MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE430

DIGITAL SIGNAL PROCESSING

PROJECT PART I REPORT

GROUP 2

19.12.2021

Berk Alperen Bener | 2165991

Seral Buse Atak | 2396398

ABSTRACT

For the EE430 Discrete Signal Processing Project Part I, a computer program has been implemented via MATLAB according to the specifications given in the project manual. Data acquisition and data generation features are created along with the manually written spectrogram function, which will be discussed in detail in Methods part. According to the Implementation steps given in the project manual, some tests have been applied and the results are observed. It is concluded that audio signals have distincts characteristic regarding its magnitude and frequency distribution. Using these distributions we can discriminate between individual distinct audio signals. Also we have observed that STFT applications have a trade-off between time and frequency resolution and different window have different contributions that should be tailored regarding different implementations.

INTRODUCTION

For the part I of the project, it is expected to create a computer program on MATLAB that is able to both generate a signal data and acquisite a recorded a sound file from the user or a previously saved sound, then display the signal in the time domain and take it's spectrogram, which is expected to be written explicitly without using the spectrogram() command of MATLAB. After implementing the program, comments and observations for varying different parameters on spectrogram calculation have been discussed according to the project description. For the program, the command window has been used for the user interface and parameters for the processes have been tried to be adjustable as much as possible. After all, this part of the project has contributed a lot for learning new skills on MATLAB and a deep understanding of STFT and spectrograms.

BACKGROUND

Short Time Fourier Transform (STFT)

The short time Fourier transform (STFT) is obtained by computing the Fourier transform of fixed length intervals. The formula can be seen in Equation 1.

$$X[n,\omega) = \sum_{m=0}^{2M} x[n-M+m]w[m]e^{-j\omega m}$$

Equation 1.

In this equation, w[n] denotes the windowing function and helps to extract portions of the length of w[n] around changing n values so that spectral properties of x[n] are obtained at the end. Some signals may have different statistics that change over time, this is where using STFT may be a good idea. In Figure 1, an overview of the STFT is given.

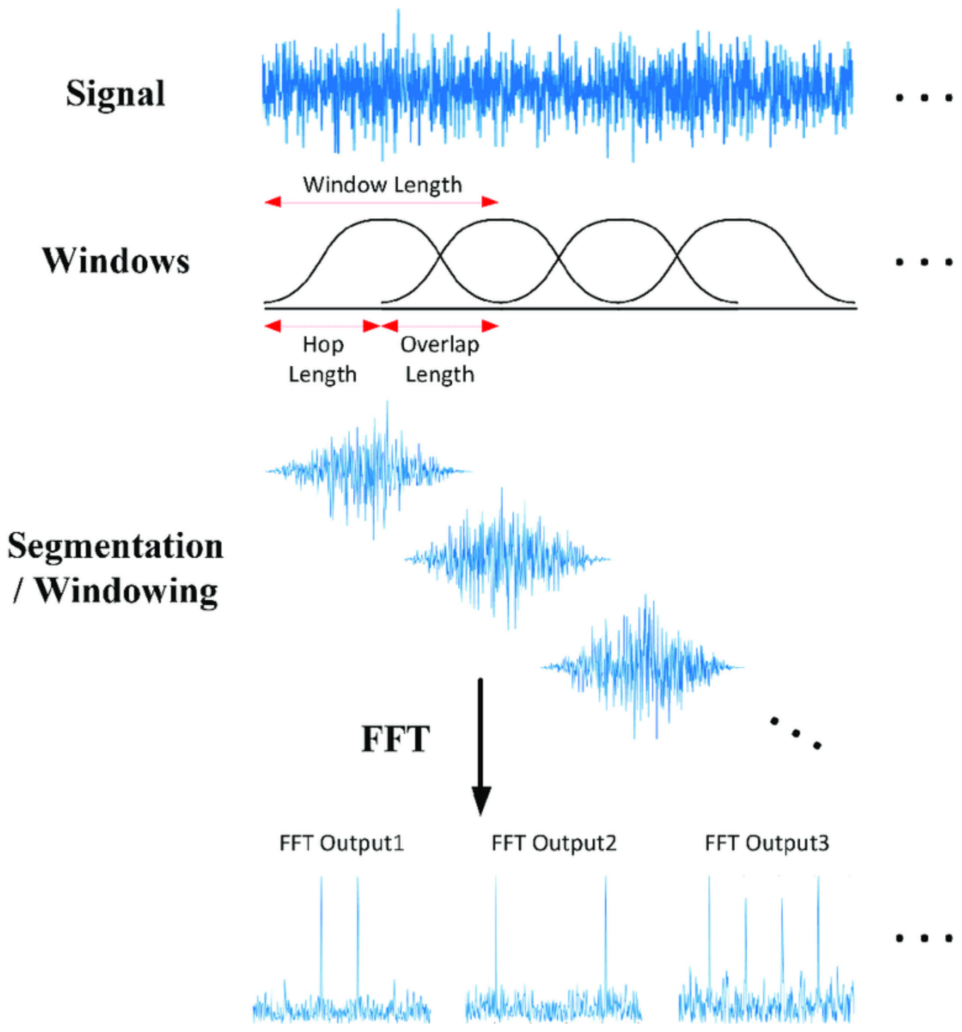


Figure 1: Illustration of Short Time Fourier Transform [1]

There exist important constraints regarding time and frequency resolution. As we have stated STFT is generally used to analyze how signal's frequency property changes over time. Regarding this application as we decrease window length we increase time resolution, which is beneficial for our application. But, on the other hand as we decrease window length we also decrease frequency resolution. Let's further elaborate this, when we take a window of N samples from a signal, with sampling rate of f, resulting FFT transformation creates N DFT coefficients, since we deal with real signals only half of it is useful to convey frequency information which is N/2 coefficients. Since the sampling rate is f, regarding Nyquist sampling theorem we conclude that at most we can find the f/2 frequency component of the signal in the corresponding interval. As a result, DFT coefficients represent frequencies with steps of (f/2)/(N/2) = f/N. This shows that as we decrease window length we increase step size, which corresponds to decrease in frequency domain resolution. Also another important point is overlap. We use overlapping windows to reduce artifacts at the boundaries, as generally windowing functions decrease to zero approaching the boundaries as a result we may lose data.

Windowing

The w[n] function in Equation 1 corresponds to the windowing function and it is also known as tapering or apodization function. It is usually a function that has zero values outside of the chosen interval and is symmetric around the middle of the interval. When w[n] is multiplied by a signal, it cancels the values of the signal that is outside of the w[n] interval and this provides a view through the window. By changing the window duration spectral leakage, which is caused by DFT operation, can be adjusted accordingly considering the requirements of the application. For this purpose, different kinds of windowing functions are used. In the project, Rectangular Window and Hamming Window are used which can be seen in Figure 2 and Figure 3 respectively.

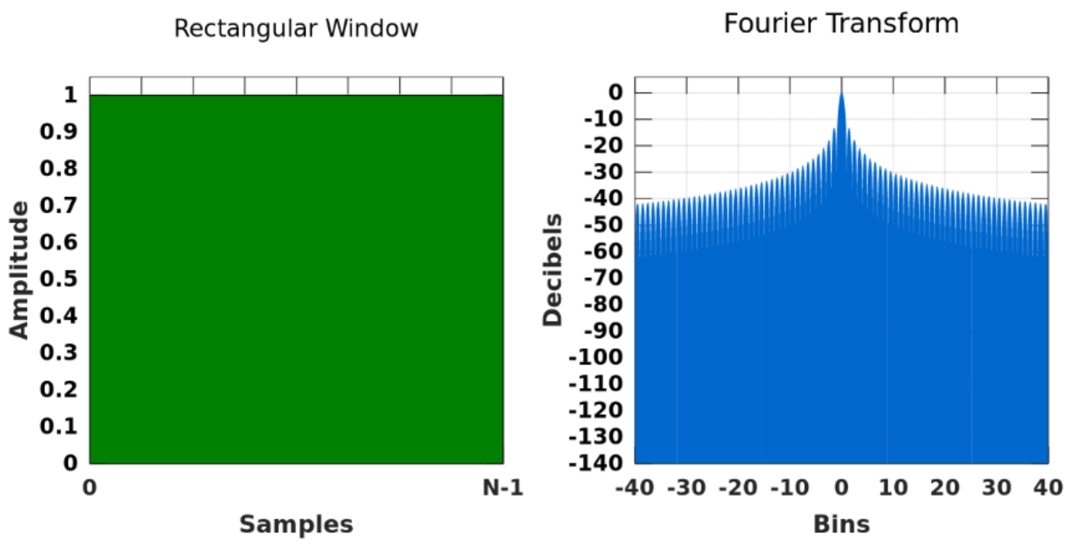


Figure 2: Rectangular Window and frequency response [2]

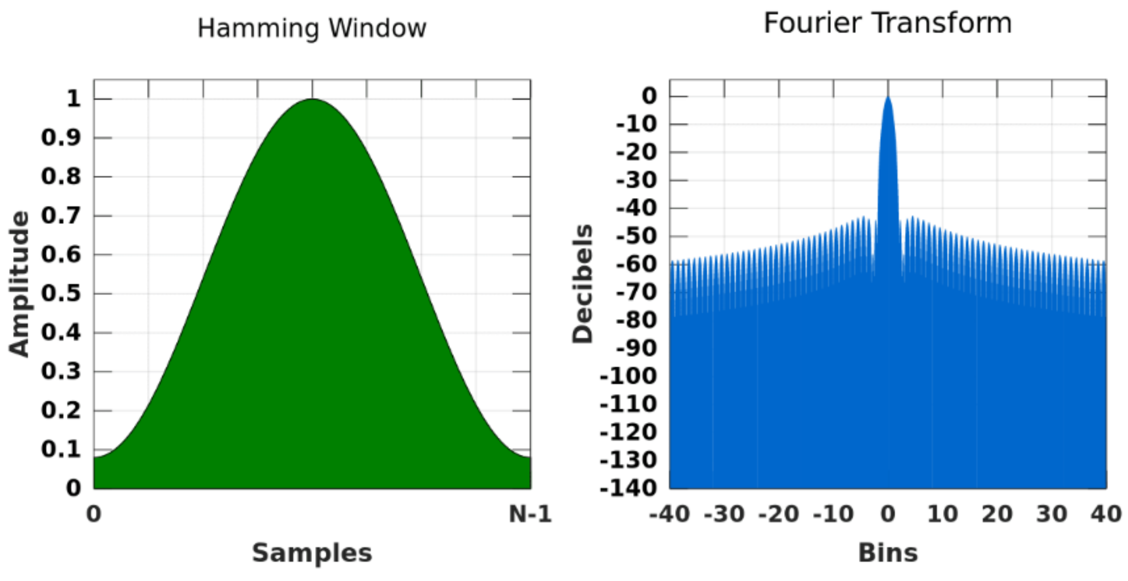


Figure 3: Hamming Window and frequency response [2]

