FDM APP: A Matlab Project on Frequency Division Multiplexing

Course title: Numerical Technique Laboratory

Course number: EEE 212

Submitted to

Mr. Mahbub Alam
Associate Professor
Department of EEE
BUET

Md. Suzit Hasan Nayem
Lecturer
Department of EEE
BUET

Submitted by

Fazle Rabbi	1806087	Department of EEE Section: B1 Group: 6 Level-2 term-1
Md. Al Shahriar Shakil	1806088	
Md. Ayenul Azim	1806089	
Indrojit Sarkar	1806090	

Objective:

In this project we will show the Frequency Division multiplexing. At first, we will take two audio signals as inputs. Then, the signal will be modulated by adding a carrier wave with high frequency. After modulation, we will simulate the channel signal which is the sum of the two modulated signals. At the receiver's end, we will demodulate the channel signal, by undergoing some consecutive operations, respectively, band pass filtering, separating carrier wave and low pass filtering. Hence, we will obtain our desired output.

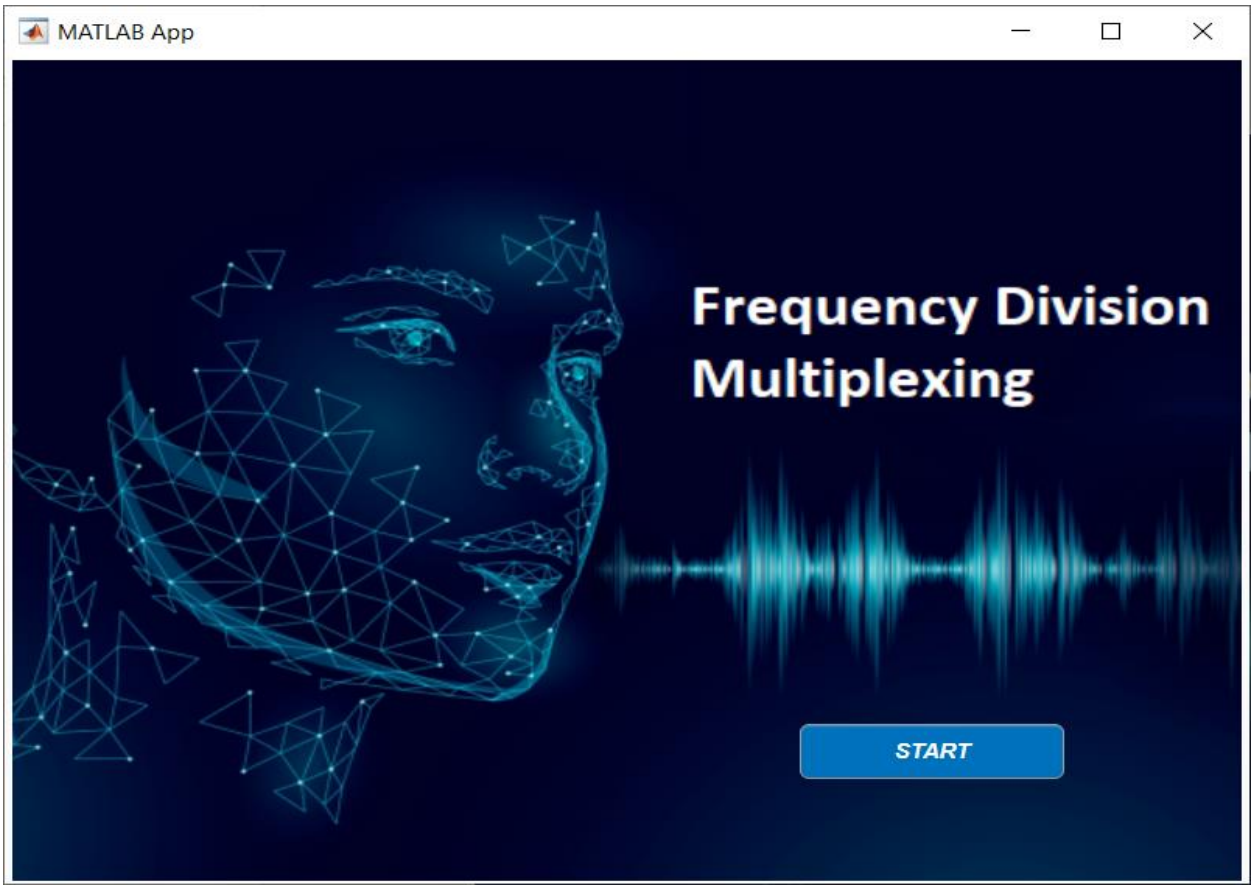
To make this programme user-friendly, we have used Matlab App-designer. Hence, the programme can be executed by a user with little experience on Matlab.

The app plots time domain and frequency domain in every step which will help the user to understand the process clearly. Further, the user can also listen to the output signal after demodulation. So, the app helps the user to acquire primary idea about signal processing and signal transmitting.

Detail overview of the project:

i) Cover:

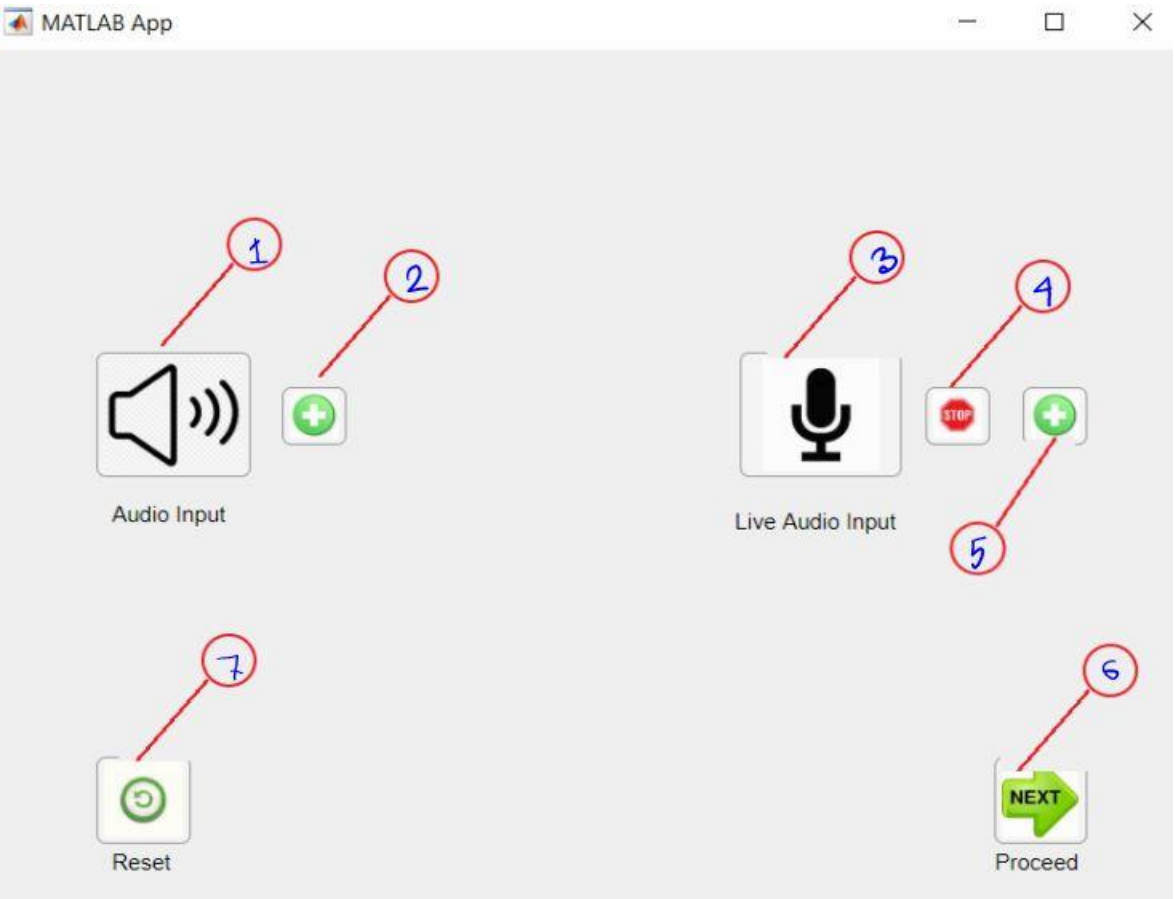
When we will click on 'Frequency_division_multiplexing.mlapp', the following window will pop-up which is the cover of the app.



The window only contains one interactive button, named ‘START’. Pressing the ‘START’ button leads us to the input window.

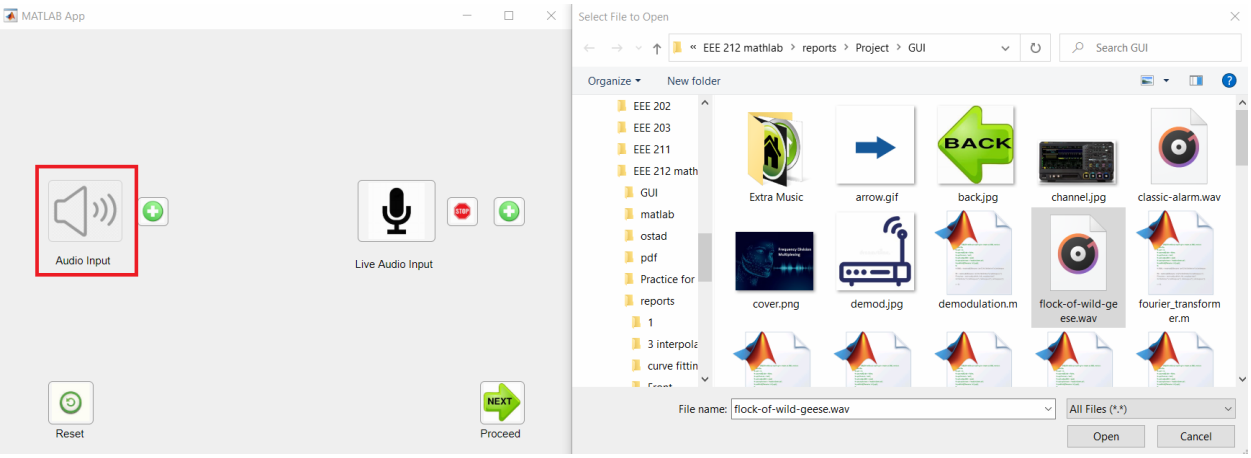
ii) Input window:

The app can take several audio files as input in the input window. But because of some shortcomings of device, more than two inputs can cause undesirable results. The input window has two alternative ways of taking input. Firstly, the user can choose a saved audio file from the device as an input. Also, there is a feature of storing instant voice record instantly, which will be used in further procedure.



