types of signals.

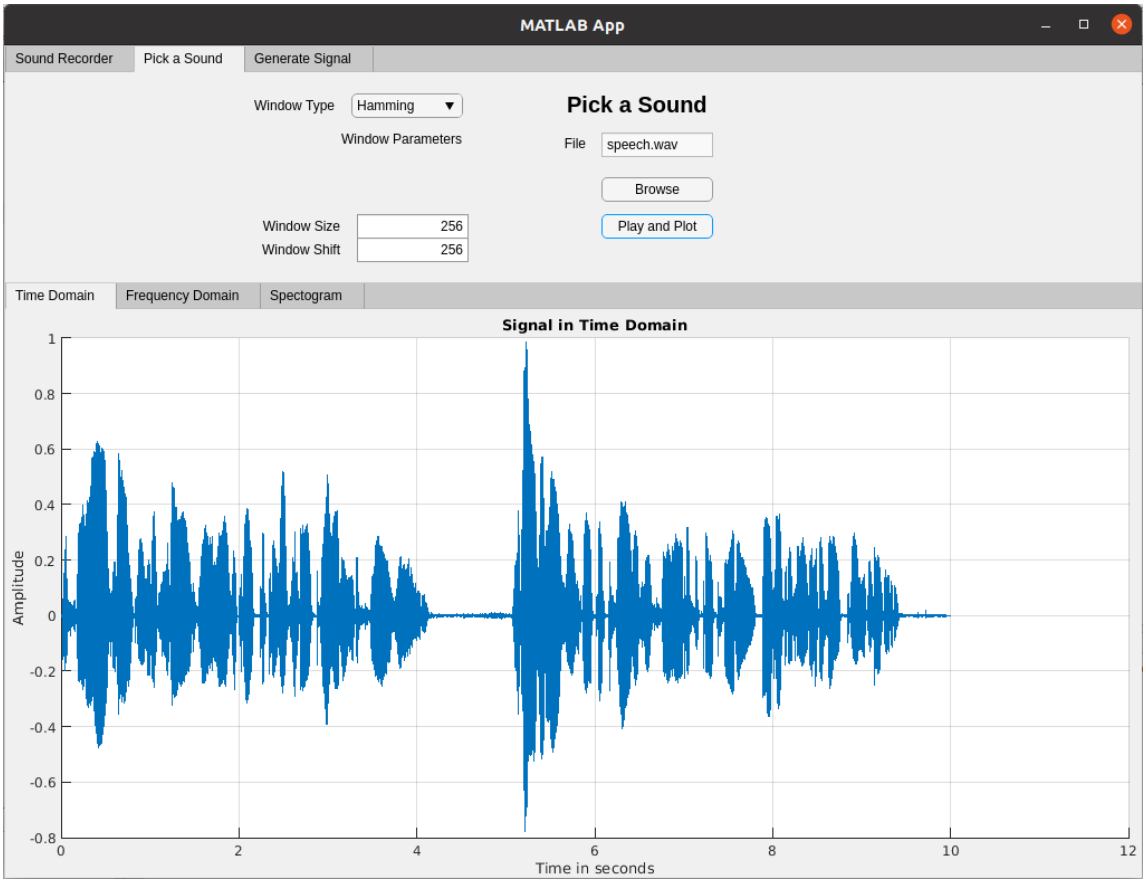
- **Recorded sound data**

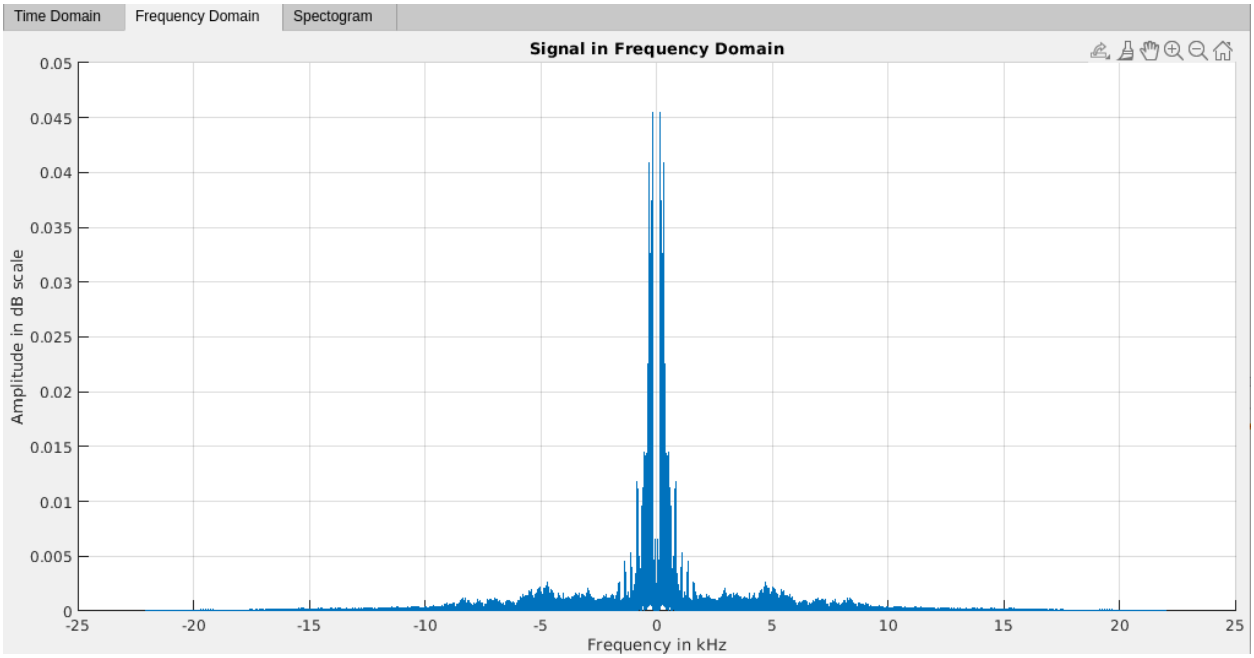