Spectrogram

Using STFT we can also generate spectrograms. Spectrogram is a visual representation of the frequency domain information of the signal. It has 3 dimensions. First dimension represents time, which is divided by time intervals that are in the size of (window length)-(overlap). Second dimension is the frequency spectrum. For each time interval, this spectrum is divided by frequencies which are represented by DFT coefficients derived from signals in the corresponding time interval in the first dimension. Finally, the third dimension is the magnitude of the DFT coefficients. This is generally represented by heatmap. Heatmap is mapping between magnitude and the color for the corresponding (time interval,frequency interval) area. Colder colors represent lower magnitudes and warmer colors represent higher magnitudes. These magnitudes are the contribution of the frequency components of the sinusoid in the corresponding time interval.

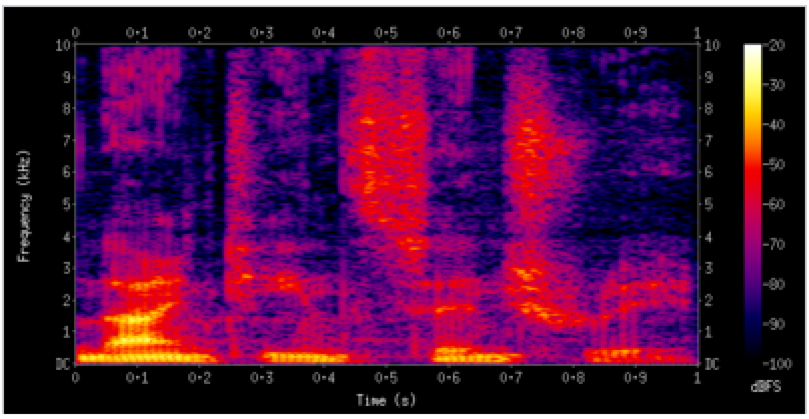
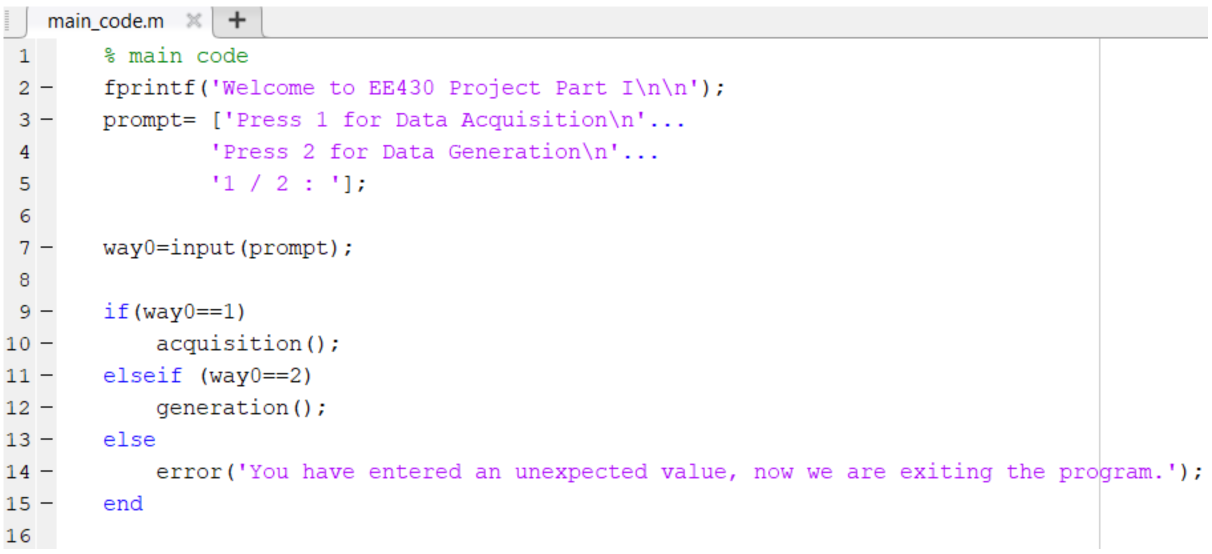


Figure 4: Example of the spectrogram, x-axis represents time intervals and y-axis represent frequency intervals. Warm colors represent higher magnitudes, while cold colors represent lower magnitudes.

METHODS

For the implementation of the program, MATLAB is used. In order to increase the readability of the code, functions are used for different types of operations. Comments are written on the code where it was necessary for easy adjustment later. The code will be inspected in terms of some major functions which lie under the *main_code.m* file. This file directs the user to acquisition() or generation() functions according to their choice as it can be seen in Figure 4.

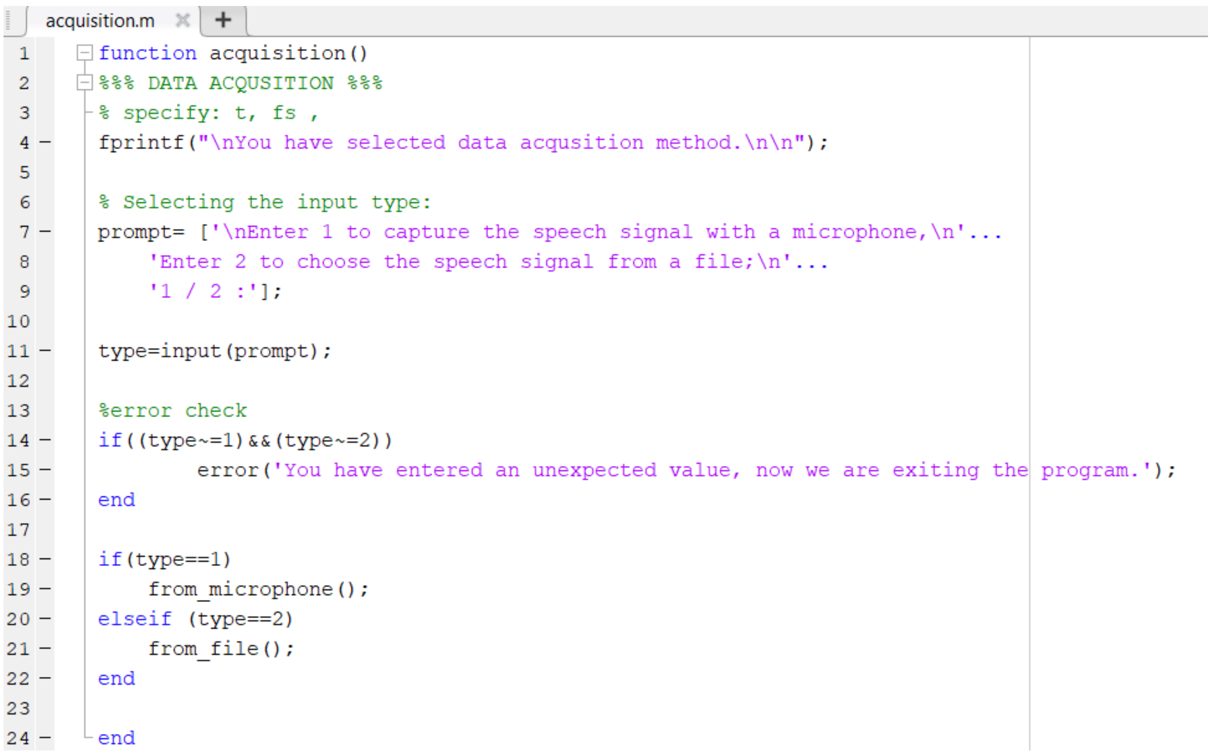


```
% main code
fprintf('Welcome to EE430 Project Part I\n\n');
prompt= ['Press 1 for Data Acquisition\n'...
         'Press 2 for Data Generation\n'...
         '1 / 2 : '];

way0=input(prompt);

if(way0==1)
    acquisition();
elseif (way0==2)
    generation();
else
    error('You have entered an unexpected value, now we are exiting the program.');
end
```

Figure 5: Screenshot of *main_code.m*

Data Acquisition and Visualization in Time Domain

If the user selects this method, *acquisition()* function starts to run. It suggests two ways for obtaining the signal as it can be seen in Figure 5