1) ‘Audio input’ button:

User can give audio file according to his choice from the device using this button.



Command executed:

```
function audioPushed(app, event)
            set(app.audio,'enable','off')        % To disable after first push
            disp(" your data has been stored ");
            disp(" press + to store more ");

            [file,path]=uigetfile();
            text=strcat(path,file);               % To open file manager
            [y,fs]=audioread(text);               % Reading audio files

            %Resampling sampling rate

            [P,Q] = rat(200000/fs);
            y = resample(y,P,Q);
            fs=200000;

            app.signal_stored = [ app.signal_stored, y ];       % Storing signal
data
            app.frequncy_stored = [ app.frequncy_stored, fs ];  % Storing
sampling rate

            [ size_of_input, ~ ] = size( y );

            app.Input_size=[app.Input_size, size_of_input];

        end
```

2) 'Plus' button:

User can add another audio file pressing the 'Plus' button.

Command executed:

```
function Plus_audioPushed(app, event)

            [file,path]=uigetfile();
            text=strcat(path,file);               % To open file manager
            [y,fs]=audioread(text);               % Reading audio files

            % Changing sampling rate

            [P,Q] = rat(200000/fs);
            y = resample(y,P,Q);
            fs=200000;

            [ size_of_matrix, ~ ] = size( app.signal_stored );
            [ size_of_input, ~ ] = size( y );
```

```
            app.Input_size=[app.Input_size, size_of_input];

            %% Stacking signal data , sampling rate , size , signal data number

            if size_of_matrix > size_of_input
                y( size_of_input+1 : size_of_matrix, : ) = 0;
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];

            elseif size_of_matrix < size_of_input
                app.signal_stored ( size_of_matrix+1 : size_of_input, : ) = 0;
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];
            else
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];
            end

            disp(" your data has been stored ");
            disp(" press + to store more ");

        end
```

3) ‘Live audio input’ button:

After pressing this, recording starts and user can record sound instantly.

Command executed:

```
function live_audioButtonPushed(app, event)
            set(app.live_audio,'enable','off')
            set(app.audio,'enable','off')
            disp('Recording start');

            fs=200000;                                          % Setting sampling
rate
            noc=2;
            nob=8;
            app.recObj = audiorecorder(fs,nob,noc);
            record(app.recObj);                                 % Start recording

            disp('Recording complete');

        end
```

4) ‘Stop’ button:

The button stops recording.

Command executed:

```
function Stop_inputButtonPushed(app, event)
            stop(app.recObj);                           % Stop recoring
            y=getaudiodata(app.recObj);                 % Storing signal data
            fs=200000;

            [ size_of_matrix, ~ ] = size( app.signal_stored );
            [ size_of_input, ~ ] = size( y );

            app.Input_size=[app.Input_size, size_of_input];

            %% Stacking signal data , sampling rate , size , signal data number

            if size_of_matrix > size_of_input || length(app.Input_size)==1
                y( size_of_input+1 : size_of_matrix, : ) = 0;
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];

            elseif size_of_matrix < size_of_input
                app.signal_stored ( size_of_matrix+1 : size_of_input, : ) = 0;
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];
            else
                app.signal_stored = [ app.signal_stored, y ];
                app.frequncy_stored = [ app.frequncy_stored, fs ];
            end
            disp('Your data has been stored');
            disp('Press + to add more');


        end
```

5) ‘Plus’ button with live input:

The app takes preparation for next live audio. And pressing ‘Live audio input’ button the next recording can be recorded.

Command executed:

```
function Plus_live_inputButtonPushed(app, event)
            set(app.live_audio,'enable','on')
        end
```

6) ‘NEXT’ button:

The app proceeds to next step. While doing so, it saves the matrix containing signal data, sampling rate for the signals and size of data entries of each input. Besides, the signals were modulated and stacked in a single matrix, we also calculated the data of the signal passing through the channel. The matrixes are saved in a temporary mat file.

Command executed:

```
function next_buttonPushed(app, event)
%% Saving data in a temporary file
s = struct('signal_stored',
app.signal_stored,'frequncy_stored',app.frequncy_stored,'Input_size',
app.Input_size);
            save('temp.mat', '-struct', 's');

            mod_signal_stored=zeros(size(app.signal_stored));

            % Modulation

            for i=1:length(app.frequncy_stored)

                n = 2*i-1;
                w = 25000 + (i-1)*50000;
                mod_signal_stored(:,n:n+1) = modulation(
app.signal_stored(:,n:n+1) , 200000, w);

            end

            %% Saving data in a temporary file

            s = struct('mod_signal_stored', mod_signal_stored);
            save('temp2.mat', '-struct', 's');

            %% Forming channel signal by adding modulated signal
```

```
            channel_sig=mod_signal_stored(:,1);

            for i=2:length(app.frequncy_stored)

                n=2*i-1;

                channel_sig=channel_sig + mod_signal_stored(:,n);

            end

            s = struct('channel_sig', channel_sig);
            save('temp3.mat', '-struct', 's');

            Frequency_division_multiplexing_start;          % To go next page
        End
```

The function called ‘modulation’ is defined below:

```
function mod_sig = modulation( sig , fs, w)

    % w is shifting frequency in time domain
    % size takes the index number as input;
    % fs is samling frequency of input signal
    % del_t denotes time duration of two consecutive signal

    [ sig_size, ~] = size(sig);
    del_t = 1 / fs;
    mod_sig = zeros ( sig_size, 2);

    for i = 1 : sig_size
        mod_sig( i, 1) = sig( i, 1) * cos( 2*pi*del_t*i*w);
    end

    mod_sig(:,2)=mod_sig(:,1);
end
```

7) ‘Reset’ button:

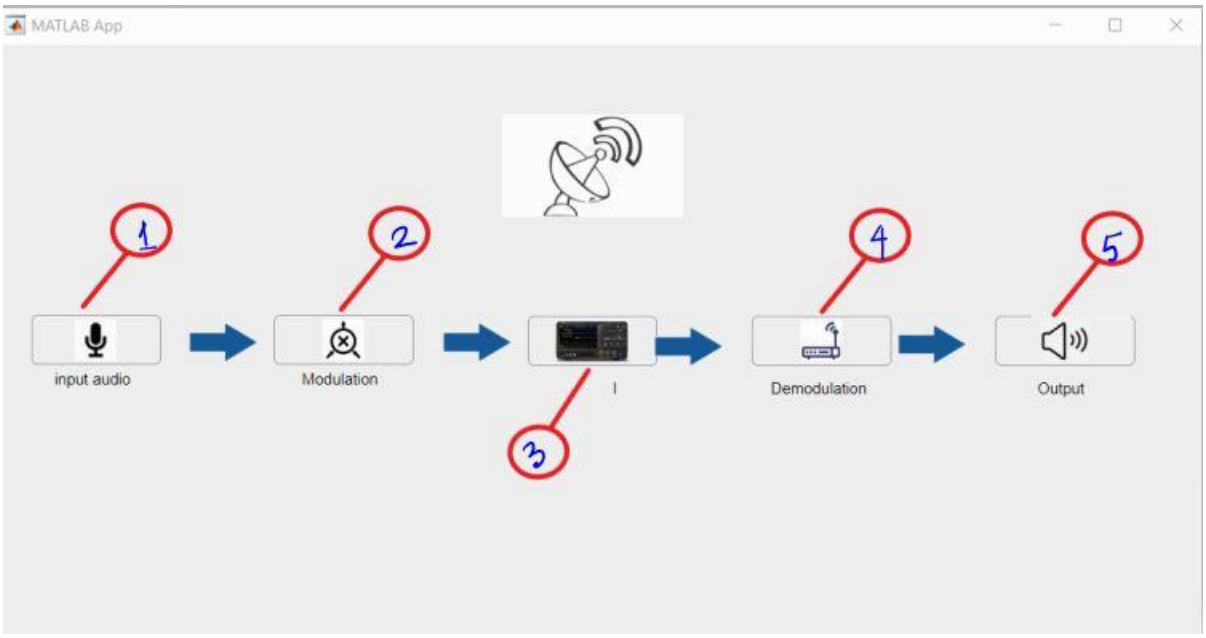
This button deletes the data which was previously stored and repeats the input process from the start.

Command executed:

```
function ResetPushed(app, event)
            % Deleting previous data

            app.frequncy_stored =[];
            app.Input_size = [];
            app.signal_stored = [] ;

            set(app.live_audio,'enable','on')
            set(app.audio,'enable','on')

            disp('Your data has been reset');
        end
```

iii) Home page:

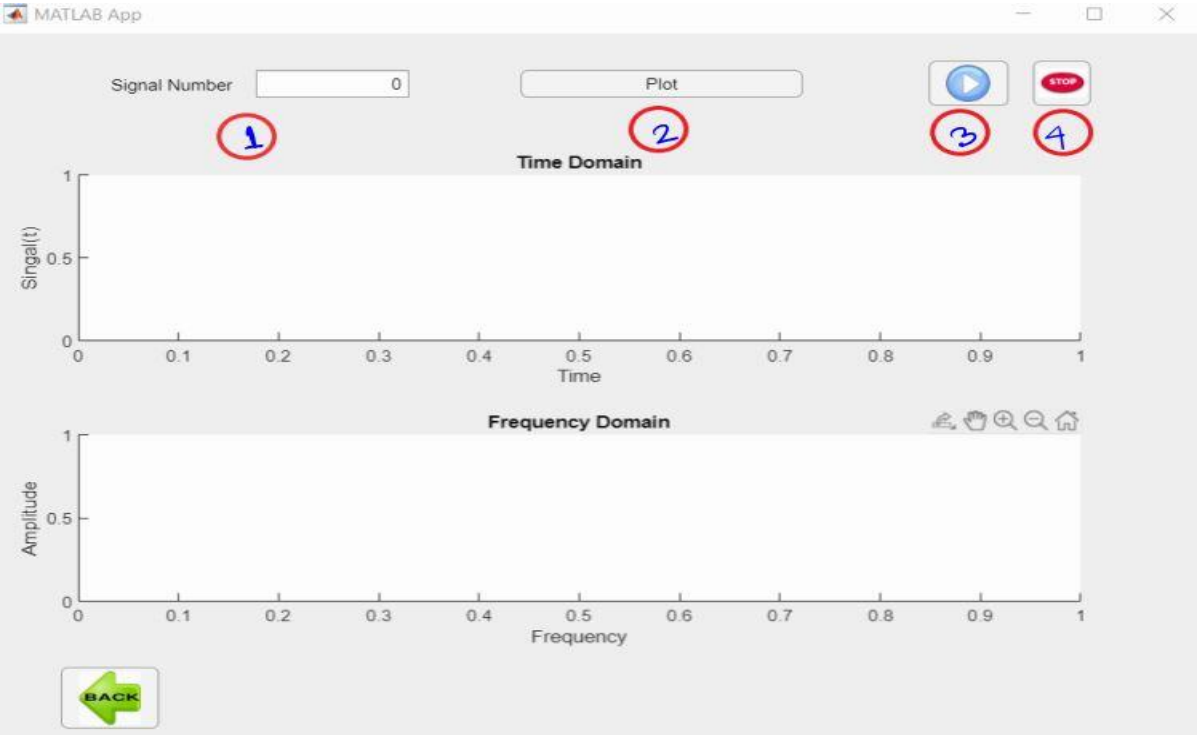
Pressing the ‘NEXT’ button placed in the bottom right corner of the input window leads us to the window shown below.