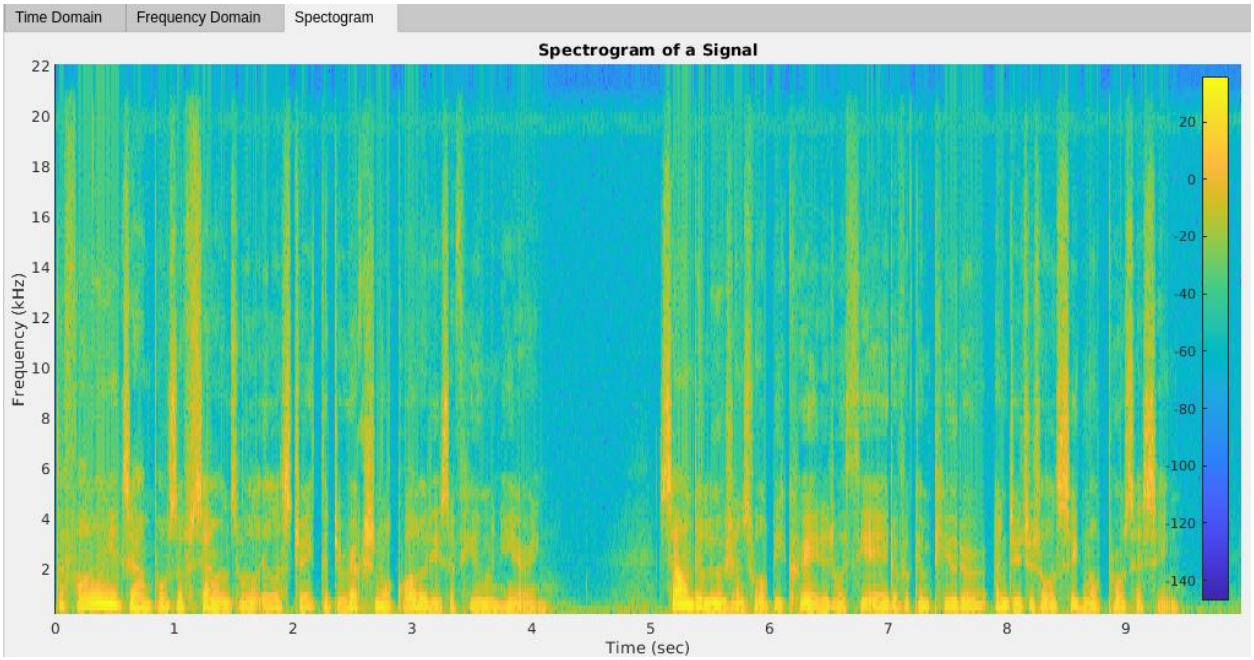



Time Domain
Frequency Domain
Spectogram
Spectrogram of a Signal
Frequency (kHz)
Time (sec)