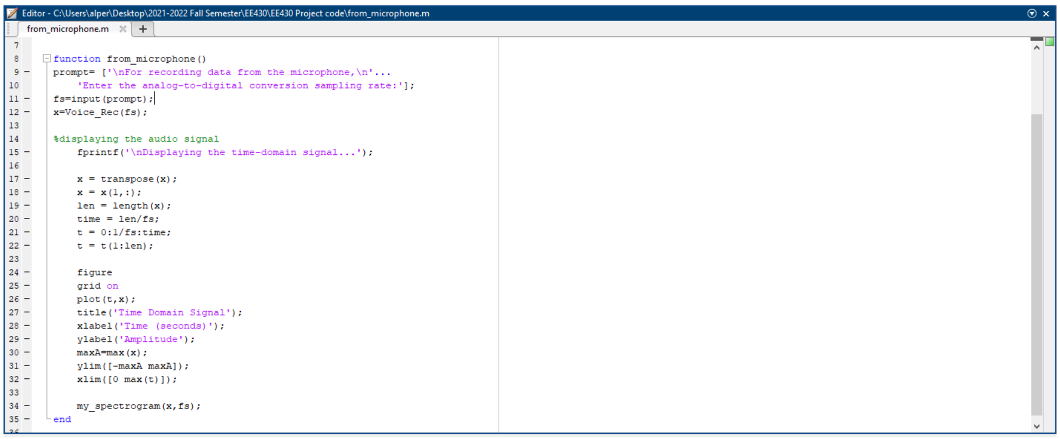


```
function acquisition()
%%% DATA ACQUSITION %%%
% specify: t, fs ,
fprintf("\nYou have selected data acqusition method.\n\n");

% Selecting the input type:
prompt= ['\nEnter 1 to capture the speech signal with a microphone,\n'...
    'Enter 2 to choose the speech signal from a file;\n'...
    '1 / 2 :'];

type=input(prompt);

%error check
if((type~=1)&&(type~=2))
        error('You have entered an unexpected value, now we are exiting the program.');
end

if(type==1)
    from_microphone();
elseif (type==2)
    from_file();
end

end
```

.**Figure 6:** Screenshot of the function file *acquisition.m*

Sound Data From a Microphone

If the user selects option 1, code allows the user to input a voice signal into the computer. First of all, the command window asks the user for the preferred sampling rate of their continuous voice signal to generate discrete time representation of it. Subsequently, input sampling rate passed into Voice_Rec function. Function first asks the user, the duration of time that the user wants to save her voice. It then proceeds to ask, whether to record her voice and 'option_rec' variable is set to 'y', this variable is later used to control for repeated voice recording operation. If the user enter 'y', subsequently command window expect user's starting que, which is enter key, to start recording for determined duration which user specified.

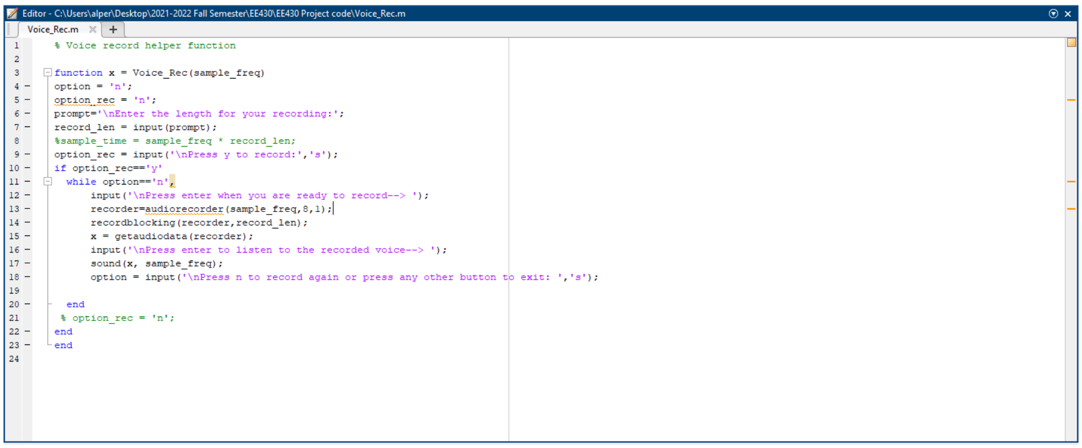

Editor - C:\Users\alper\Desktop\2021-2022 Fall Semester\EE430\EE430 Project code\from_microphone.m

from_microphone.m

```matlab
function from_microphone()
prompt= ['\nFor recording data from the microphone,\n'...
    'Enter the analog-to-digital conversion sampling rate:'];
fs=input(prompt);
x=Voice_Rec(fs);

%displaying the audio signal
    fprintf('\nDisplaying the time-domain signal...');

    x = transpose(x);
    x = x(1,:);
    len = length(x);
    time = len/fs;
    t = 0:1/fs:time;
    t = t(1:len);

    figure
    grid on
    plot(t,x);
    title('Time Domain Signal');
    xlabel('Time (seconds)');
    ylabel('Amplitude');
    maxA=max(x);
    ylim([-maxA maxA]);
    xlim([0 max(t)]);

    my_spectrogram(x,fs);
end
```

Figure 7: Screenshot of the function file *from_microphone.m*

In the recording routine we used audiorecorder(sample_freq,8,1) function which has parameters sampling frequency, resolution of amplitude which is specified by number of bits encoding in this case it is 8 and number of channels that input voice should have, which is specified as 1. Using this function we have created a recorder object. In the next step the recordblocking function takes the recorder object and recording duration and starts the recording process. After determining the recording duration, the sampled voice record is saved to a vector which is called 'x'. After this recording operation, the function allows users to listen to recorded voices. Finally, the function asks the user whether she wants to record again. If she wants to record again, the whole process repeats until the user decides to finish recording by entering any other button than the character 'n'. Then code proceeds to exit Voice_Rec function.


Editor - C:\Users\alper\Desktop\2021-2022 Fall Semester\EE430\EE430 Project code\Voice_Rec.m

Voice_Rec.m

```matlab
% Voice record helper function

function x = Voice_Rec(sample_freq)
option = 'n';
option_rec = 'n';
prompt='\nEnter the length for your recording:';
record_len = input(prompt);
%sample_time = sample_freq * record_len;
option_rec = input('\nPress y to record:','s');
if option_rec=='y'
  while option=='n',
      input('\nPress enter when you are ready to record--> ');
      recorder=audiorecorder(sample_freq,8,1);
      recordblocking(recorder,record_len);
      x = getaudiodata(recorder);
      input('\nPress enter to listen to the recorded voice--> ');
      sound(x, sample_freq);
      option = input('\nPress n to record again or press any other button to exit: ','s');

  end
 % option_rec = 'n';
end
end
```

Figure 7: Screenshot of the function file *Voice_Rec.m*

To visualize sampled voice signals first of all the output vector is transposed. This step is necessary to use it in plot() function.Then, we extracted only the first row of the outputted sampled signal, this step is done because in the ‘sound data from file’ section, we read .wav file which includes multiple channels, since we only want to process single channel we make regarding extraction. This step is not useful in the ‘sound data from microphone’ part. But, for the sake of simplicity we allow sound data from the microphone to use it too. Then, we create a time array, which has (1/fs) wide steps from zero to sound duration. Finally in line 22 in figure 7, we make sure that time and output signal have the same number of elements which makes sure plot() function runs properly. Finally, we plot time domain signals by normalizing amplitude with highest amplitude of the voice signal and x axis as seconds.

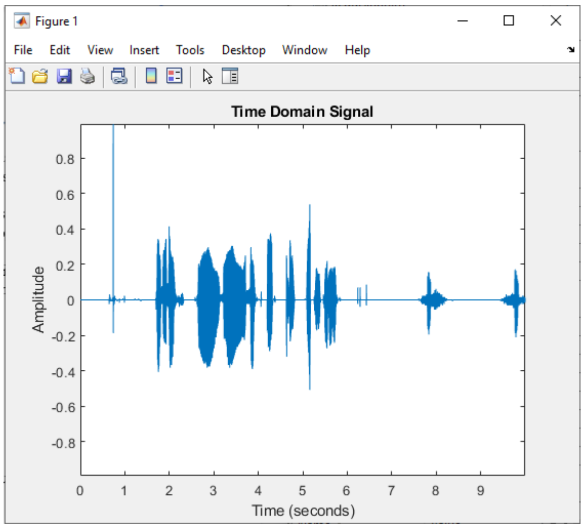
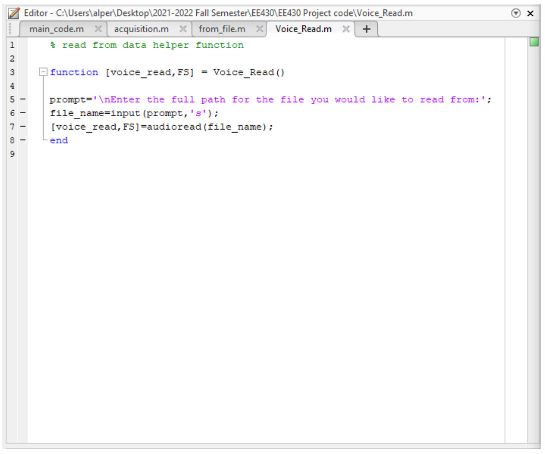


Figure 8: Voice signal in the time domain representation.

Sound data from a file

If the user selects option 2, the program asks the user to input file name with its extension. The function audioread() which is used to get samples in the audio file and sampling rate is capable of both working with .wav and .mp3 files. Function outputs this information as a vector in the form of [voice_read,FS].


Editor - C:\Users\alper\Desktop\2021-2022 Fall Semester\EE430\EE430 Project code\Voice_Read.m
main_code.m
acquisition.m
from_file.m
Voice_Read.m

```
% read from data helper function

function [voice_read,FS] = Voice_Read()

prompt='\nEnter the full path for the file you would like to read from:';
file_name=input(prompt,'s');
[voice_read,FS]=audioread(file_name);
end
```

Figure 9: Screenshot of the function file *Voice_Read.m*

Subsequently the same algorithm to visualize the 'sound data from a microphone' also deployed in 'sound data from a file' part. For an example of … second .wav file resulting data in the time domain can be seen in figure….