The window has five buttons respectively ‘Input audio’, ‘Modulation’, ‘Channel’, ‘Demodulation’ and ‘Output’.

a) 'Input audio' button:

Pressing the 'Input audio' button the following window will pop out.



The user has to input signal number and then pressing 'Plot' will give the time domain and frequency domain plot of the respective signal. Also, there is a 'Play' which plays the audio signal and 'Stop' button force-stop the audio at any moment. Again, changing signal number, using 'Plot' and 'Play' button the user can observe the desired signal.

1) 'Signal number' edit field:

The app stores the signal number which we want to analyze.

Command executed:

```
  function SignalNumberEditFieldValueChanged(app, event)
            value = app.SignalNumberEditField.Value;        % Taking the Signal
Number
            if true
                app.a=value;
            end
```

```
        end
```

2) ‘Plot’ button:

Plots the time domain and frequency domain of the desired signal.

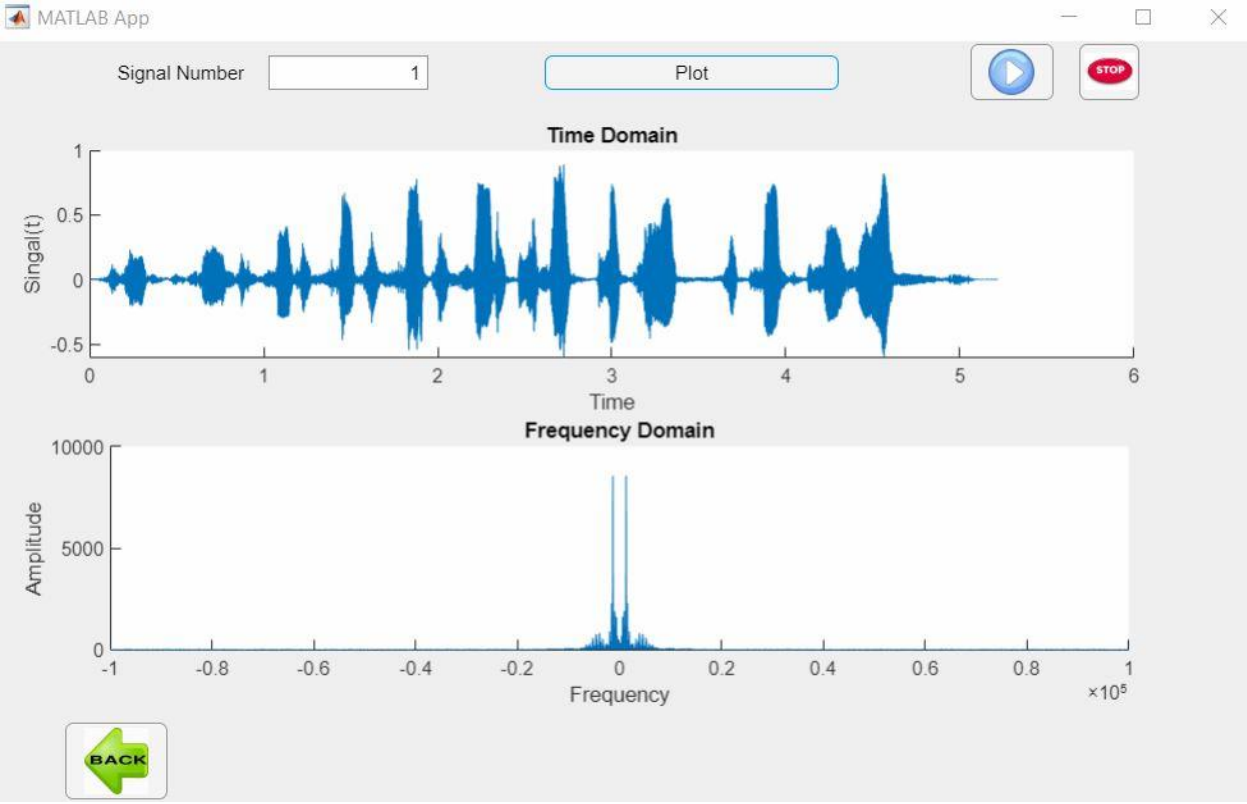
Command executed:

```
function PlotButtonPushed(app, event)
            % Loading temporary file and data
            load('temp.mat','signal_stored','frequncy_stored', 'Input_size');

            i = 2*app.a-1;
            y = signal_stored(1:Input_size(app.a),i);
            t = Input_size(app.a)/frequncy_stored(app.a);       % Total time
            x = linspace(0,t,Input_size(app.a));
            plot(app.UIAxes, x ,y);                             % Plot Time
Domain

            %% Creating 2^n by 1 matrix

            n = log2(Input_size(app.a));
            n = ceil(n);
            x = zeros( 2^n,1);
            x( 1:Input_size(app.a), 1) = y;


            y_fft = my_fft(x);                                  % Fourier
Transformation


            L = length(y_fft);

            f = frequncy_stored(app.a)*((-L/2) : (L/2)-1 )/L;   % Frequency
labeling
            plot(app.UIAxes_2, f, fftshift(abs ( y_fft ) ) );   % Plot Frequency
Domain

        end
```

A recursive function named my_fft is used here which is defined below.

```
function ft_mat = my_fft(mat)
    %only works if N = 2^k
    N = length(mat);

    %divide the given matrix into even and odd rows.
    mat_odd = mat(1:2:end);
    mat_even = mat(2:2:end);

    %recursion
    if N>=8
        mat_odd = my_fft(mat_odd);
        mat_even = my_fft(mat_even);
        ft_mat = zeros(N,1);
        Wn = exp(-1i*2*pi*((0:N/2-1)')/N);
        tmp = Wn .* mat_even;
        ft_mat = [(mat_odd + tmp);(mat_odd -tmp)];
    else
        if N == 2
            ft_mat = [1 1;1 -1]*mat;
        elseif N == 4
            ft_mat = [1 0 1 0; 0 1 0 -1i; 1 0 -1 0;0 1 0
1i]*[1 0 1 0;1 0 -1 0;0 1 0 1;0 1 0 -1]*mat;
        else
            error('N not correct.');
        end
    end
end
```

After pressing the ‘Plot’ button these graphs will appear. Here, we have used a signal which was stored in the input window.



3) ‘Play’ button:

The user can listen to the signal which is being analyzed.

Command executed:

```
function PlayPushed(app, event)
            set(app.Play,'enable','off')        % Disable after one push

            %% Load temporary file and data
            load('temp.mat','signal_stored','frequncy_stored', 'Input_size');

            i = 2*(app.a)-1;
            t = Input_size(app.a)/frequncy_stored(app.a);       % Total time

            y = signal_stored(1:Input_size(app.a),i:i+1);       % Storing signal
            fs = frequncy_stored(app.a);                        % Sampling rate

            sound(y,fs);                                        % Play the music
```

```
            pause(t);                                   % Pause the play
push button till sound end
            set(app.Play,'enable','on')                 % Enable

        end
```

4) ‘Stop’ button:

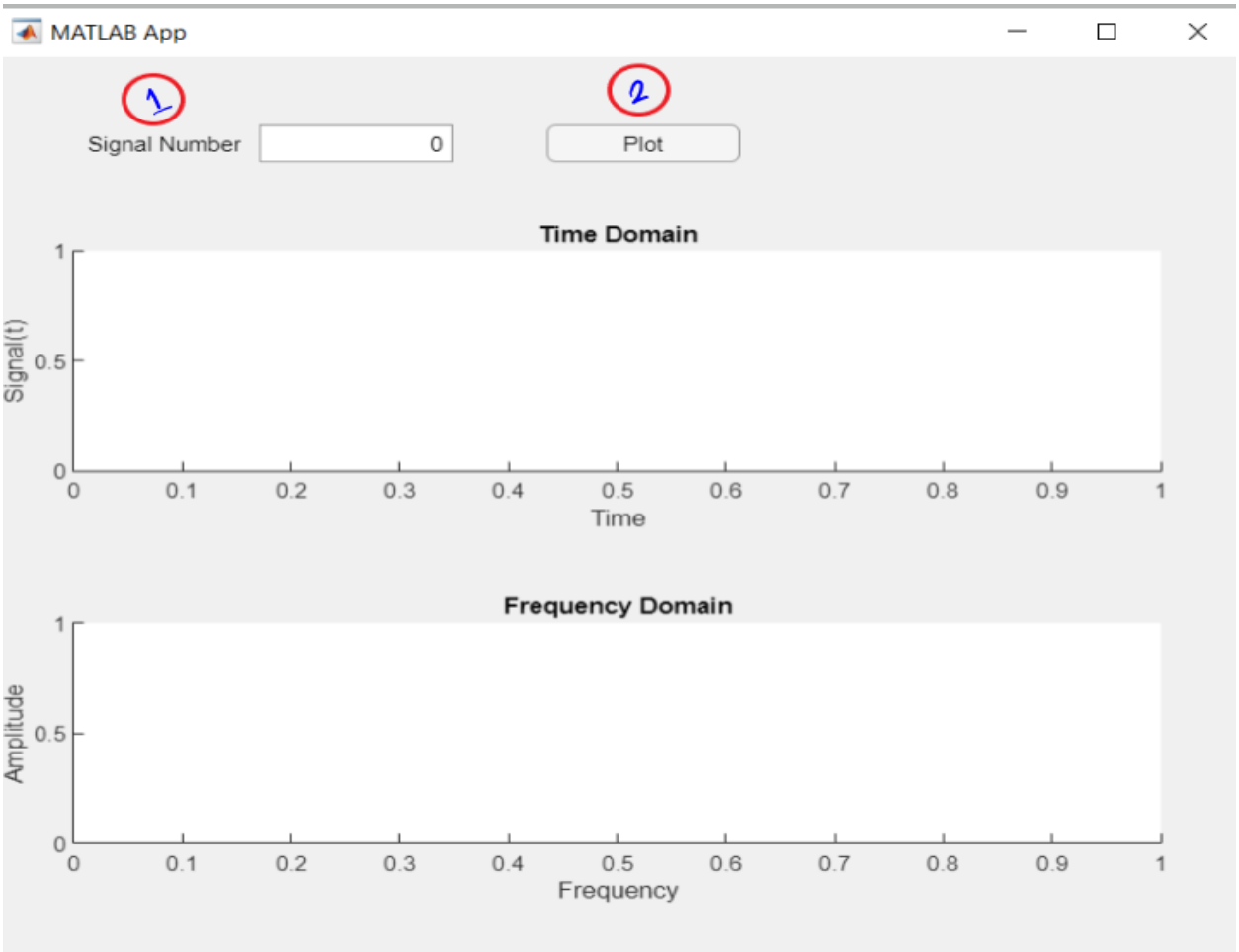
The button stops the audio file which is being played pressing the ‘Play’ button.