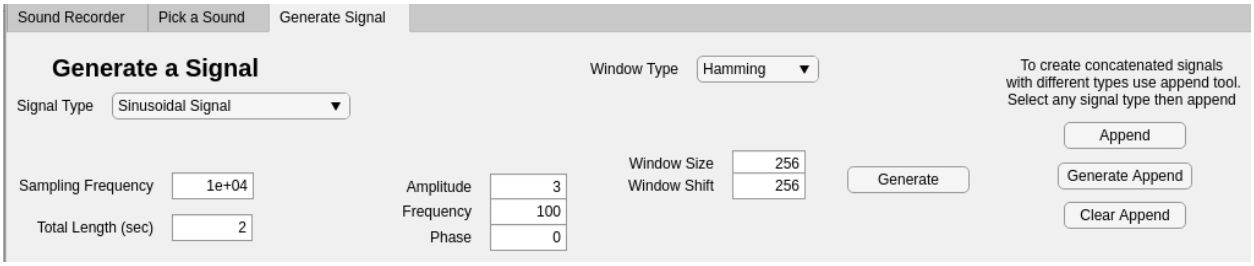
- **Sound File**

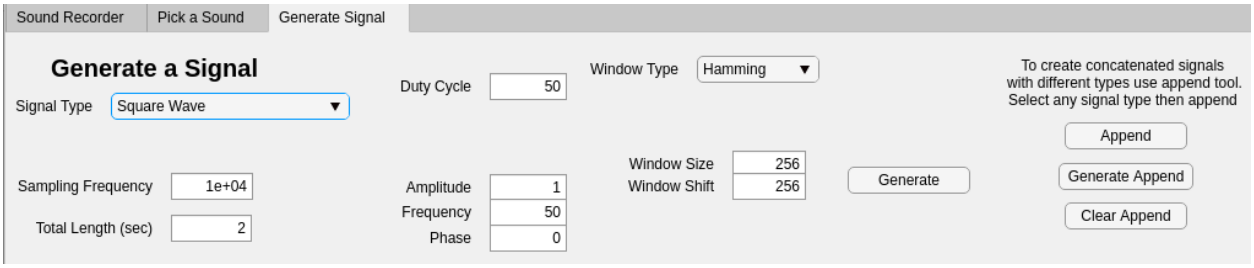

MATLAB App
Sound Recorder
Pick a Sound
Generate Signal
Window Type
Hamming
Window Parameters
Window Size
256
Window Shift
256
Pick a Sound
File
speech.wav
Browse
Play and Plot
Time Domain
Frequency Domain
Spectogram
Signal in Time Domain
Amplitude
Time in seconds

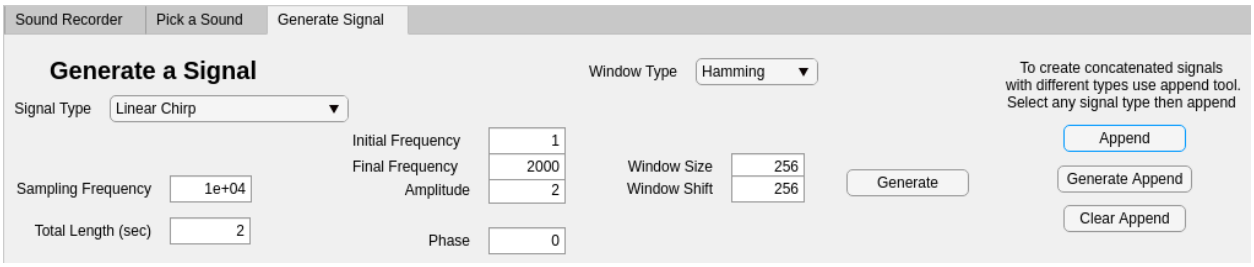

Time Domain
Frequency Domain
Spectogram
Signal in Frequency Domain
Amplitude in dB scale
Frequency in kHz
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-25
-20
-15
-10
-5
0
5
10
15
20
25