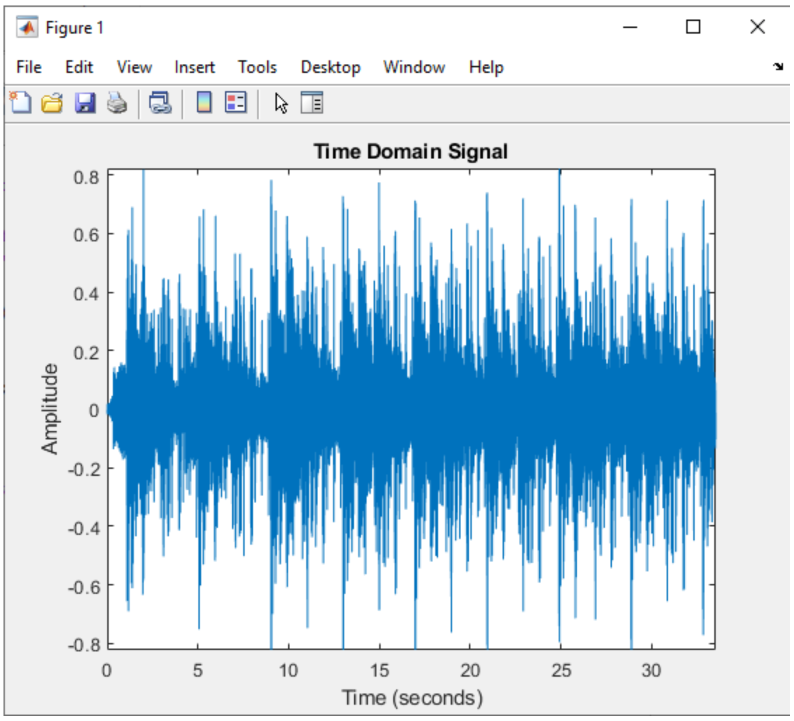
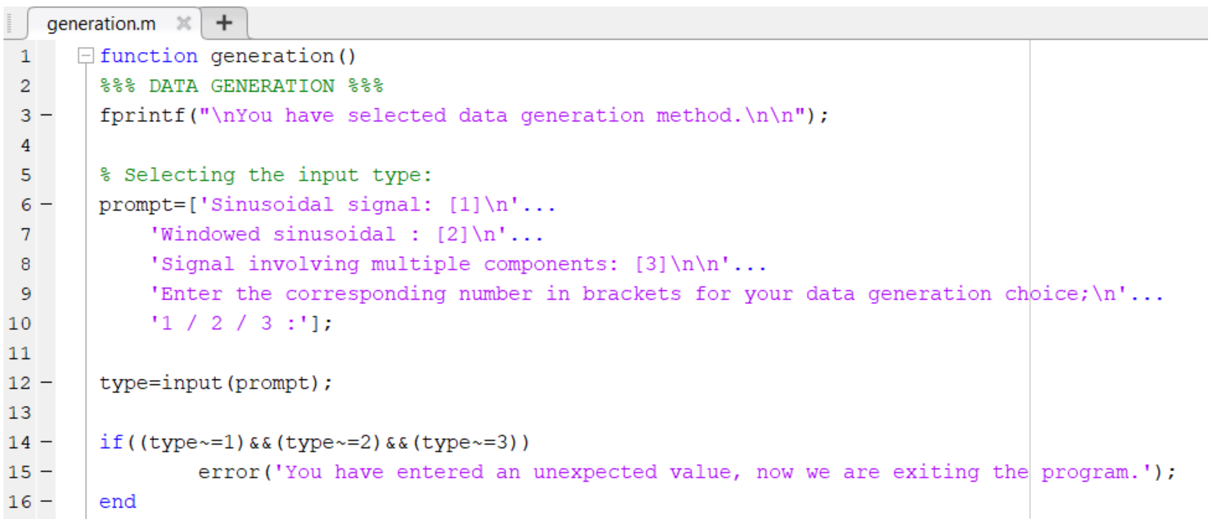


Figure 10: Music.wav file in the time domain representation.

Data Generation

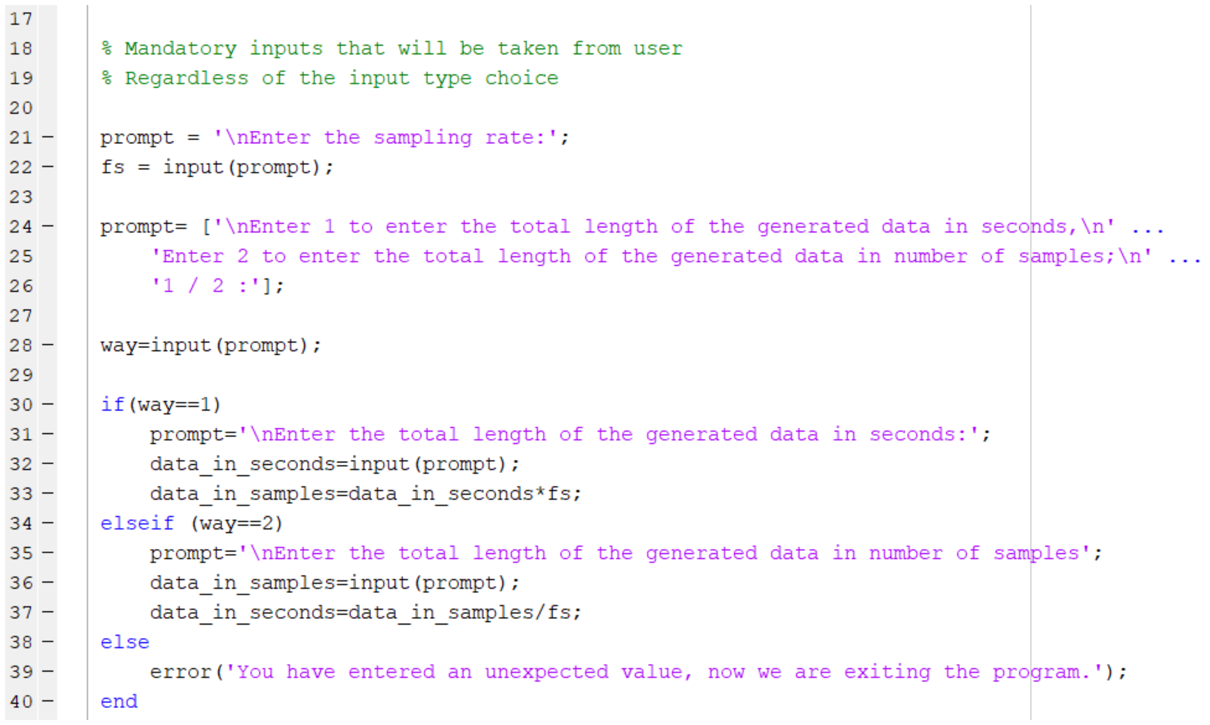
If the user selects this method, *generation()* function starts to run. It suggests three ways for generating the signal as it can be seen in Figure x.



```
function generation()
%%% DATA GENERATION %%%
fprintf("\nYou have selected data generation method.\n\n");

% Selecting the input type:
prompt=['Sinusoidal signal: [1]\n'...
    'Windowed sinusoidal : [2]\n'...
    'Signal involving multiple components: [3]\n\n'...
    'Enter the corresponding number in brackets for your data generation choice;\n'...
    '1 / 2 / 3 :'];

type=input(prompt);

if((type~=1)&&(type~=2)&&(type~=3))
        error('You have entered an unexpected value, now we are exiting the program.');
end
```

Figure 11: Screenshot 1 of the function file *generation.m*

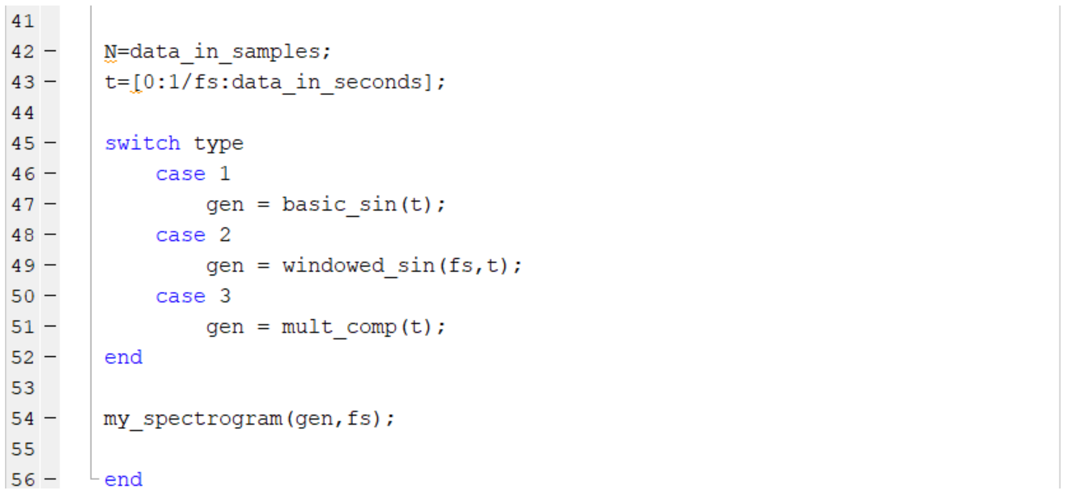
Then the common parameters regardless of the user's selected generation type are taken from the user as it can be seen in Figure 12.



```
% Mandatory inputs that will be taken from user
% Regardless of the input type choice

prompt = '\nEnter the sampling rate:';
fs = input(prompt);

prompt= ['\nEnter 1 to enter the total length of the generated data in seconds,\n' ...
    'Enter 2 to enter the total length of the generated data in number of samples;\n' ...
    '1 / 2 :'];

way=input(prompt);

if(way==1)
    prompt='\nEnter the total length of the generated data in seconds:';
    data_in_seconds=input(prompt);
    data_in_samples=data_in_seconds*fs;
elseif (way==2)
    prompt='\nEnter the total length of the generated data in number of samples';
    data_in_samples=input(prompt);
    data_in_seconds=data_in_samples/fs;
else
    error('You have entered an unexpected value, now we are exiting the program.');
end
```

Figure 12: Screenshot 2 of the function file *generation.m*

The code computes the total number of samples N and the time vector t whether the user has entered the total length of the generated data in number of samples or seconds and after this operation switch case block directs the code on the according function as a result of user's choice. At the end, *my_spectrogram()* function is used with the generated signal, which will be explained in detail later. This process can be seen in Figure x+2.