Command executed:

```
function StopPushed(app, event)
            clear sound;                                % To stop the
sound
            set(app.Play,'enable','on')
        end
```

b) 'Modulation' button:

Pressing the 'Modulation' button the following window will pop out.



Mentioning the signal number and after pressing 'Plot', time and frequency domain plot will appear.

1) 'Signal number' edit field:

The app stores the signal number which we want to analyze.

Command executed:

```
function SignalNumberEditFieldValueChanged(app, event)
            value = app.SignalNumberEditField.Value;
            % Taking the Signal Number
            if true
```

```
            app.a=value;
        end
    end
```

2) ‘Plot’ button:

Plots the time domain and frequency domain of the desired signal.

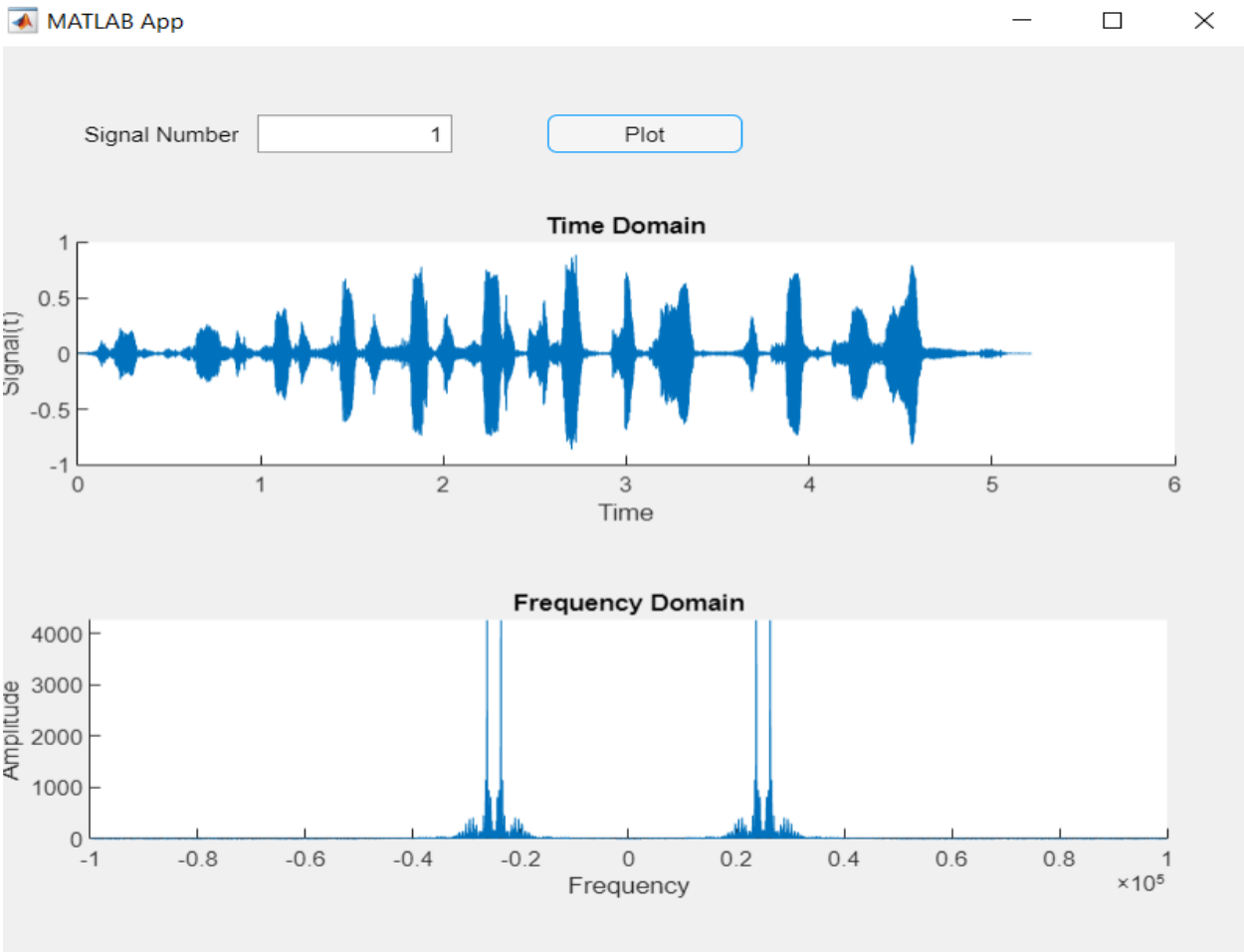
Command executed:

```
function SignalNumberEditFieldValueChanged(app, event)
        value = app.SignalNumberEditField.Value;
        % Taking the Signal Number
        if true
            app.a=value;
        end
    end


    % Button pushed function: PlotButton
    function PlotButtonPushed(app, event)
        %% Loading temporary file and data

        load('temp.mat','frequncy_stored', 'Input_size');

        load('temp2.mat','mod_signal_stored');

        i = 2*app.a-1;
        y = mod_signal_stored(1:Input_size(app.a),i);
        t = Input_size(app.a)/frequncy_stored(app.a);       % Total time
        x = linspace(0,t,Input_size(app.a));
        plot(app.UIAxes, x ,y);


        y_fft = fourier_transformer(y);


        L = length(y_fft);                              % Length of signal

        f = frequncy_stored(app.a)*(-L/2:(L/2)-1)/L;

        plot(app.UIAxes_2, f, fftshift(abs ( y_fft ) ) );   % Ploting
frequency domain


    end
 end
```

A function named my_fft is used here which is defined below.

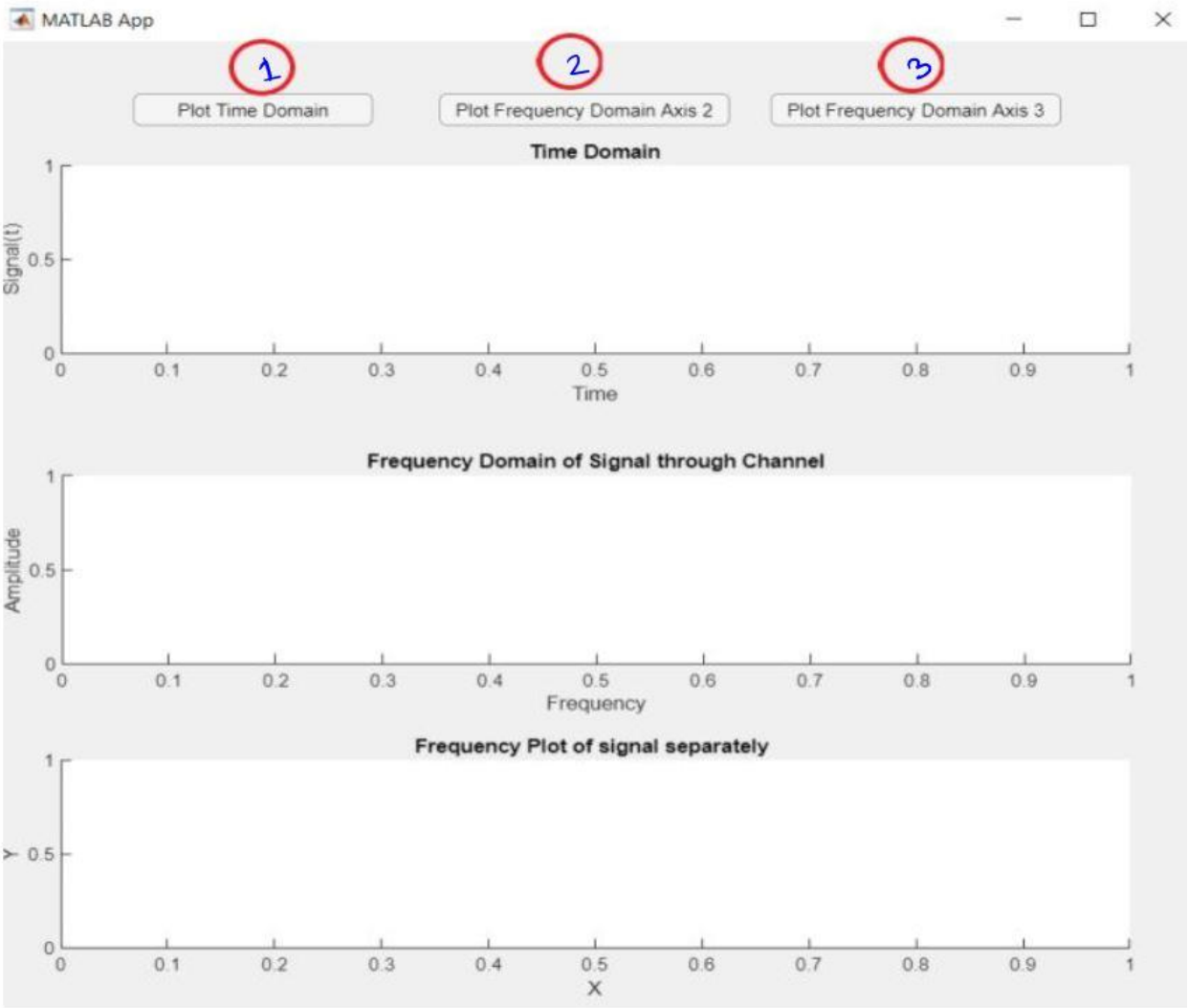
```
function transformed_mat = fourier_transformer(given_mat)
[rows,~] = size(given_mat);
n = log2(rows);
n = ceil(n);
x = zeros( 2^n,1);
x( 1:rows, 1) = given_mat;

transformed_mat = my_fft(x);

end
```

The recursive function called in the function named ‘fourier_transformer’ has been mentioned before. After pressing the ‘Plot’ button these graphs will appear. Here, the graph corresponds to the previously stored signal after demodulation.



c) ‘Channel’ button:

Pressing the ‘Channel’ button the following window will pop out.