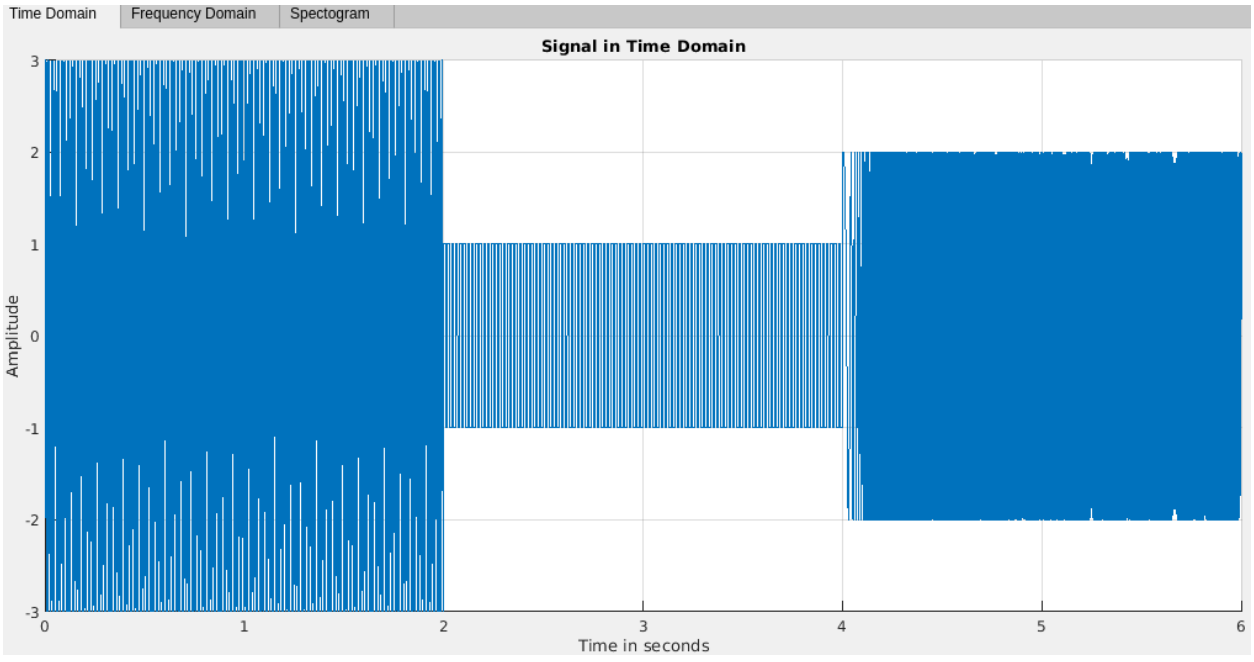

Time Domain
Frequency Domain
Spectogram
Spectrogram of a Signal
Frequency (kHz)
Time (sec)
22
20
18
16
14
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
20
0
-20
-40
-60
-80
-100
-120
-140