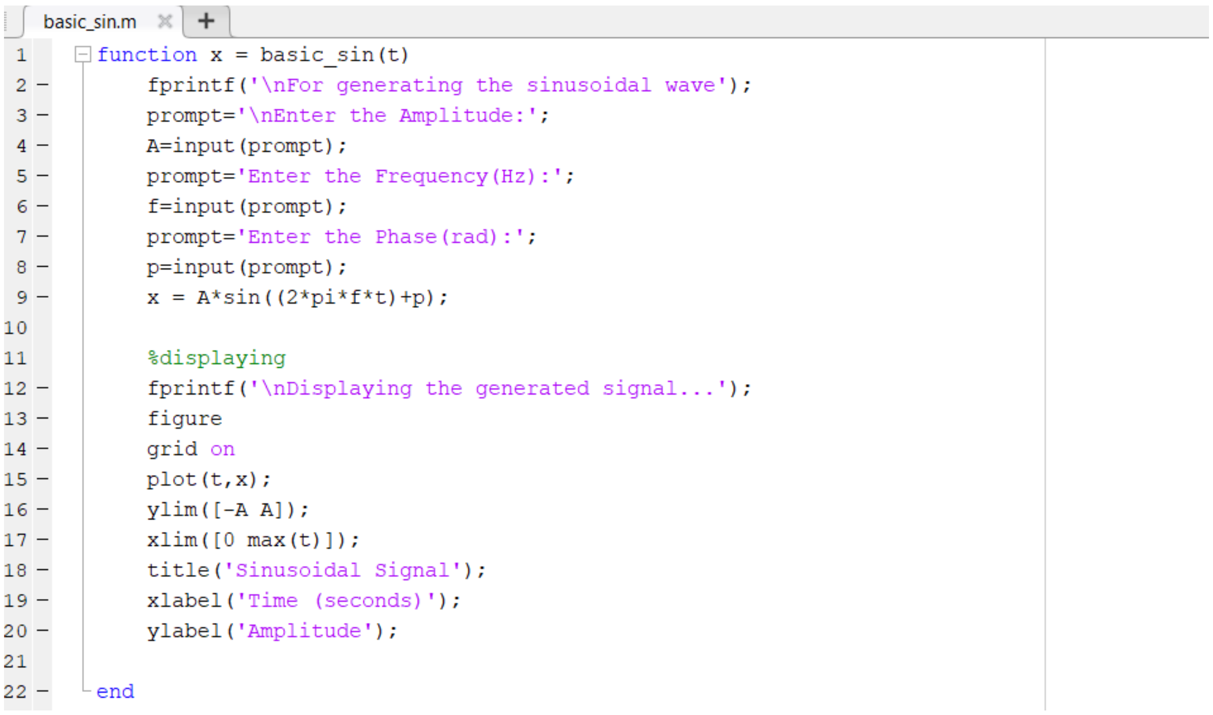


```
N=data_in_samples;
t=[0:1/fs:data_in_seconds];

switch type
    case 1
        gen = basic_sin(t);
    case 2
        gen = windowed_sin(fs,t);
    case 3
        gen = mult_comp(t);
end

my_spectrogram(gen,fs);

end
```

Figure 13: Screenshot 3 of the function file *generation.m*

Sinusoidal signal

If the user selects this generation method, *basic_sin()* function takes amplitude, frequency and phase parameters from the user and creates the sinusoid by using sin() function. Then the signal is displayed in the figure window. These processes can be seen respectively in Figure 14.

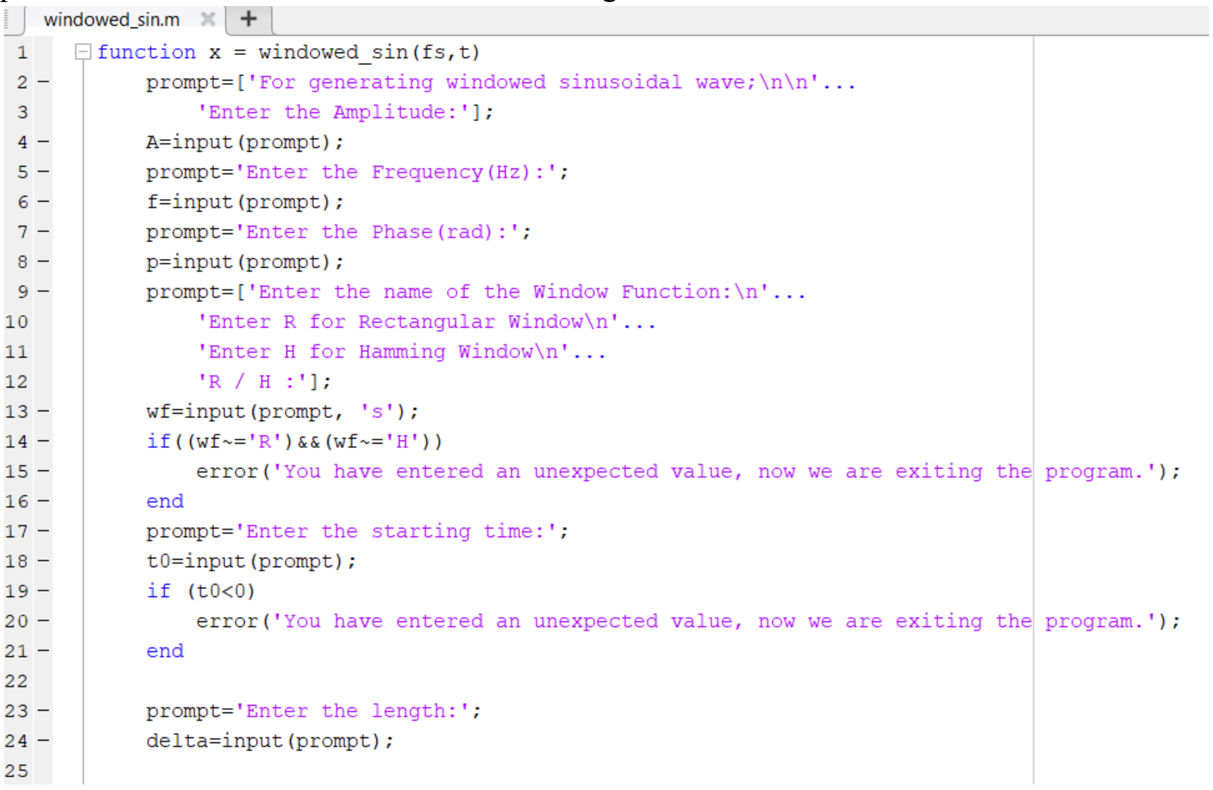


```
function x = basic_sin(t)
    fprintf('\nFor generating the sinusoidal wave');
    prompt='\nEnter the Amplitude:';
    A=input(prompt);
    prompt='Enter the Frequency(Hz):';
    f=input(prompt);
    prompt='Enter the Phase(rad):';
    p=input(prompt);
    x = A*sin((2*pi*f*t)+p);

    %displaying
    fprintf('\nDisplaying the generated signal...');
    figure
    grid on
    plot(t,x);
    ylim([-A A]);
    xlim([0 max(t)]);
    title('Sinusoidal Signal');
    xlabel('Time (seconds)');
    ylabel('Amplitude');

end
```

Figure 14: Screenshot of the function file *basic_sin.m*

Windowed sinusoidal

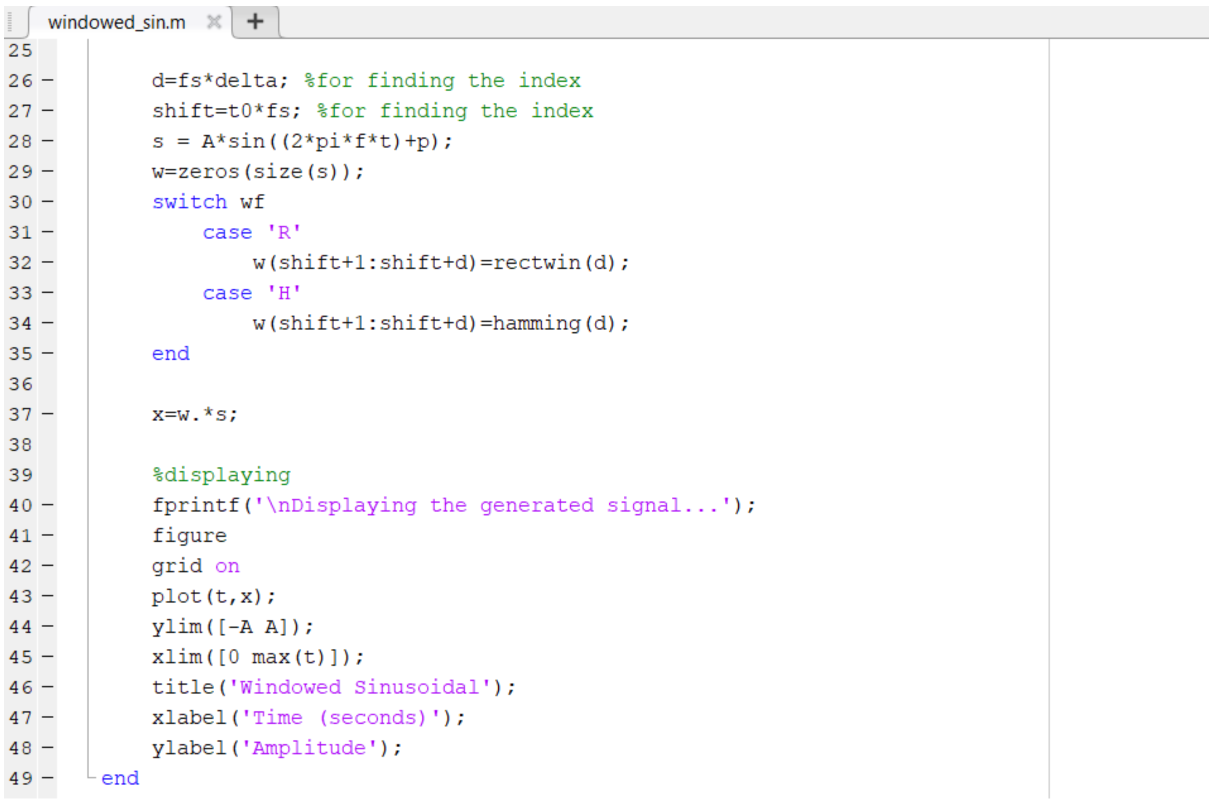
If the user selects this generation method, *windowed_sin()* function takes amplitude, frequency, phase, window function type, window function starting time and duration parameters from the user as it can be seen in Figure 15.



```
function x = windowed_sin(fs,t)
    prompt=['For generating windowed sinusoidal wave;\n\n'...
        'Enter the Amplitude:'];
    A=input(prompt);
    prompt='Enter the Frequency(Hz):';
    f=input(prompt);
    prompt='Enter the Phase(rad):';
    p=input(prompt);
    prompt=['Enter the name of the Window Function:\n'...
        'Enter R for Rectangular Window\n'...
        'Enter H for Hamming Window\n'...
        'R / H :'];
    wf=input(prompt, 's');
    if((wf~='R')&&(wf~='H'))
        error('You have entered an unexpected value, now we are exiting the program.');
    end
    prompt='Enter the starting time:';
    t0=input(prompt);
    if (t0<0)
        error('You have entered an unexpected value, now we are exiting the program.');
    end

    prompt='Enter the length:';
    delta=input(prompt);