The ‘Plot Time Domain’ button plots the time domain of the signal passing through the channel in the first graph. The channel signal is a sum of the modulated audio signal. The ‘Plot Frequency Domain Axis 2’ button prints the frequency domain of the signal passing through the channel. Finally, ‘Plot Frequency Domain Axis 3’ shows us the graph of the frequency domain of the modulated signal plotted in the same graph separately.

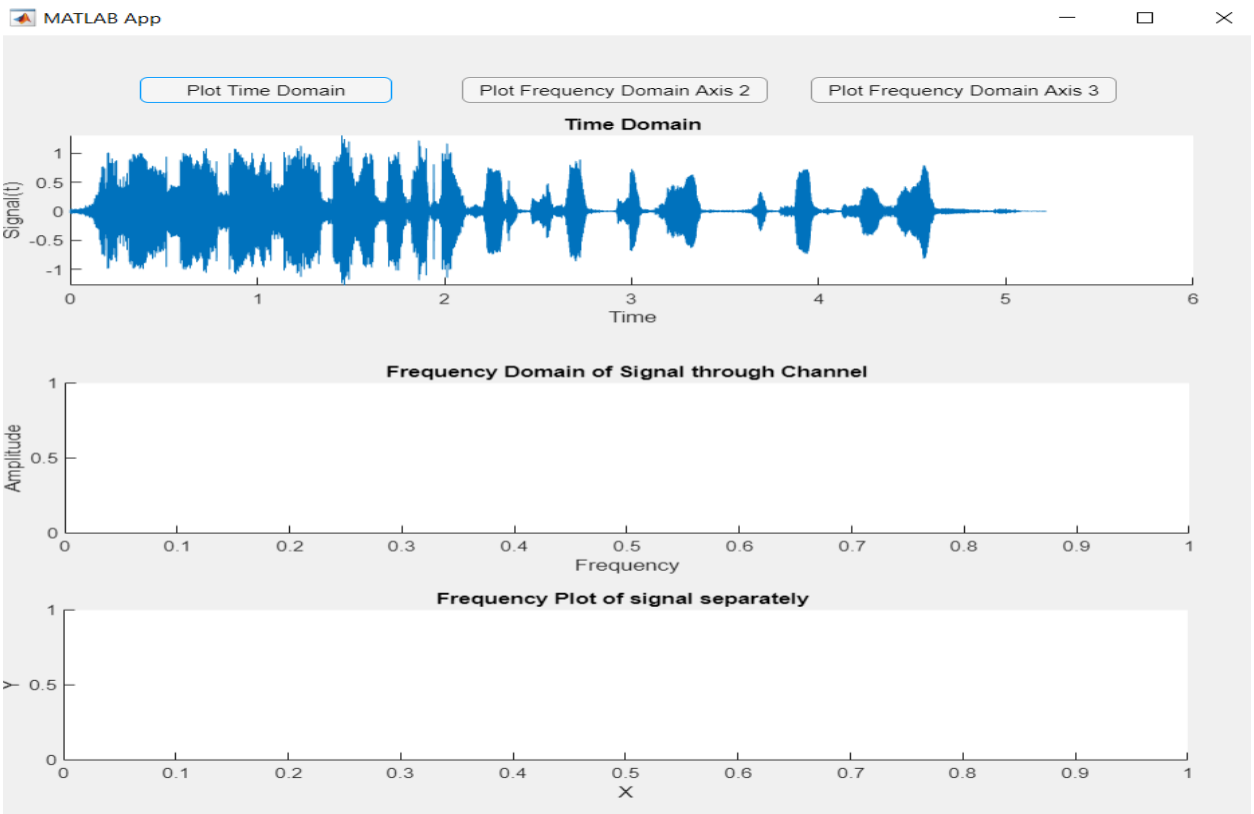
1) 'Plot Time Domain' button:

This button prints the time domain of the signal passing through the channel.

Command executed:

```
function PlotTimeDomainButtonPushed(app, event)
            %% Loading data

            load('temp3.mat','channel_sig');
            [size_channel_sig,~] = size(channel_sig);
            T = size_channel_sig/200000;                % Total time
            t = linspace(0,T,size_channel_sig);

            plot(app.UIAxes, t ,channel_sig);           % Plotting time domain

        end
```

After pressing the button we will observe the following graph. Here, we have stored two signals, after modulation we added the signals. Finally, they are ready for transmission.



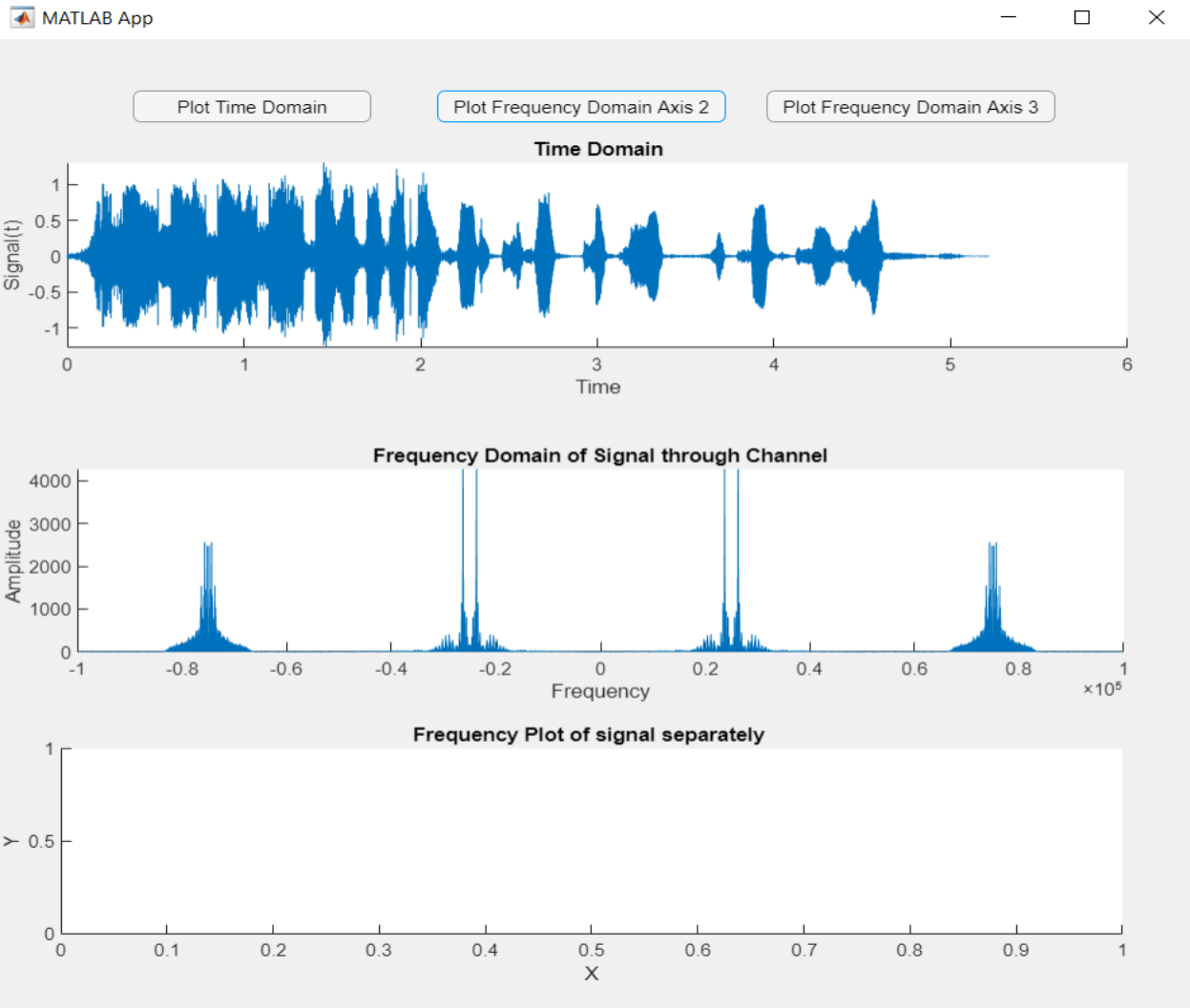
2) ‘Plot Frequency Domain Axis 2’ button:

This button prints the frequency domain of the signal passing through the channel.

Command executed:

```
function PlotFrequencyDomainAxis2ButtonPushed(app, event)
            % Loading data

            load('temp3.mat','channel_sig');

            channel_sig_fre = fourier_transformer(channel_sig(:,1));
%Fourier transform

            L = length(channel_sig_fre);
            f = 200000*(-L/2:(L/2)-1)/L;

            plot(app.UIAxes_2, f ,abs(fftshift(channel_sig_fre)));
%Plotting Frequency domain


        end
```

After pressing this button we will observe the following graph. The graph is the frequency domain of the channel signal, , it will be plotted in the second.