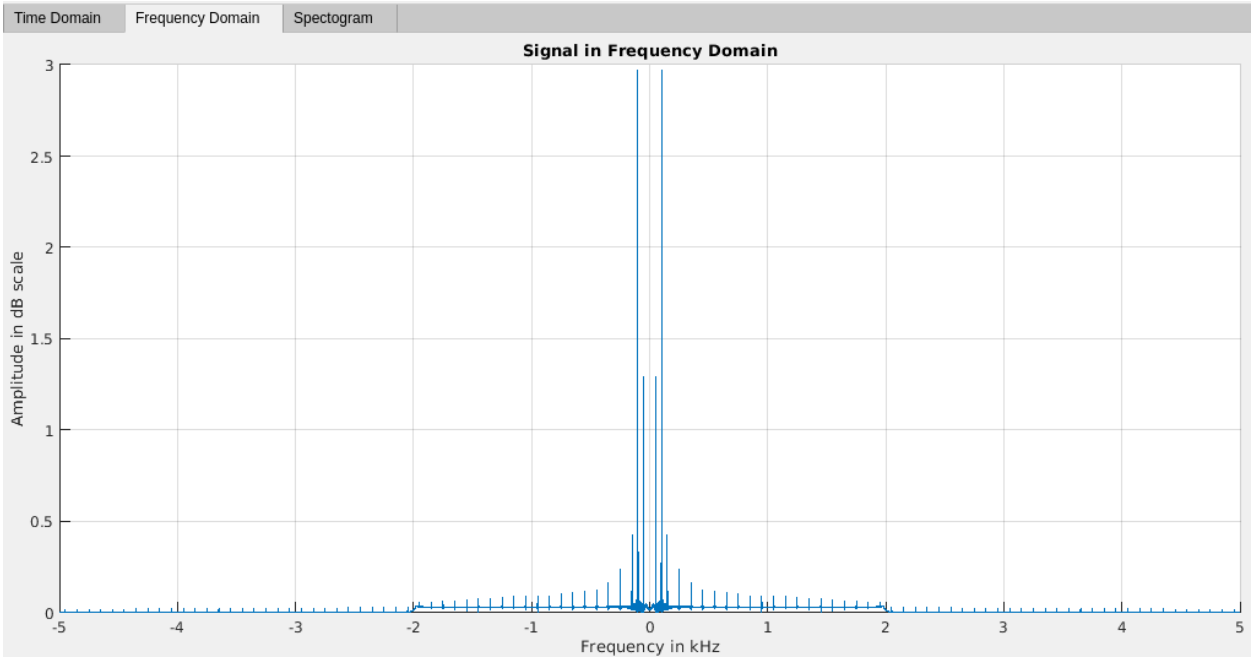
- **Generated Signal**

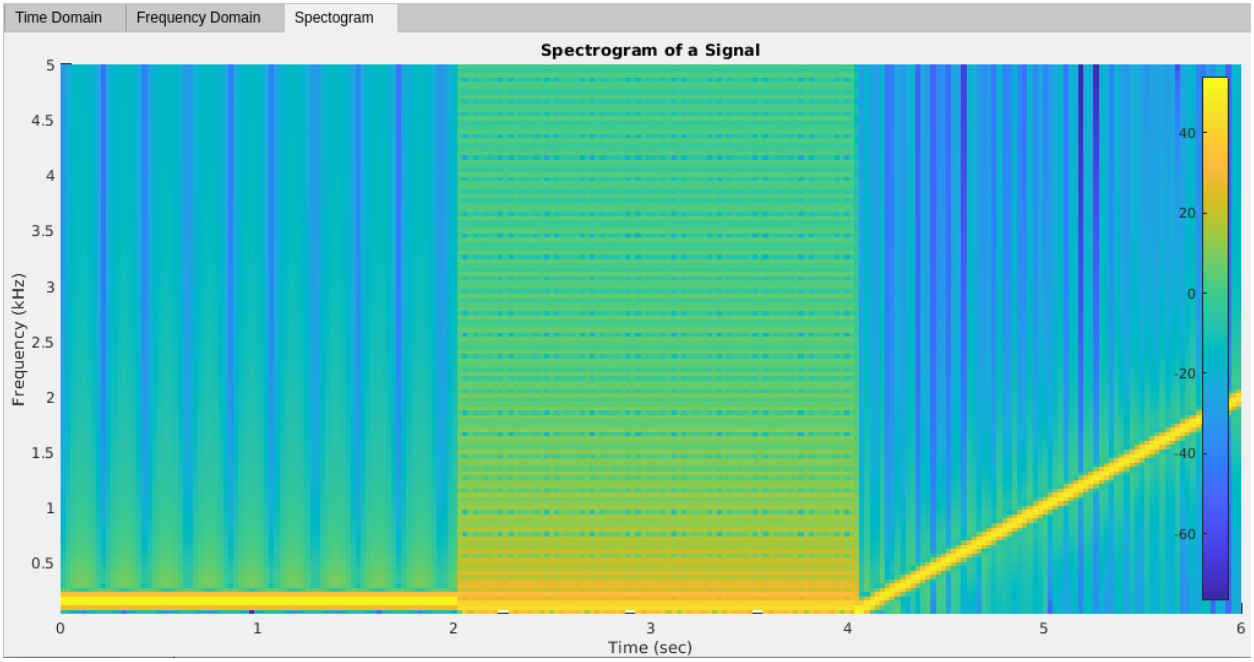









Time Domain
Frequency Domain
Spectogram
Signal in Frequency Domain
Amplitude in dB scale
Frequency in kHz


Time Domain
Frequency Domain
Spectogram
Spectrogram of a Signal
Frequency (kHz)
Time (sec)