```

Figure 15: Screenshot 1 of the function file *windowed_sin.m*

By multiplying the sampling frequency with delta and t0, the indexes for the time vector to create the window function are found. Then the main sinusoid is created using *sin()* function with given parameters. An empty vector is created with the size of the sinusoid and a window function is created on the computed interval according to selected type, which are Rectangular or Hamming for this project. These processes can be seen in Figure x+5.

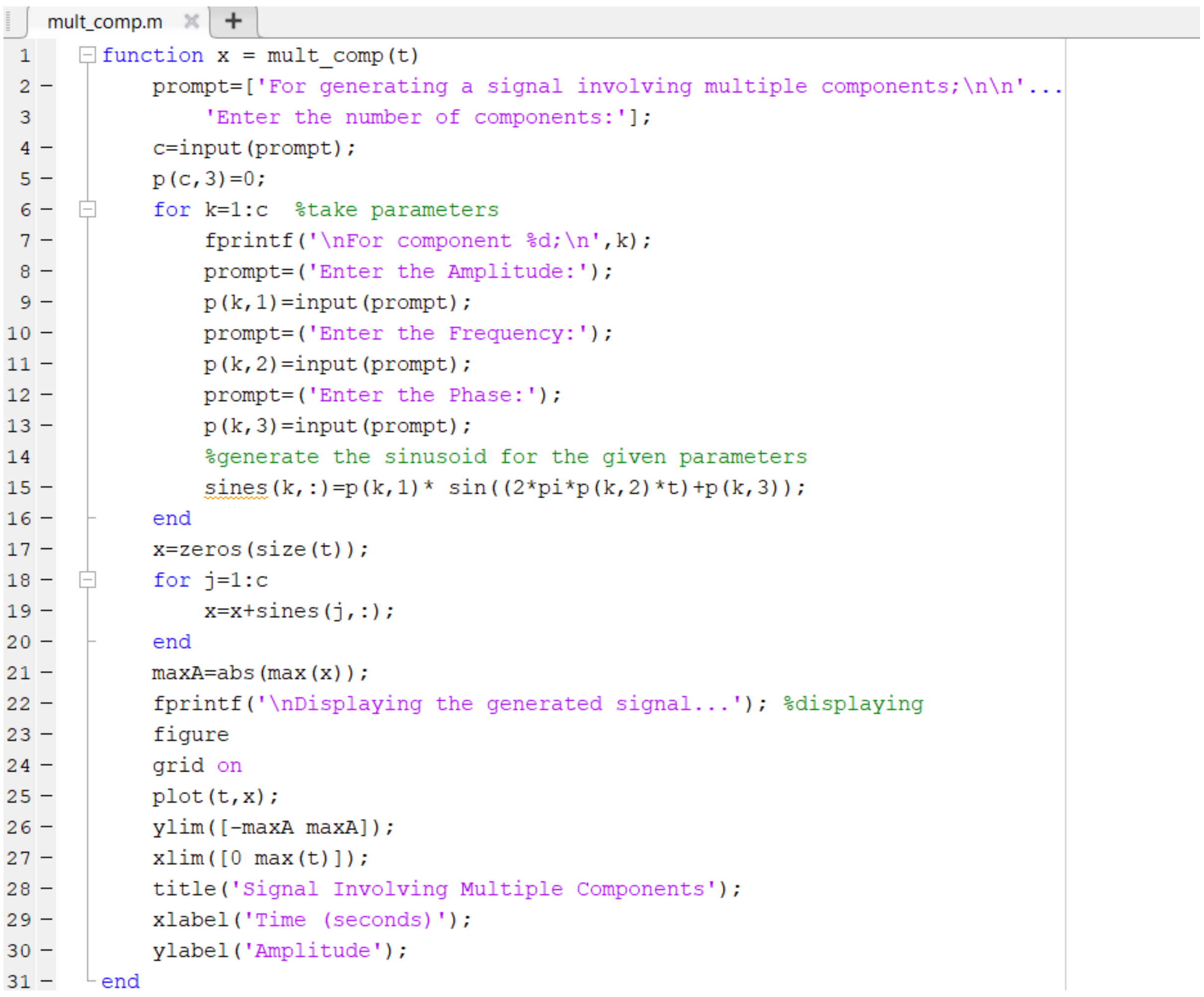

windowed_sin.m

```

    d=fs*delta; %for finding the index
    shift=t0*fs; %for finding the index
    s = A*sin((2*pi*f*t)+p);
    w=zeros(size(s));
    switch wf
        case 'R'
            w(shift+1:shift+d)=rectwin(d);
        case 'H'
            w(shift+1:shift+d)=hamming(d);
    end

    x=w.*s;

    %displaying
    fprintf('\nDisplaying the generated signal...');
    figure
    grid on
    plot(t,x);
    ylim([-A A]);
    xlim([0 max(t)]);
    title('Windowed Sinusoidal');
    xlabel('Time (seconds)');
    ylabel('Amplitude');
end
```

Figure 16: Screenshot 2 of the function file *windowed_sin.m*

Signal involving multiple components

If the user selects this generation method, *mult_comp()* function firstly askes the number of components and with a for loop takes amplitude, frequency and phase parameters for each sinusoid from the user and creates the sinusoids by using sin() function and stores them in an array. After that,with another for loop it sums the created sinusoids and then the resulting signal is displayed in the figure window. These processes can be seen respectively in Figure x+6.



```
mult_comp.m
function x = mult_comp(t)
    prompt=['For generating a signal involving multiple components;\n\n'...
        'Enter the number of components:'];
    c=input(prompt);
    p(c,3)=0;
    for k=1:c  %take parameters
        fprintf('\nFor component %d;\n',k);
        prompt=('Enter the Amplitude:');
        p(k,1)=input(prompt);
        prompt=('Enter the Frequency:');
        p(k,2)=input(prompt);
        prompt=('Enter the Phase:');
        p(k,3)=input(prompt);
        %generate the sinusoid for the given parameters
        sines(k,:)=p(k,1)* sin((2*pi*p(k,2)*t)+p(k,3));
    end
    x=zeros(size(t));
    for j=1:c
        x=x+sines(j,:);
    end
    maxA=abs(max(x));
    fprintf('\nDisplaying the generated signal...'); %displaying
    figure
    grid on
    plot(t,x);
    ylim([-maxA maxA]);
    xlim([0 max(t)]);
    title('Signal Involving Multiple Components');
    xlabel('Time (seconds)');
    ylabel('Amplitude');
end
```

Figure 17: Screenshot of the function file *mult_comp.m*

Spectrogram

Whether a user acquires audio data from a microphone, from a file or it is a generated signal. All of them are finally visualized by a function which is called my_spectogram(x,fs). My spectrogram creates heat map visualization of frequency content of the chunk of time series signal. Function parameters have sampled audio signal vector or sampled user generated signal ‘x’ and sampling frequency ‘fs’. After the program is called. It asks the user which windowing function she prefers. There are 2 options which are ‘Rectangular’ and ‘Hamming’ window. After taking window type preference it asks window size ‘ws’ which determines length of the windows. Depending on user inputs it creates these windows using rectwin(ws) and hamming(ws) functions. To create a spectrogram we need DFT coefficients for the time intervals that we divided sampled audio vectors into. To this end we used [S,F,T] = stft(x,fs,'Window',w,'FrequencyRange',rngs(1),'Overlaplength',overlap,'FFTLength',nfft) function. Function works as follows, first we put input parameter ‘x’ to our sampled audio data then we enter sampling frequency ‘fs’,also we input window as ‘w’.

Since, STFT outputs DFT coefficients mathematically we output coefficients that are symmetric around y axis. Since we need only positive frequency coefficients for visualization purposes, we select “onesided” parameter as frequency range. It is important that we input the parameter as pairs which are label of the parameter ‘FrequencyRange’ and range value “onesided.”. The next parameter is ‘Overlaplength’ which should be smaller than window size since we slide the window over the audio signal vector and time interval moves toward the end of it by (window size-overlap) steps. Finally we include input ‘FFTLength’. This parameter determines the resolution of the frequency domain representation of the signal that is inside the concerning time interval. We determine this to be 512 and overlap to be 220.