3) 'Plot Frequency Domain Axis 3' button:

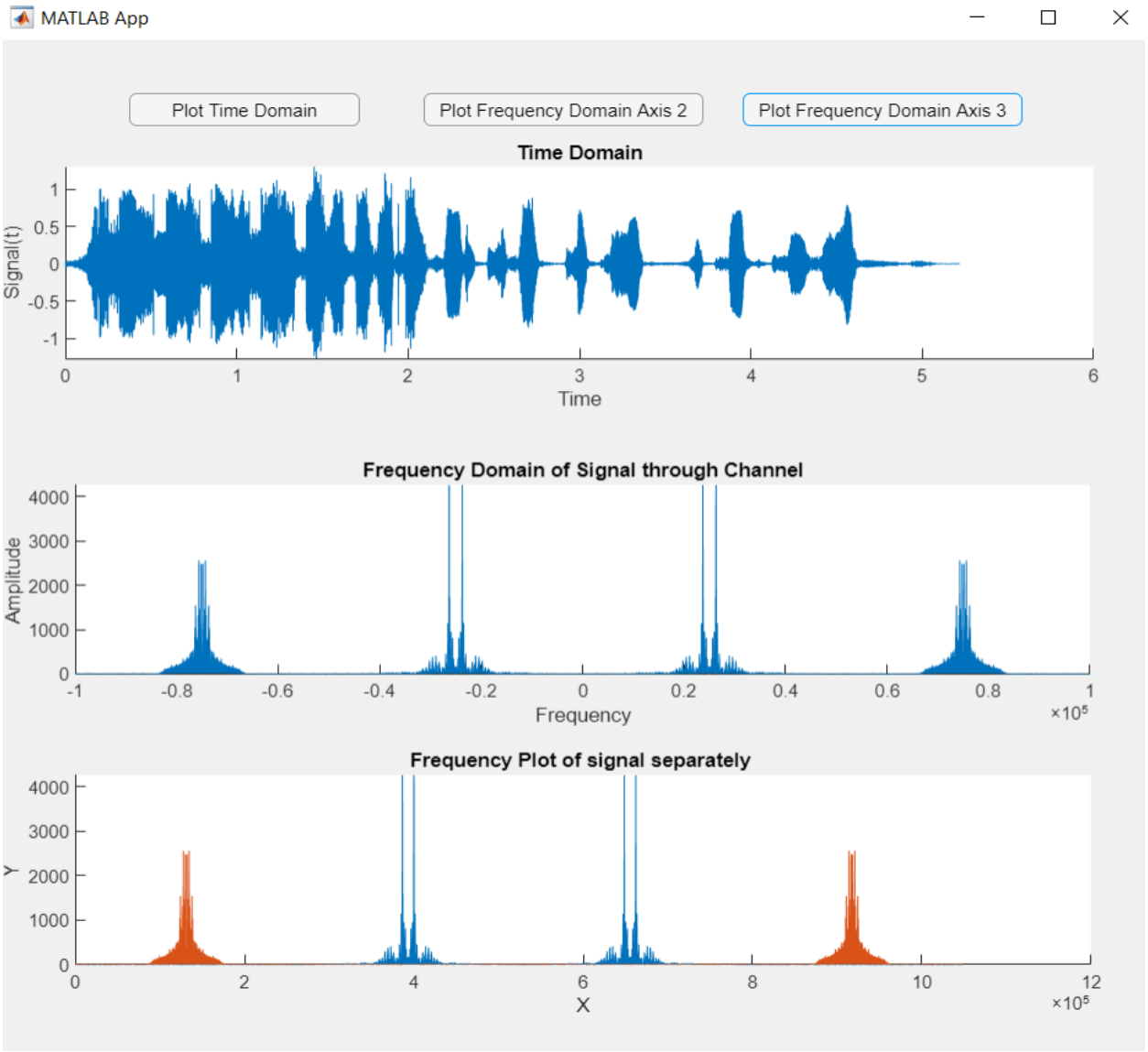
This button prints the frequency domain of the two modulated signals separately.

Command executed:

```
function PlotFrequencyDomainAxis3ButtonPushed(app, event)
            %% Loading data

            load('temp2.mat','mod_signal_stored');

            [~,num_signal] = size(mod_signal_stored);

            %% Ploting signal separately in same graph
```

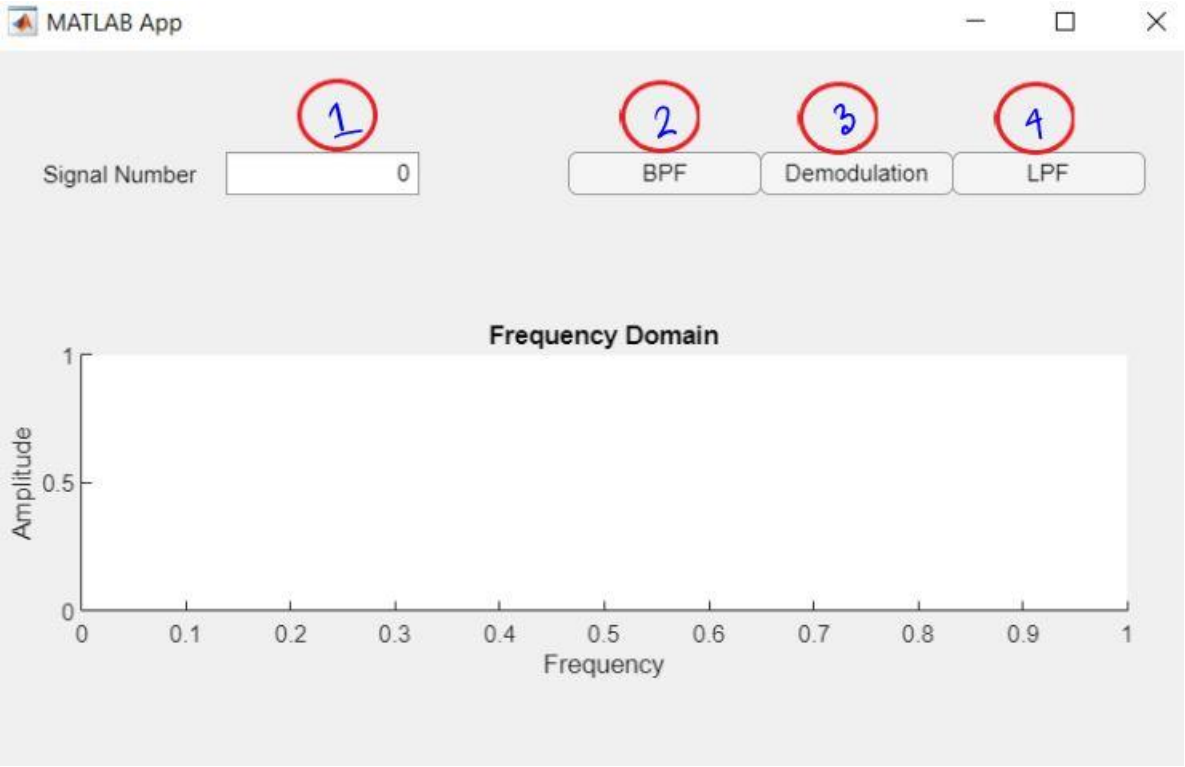
```
            for i= 1: (num_signal/2)

                n = 2*i -1;
                y = fourier_transformer(mod_signal_stored(:,n));
                plot(app.UIAxes_3, abs(fftshift(y)));
                hold (app.UIAxes_3, "on");


            end

            hold (app.UIAxes_3, "off");
        end
    end
```

After pressing this button we will observe the following graph.



d) ‘Demodulation’ button:

Pressing the ‘Demodulation’ button the following window will pop out.



This window helps us to demodulate the channel signal and get our desirable signal back from multiple modulated signal passing simultaneously. At first, we will enter the signal number which we want to demodulate. Then, we pass the signal through band pass signal using ‘BPF’ button, and we will be able to see the result in the graph. After that, we will press ‘Demodulation’ button, this will extract the original information from a carrier wave. The demodulated signal will pass through the low pass filter to remove the high frequency components and it will give us a better output. And this action is completed pressing ‘LPF’ button.

1) ‘Signal number’ edit field:

The app stores the signal number which we want to demodulate.

Command executed:

```
function SignalNumberEditFieldValueChanged(app, event)
            value = app.SignalNumberEditField.Value;        % Signal number
            if true
                app.a=value;
            end
        end
```

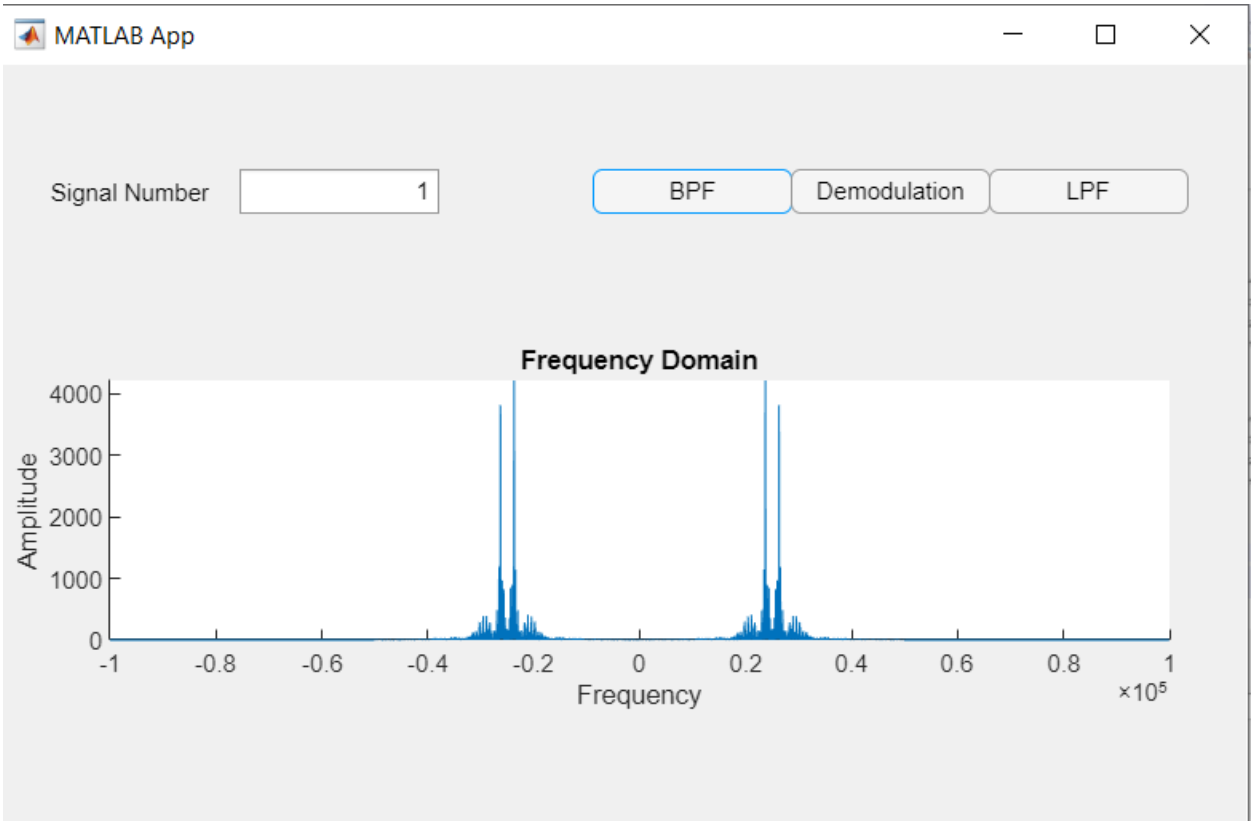
2) ‘BPF’ button:

This button allows to pass the signal with desired frequency.

Command executed:

```
function BPFButtonPushed(app, event)
            %% Loading data
            load('temp3.mat','channel_sig');

            %% Bandpass filter
            app.channel_sig_band = fftshift(fft(channel_sig));
            i = app.a;
            [app.size_channel_sig,~] = size(app.channel_sig_band);
            wc = floor(app.size_channel_sig/4);
            app.channel_sig_band((i-1)*wc+1 : i*wc , 1 ) = 0;
            app.channel_sig_band((4-i)*wc:(5-i)*wc , 1 ) = 0;


            %% Plotting frequency domain of filterred signal
            L = app.size_channel_sig;
            f = 200000*(-L/2:(L/2)-1)/L;

            plot(app.UIAxes, f, (abs(app.channel_sig_band)));


        end
```

After pressing this button the frequency domain of the signal will look like the following graph.



3) 'Demodulation' button:

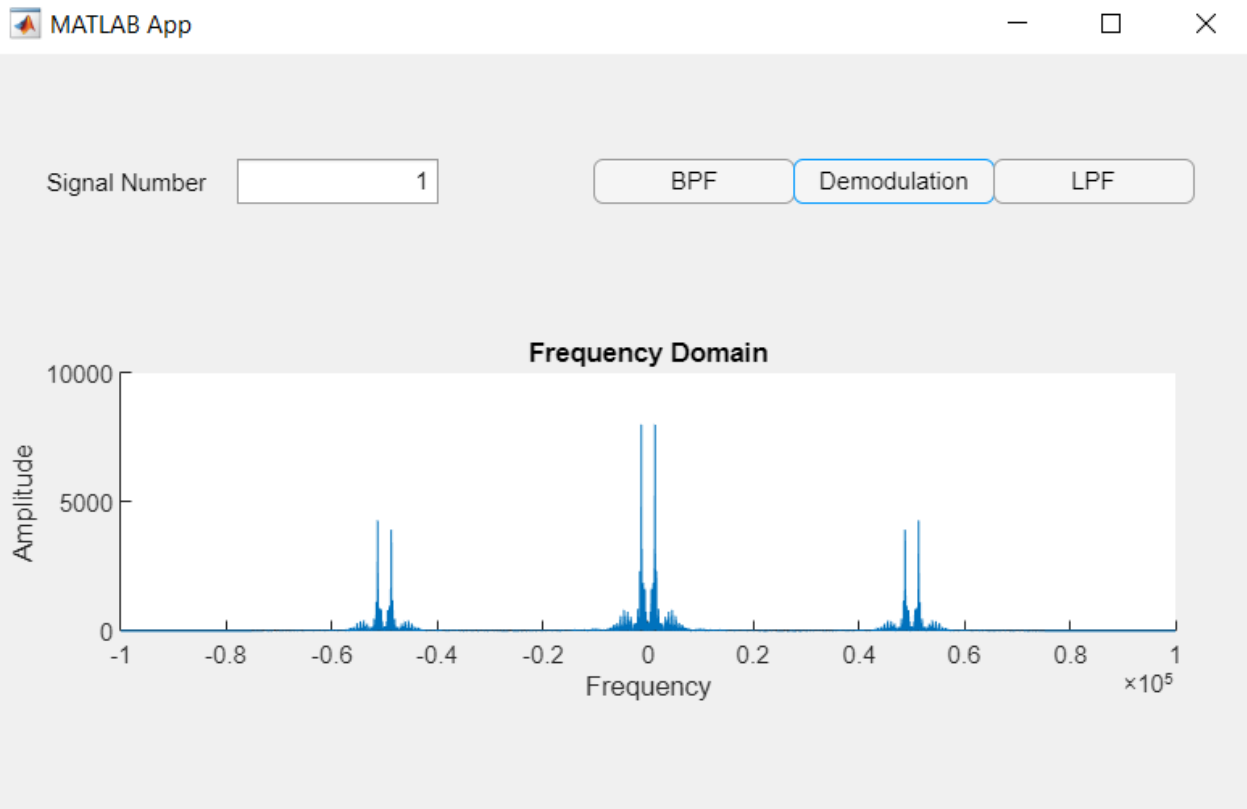
This button extracts the original information from a carrier wave.

Command executed:

```
function DemodulationButtonPushed(app, event)
            %% Demodulating
            i = app.a;
            w = 25000 + (i-1)*50000;
            channel_sig_band_time = ifft(fftshift(app.channel_sig_band));
            channel_sig_band_time_demod = demodulation ( channel_sig_band_time ,
200000, w);
            app.channel_sig_band_freq_demod =
fftshift(fft(channel_sig_band_time_demod ));

            %% Plotting frequency domain of demodulated signal
            L = app.size_channel_sig;
            f = 200000*(-L/2:(L/2)-1)/L;
```

```
        plot (app.UIAxes, f, abs(app.channel_sig_band_freq_demod));

    end
```

The appearing graph will be like following.



4) 'LPF' button:

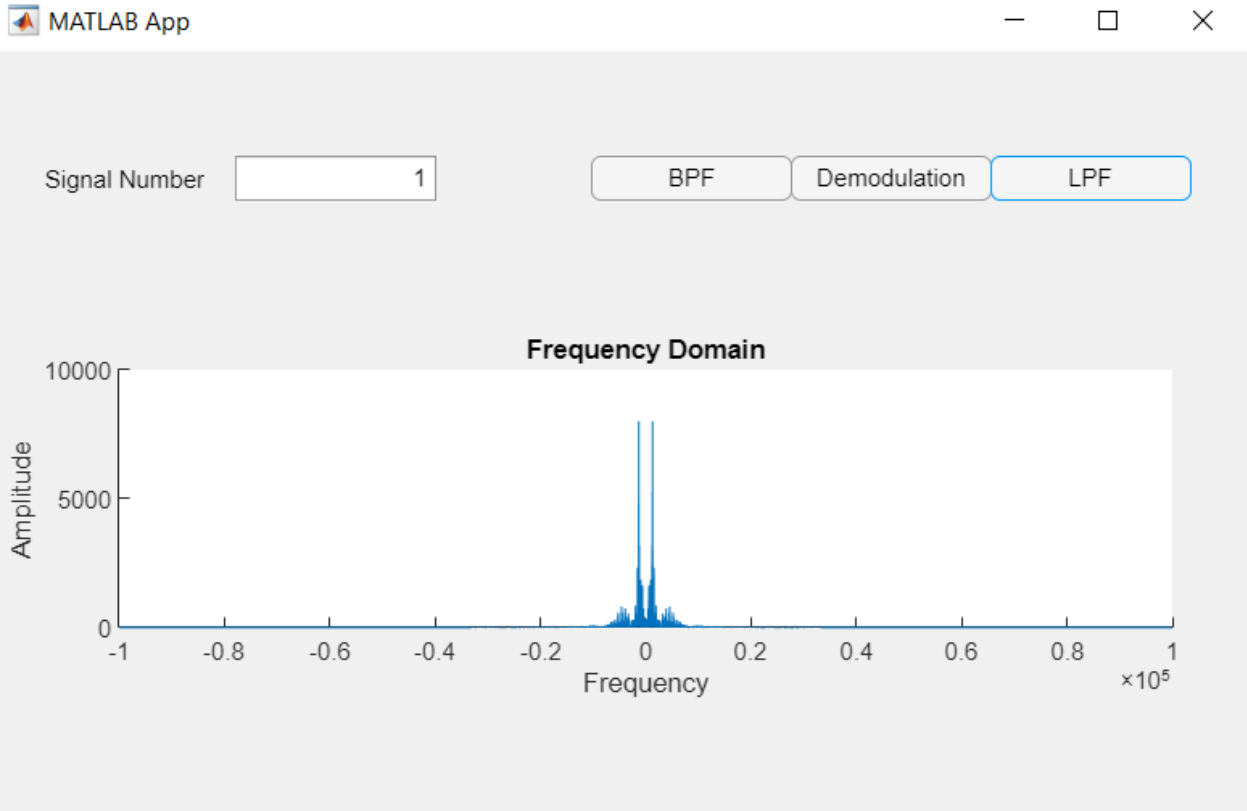
The 'LPF' button, only allows low frequency signals eliminating higher frequency components and functions as a low pass filter.

Command executed:

```
function LPFButtonPushed(app, event)
            %% Low pass filter

            wc = floor(app.size_channel_sig/3);
            app.channel_sig_band_freq_demod (1 : wc , 1 ) = 0;
            app.channel_sig_band_freq_demod ( 2*wc : 3*wc , 1 ) = 0;

            %% Plotting frequency domain of filterred signal
            L = app.size_channel_sig;
```