RESULTS

According to the steps given in the implementation part of the project, the following tests are applied.

a. In the first part we have recorded the word electricity. Figure 18 shows the distinct letters of the words by regarding the emphasis on each letter.

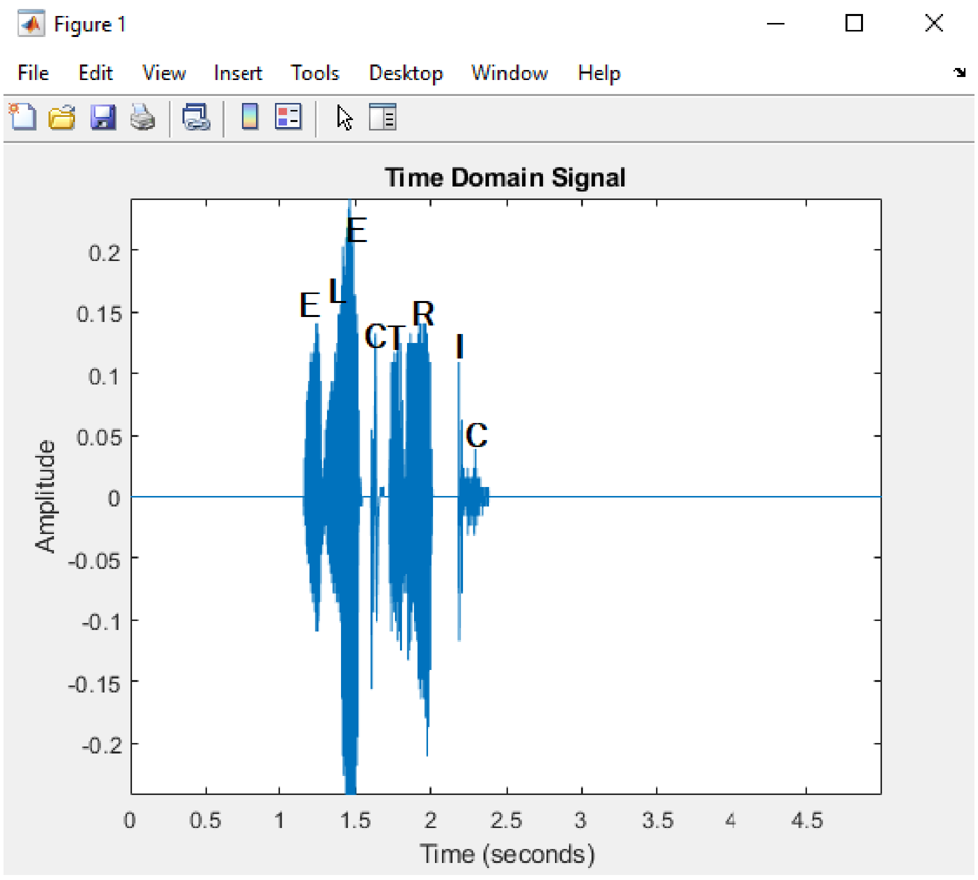


Figure 18: Time domain representation of the word “electric”

b. Spectrograms of the word "electricity" are given below which are sampled with a sampling frequency of 20 kHz for 5 seconds, with window length of 2000 samples and 300 overlaps. In figure 19, we have used rectangular, in figure 20, we have used Hamming window.

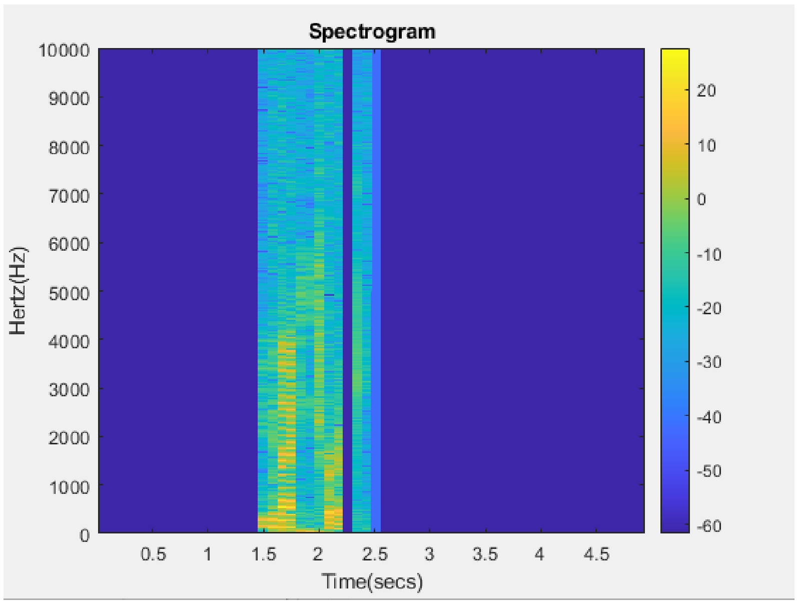


Figure 19: The spectrogram of word "electric" with rectangular window

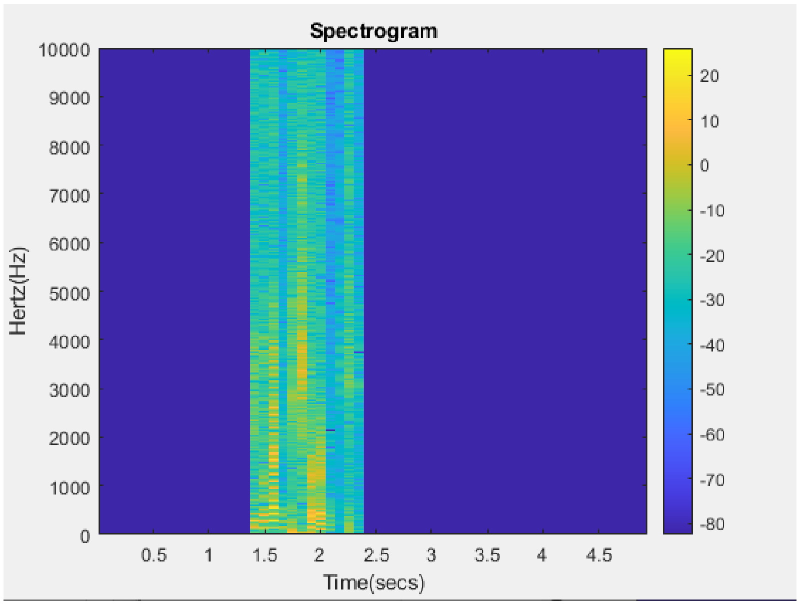


Figure 20: The spectrogram of word "electric" with Hamming window

c. We have created a sinusoidal signal with amplitude 1, frequency 200 Hz, phase 0. And we have sampled it with a sampling frequency of 1 kHz for 10 seconds. In figure 21,22,23 we have increased window lengths from 250,500 to 750 respectively. In the figures 24,25,26 we have used windows which have lengths of 200,400,600 which are integer multiples of the sinusoid.

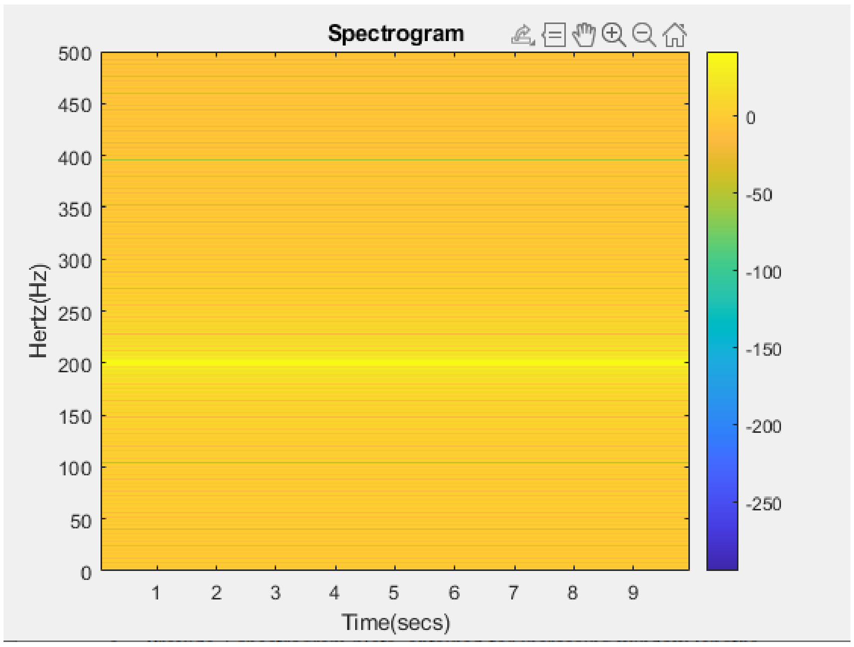


Figure 21: The spectrogram of word “electric” with Hamming window

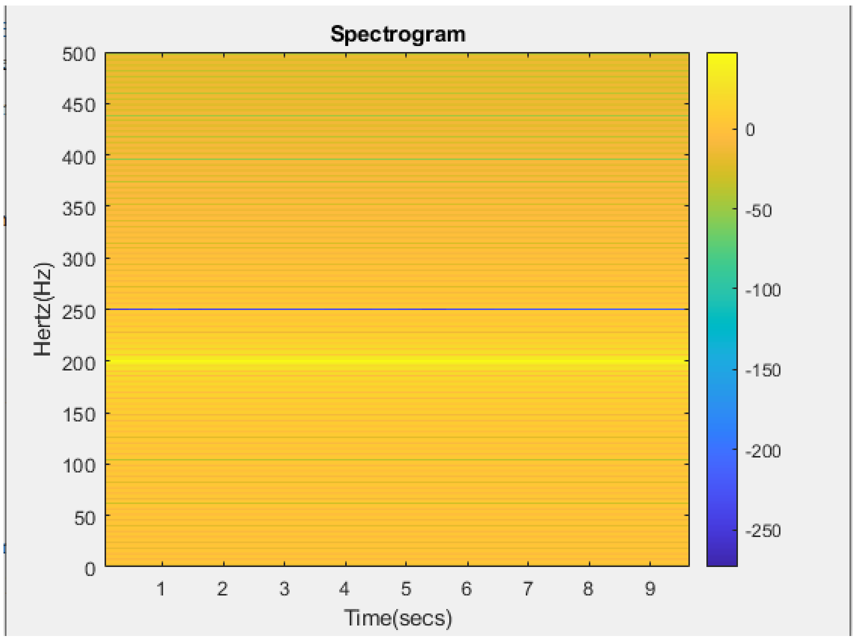


Figure 22: The spectrogram of word “electric” with Hamming window

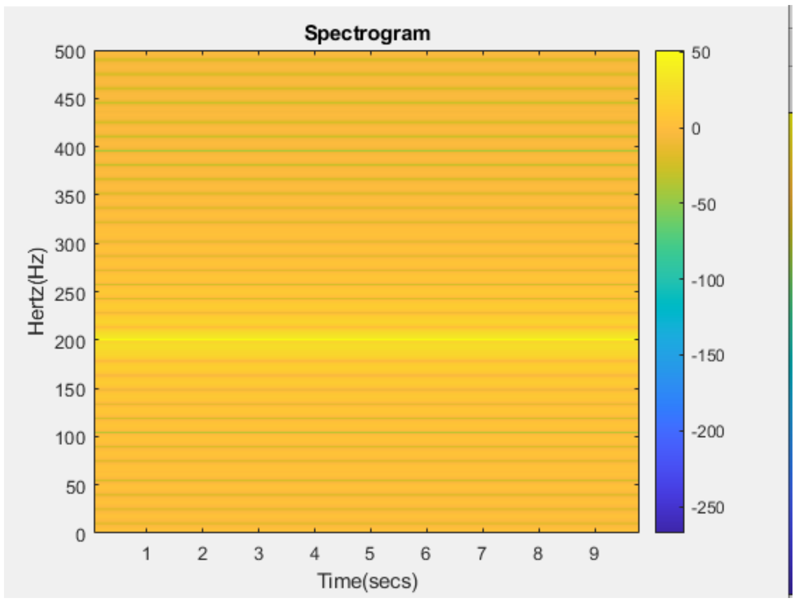


Figure 23: The spectrogram of word "electric" with Hamming window

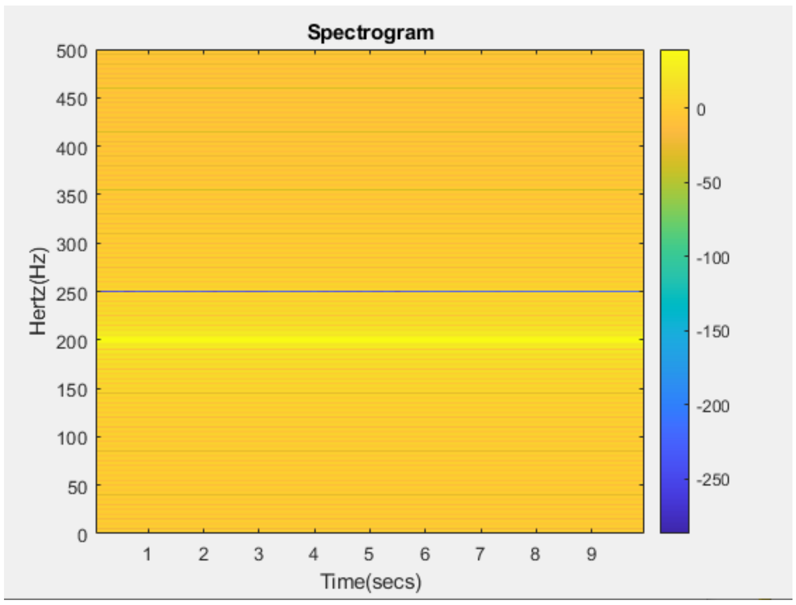


Figure 24: The spectrogram of word "electric" with Hamming window

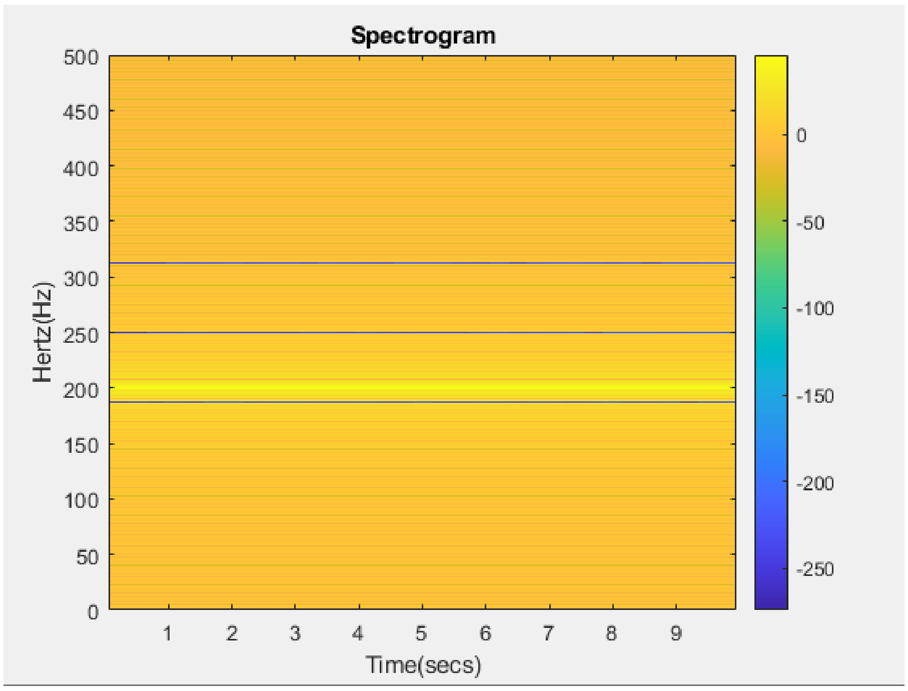


Figure 25: The spectrogram of word “electric” with Hamming window

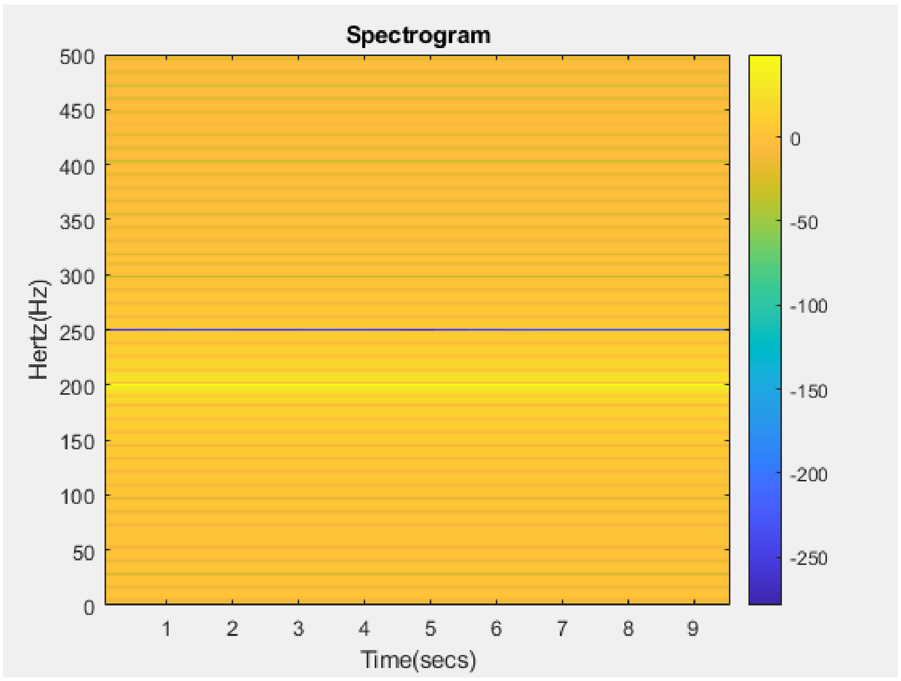


Figure 26: The spectrogram of word “electric” with Hamming window

We have observed that as we increase window length in figure 21,22,23 we have increased frequency resolution. As we have observed that color changes between frequency intervals become sharper which can be easily observed around 200 Hz which is the main harmonic of the sinusoid. Most distinguishing feature about figure 24,25,26 is that since N is multiple of sinudoid’s frequency, it has homogeneous coor distribution along the x-axis.

CONCLUSIONS AND DISCUSSION

The results that were obtained show that audio signals have distincts characteristic regarding its magnitude and frequency distribution. Using these distributions we can discriminate between individual distinct audio signals. Also we have observed that STFT applications have a trade-off between time and frequency resolution and different windows have different contributions that should be tailored regarding different implementations.

For future improvement, more parameters can be made adjustable via user input. More windowing function choices may be used in *my_spectrogram()* function. User interface can be designed more neatly using MATLAB App Designer instead of Command Window. A function can be created for displaying time plots in order to avoid redundancy in the code. More comments can be added through the code. More input error checks may be applied. Frequency domain plots as magnitude and phase versus frequency can also be added. For the final spectrogram, *spectrogram()* function output can also be provided for the user to compare the two results.

REFERENCES

[1] H. Jeon, Y. Jung, S. Lee, and Y. Jung, “Area-efficient short-time Fourier transform processor for time–frequency analysis of non-stationary signals,” Appl. Sci. (Basel), vol. 10, no. 20, p. 7208, 2020.

[2] “Windowing,” VRU, 02-Jun-2021. [Online]. Available: https://vru.vibrationresearch.com/lesson/windowing-process-data/. [Accessed: 19-Dec-2021].

[3] https://www.mathworks.com/help/audio/gs/real-time-audio-in-matlab.html