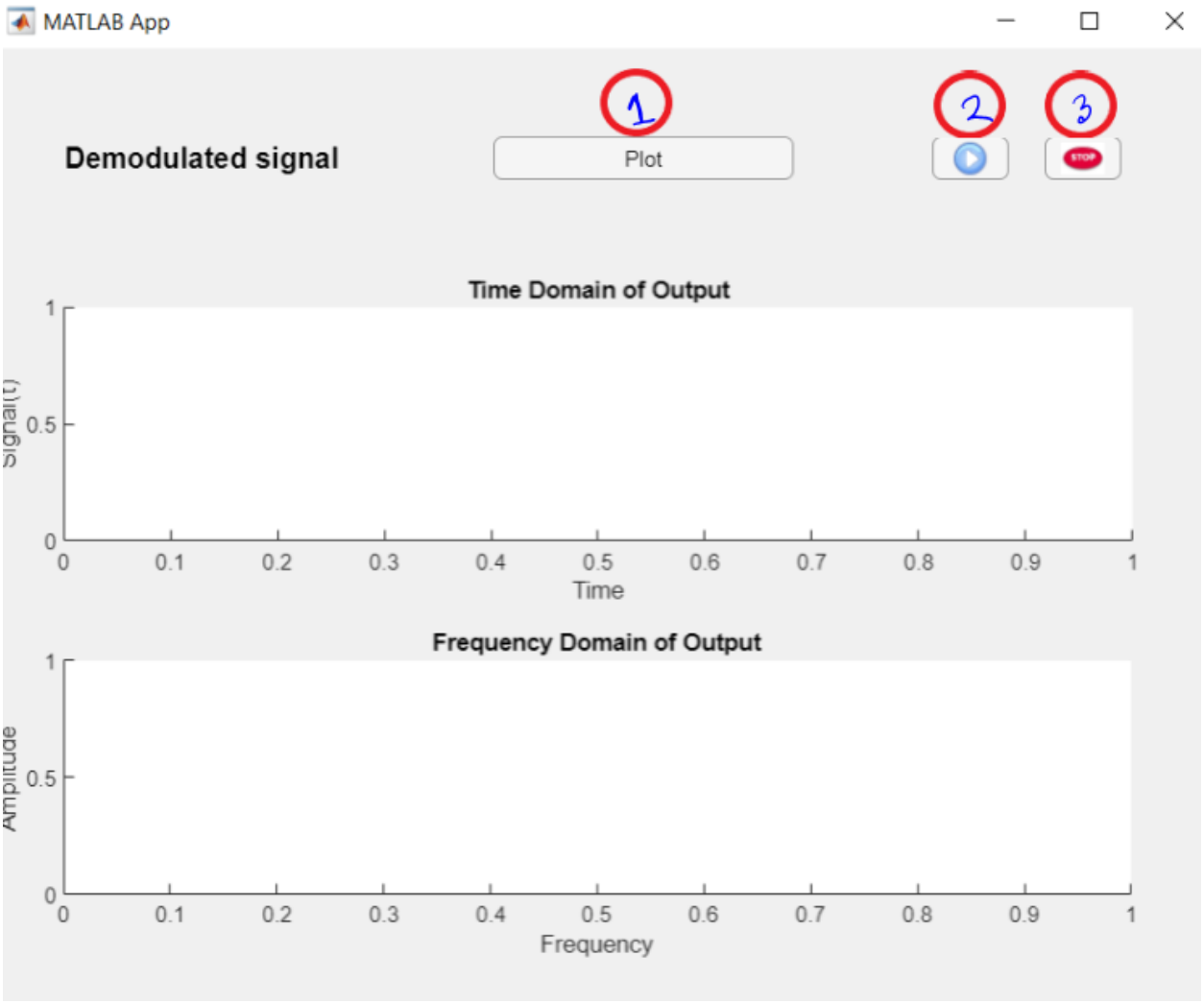
```
            f = 200000*(-L/2:(L/2)-1)/L;

            plot(app.UIAxes, f, abs( app.channel_sig_band_freq_demod ));

            %% Saving frequency of demodulated signal
            s = struct('channel_sig_band_freq_demod',
app.channel_sig_band_freq_demod );
            save('temp4.mat', '-struct', 's');
        end
```

A frequency domain plot will show up where only low frequency is allowed, the maximum frequency allowed is about 20kHz.



e) ‘Output’ button:

Pressing the ‘Output’ button the following window will pop out.



By pressing the ‘Plot’ button we will see the time and frequency domain of the signal which was obtained through demodulation process previously. There is also a ‘Play’ button which allows us to hear the sound of the audio which will appear at the receiver end. And a ‘Stop’ button is present to stop the audio being played to stop immediately.

1) ‘Plot’ button:

Time domain and frequency domain of the demodulated signal can be

plotted.

Command executed:

```
function PlotButtonPushed(app, event)
            %% loading data
            load('temp4.mat','channel_sig_band_freq_demod');

            %% Inverse fourier transformation
             output_sig = inverse_fourier_transformer
      (fftshift(channel_sig_band_freq_demod));
            T = length(output_sig)/200000 ;
            t = linspace(0 , T, length(output_sig)) ;

            plot(app.UIAxes, t, real( output_sig ));                % Plotting
time domain

            plot(app.UIAxes_2, abs(channel_sig_band_freq_demod ));  % Plotting
frequency domain
        end
```

The functioned called ‘inverse_fourier_transformer’ is defined below:

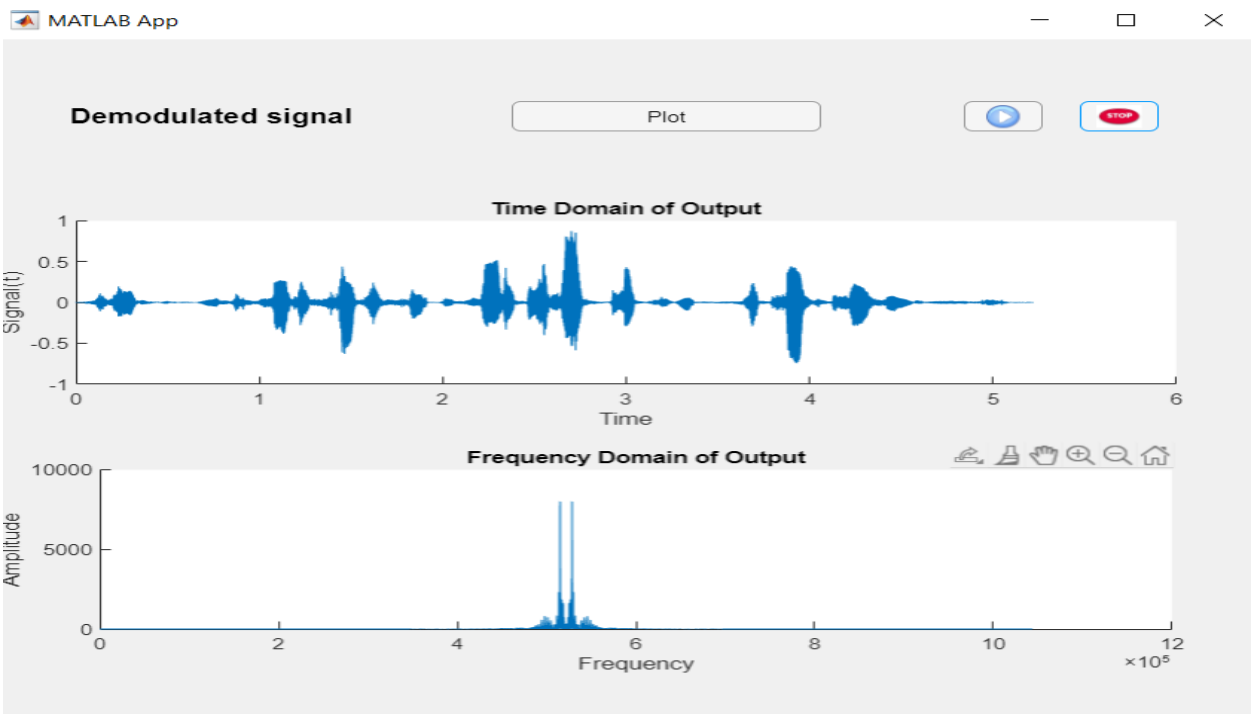
```
function transformed_mat =
inverse_fourier_transformer(given_mat)
[rows,~] = size(given_mat);
n = log2(rows);
n = ceil(n);
x = zeros( 2^n,1);

x( 1:rows,1) = given_mat(:,1);

transformed_mat = my_ifft(x)/(2^n);

end
```

The recursive function named ‘my_ifft’ is used here which is defined below.

```
function ft_mat = my_ifft(mat)
    %only works if N = 2^k
    N = length(mat);
```

```
    %divide the given matrix into even and odd rows.
    mat_odd = mat(1:2:end);
    mat_even = mat(2:2:end);

    %recursion
    if N>=8
        mat_odd = my_ifft(mat_odd);
        mat_even = my_ifft(mat_even);
        ft_mat = zeros(N,1);
        Wn = exp(1i*2*pi*((0:N/2-1)')/N);
        tmp = Wn .* mat_even;
        ft_mat = [(mat_odd + tmp);(mat_odd -tmp)];
    else
        if N == 2
            ft_mat = [1 1;1 -1]*mat;
        elseif N == 4
            ft_mat = [1 0 1 0; 0 1 0 -1i; 1 0 -1 0;0 1 0
1i]*[1 0 1 0;1 0 -1 0;0 1 0 1;0 1 0 -1]*mat;
        else
            error('N not correct.');
        end
    end
end
```

The graph will look like the following.



2) ‘Play’ button:

The user can listen to the demodulated audio using this button.

Command executed:

```
function PlotButtonPushed(app, event)
            %% loading data
            load('temp4.mat','channel_sig_band_freq_demod');

            %% Inverse fourier transformation
             output_sig = ifft(fftshift(channel_sig_band_freq_demod));
            T = length(output_sig)/200000 ;
            t = linspace(0 , T, length(output_sig)) ;

            plot(app.UIAxes, t, real( output_sig ));                % Plotting
time domain

            plot(app.UIAxes_2, abs(channel_sig_band_freq_demod ));  % Plotting
frequency domain
        end
```

3) ‘Stop’ button:

The user can stop playing the audio signal immediately.

Command executed:

```
function StopPushed(app, event)
            clear sound;                    % Stop music
            set(app.Play,'enable','on')
        end
```

Discussion:

The purpose of this project is to help the users to understand the basic concept of Frequency Division Multiplexing. FDM is a very well-established process used for efficient data transmission. This app briefly shows every step of the transmission process. Such as, The time domain graph and frequency domain after every step. As a result, the user can visualize and get the graphical idea of every steps of transmission process. So, this app may be handy for the students of primary signal course. Besides, this app maybe used for modulating and demodulating in a small range.

Though the app is best suited for two audio signals, we can take several inputs and process them accordingly. But, due to some shortcomings of the device and Matlab software, we cannot successfully program the app. The problem arises when there are more than two input signals. The reason of the problem is, the frequency of the carrier wave increases proportionally with the number of inputs. For higher input number, the modulated signals have very high frequency. And so, if we want to demodulate the signal we need higher sampling rate. As per Nyquist theorem, sampling rate must be double of the maximum frequency. But in Matlab we managed to increase the sampling rate to 200k/s. And with this sampling rate we can successfully demodulate signals having less than 100kHz frequency. So, if a third audio signal is inputted, then the frequency of the modulated signal exceeds 100kHz limit. As well as, if we modulate the signal at low frequency, some overlap in frequency occurs. Then, the output audio signal becomes distorted and unexpected result appears.

But if we are able to increase the sampling rate of the audio inputs, we

can use this app widely. We will be able to process multiple signals at the same moment and simulate how massive number of signals are transmitted in real life scenario.

We have tried to make this app as better as possible, in a limited period of time. Though, there are some constraints, the app is quite useful in enhancing the basic idea of frequency division multiplexing and learning this complex topic thoroughly.