Final Project Report

Kyrillos Gayed Ishak	6804
Ahmed Hassan Abdelkader	6737
Alaa Hossam Abdelmawla	6750
Amr Abdelsamee Youssef	7126
Mohamed Saeed Abdelhafez	7034

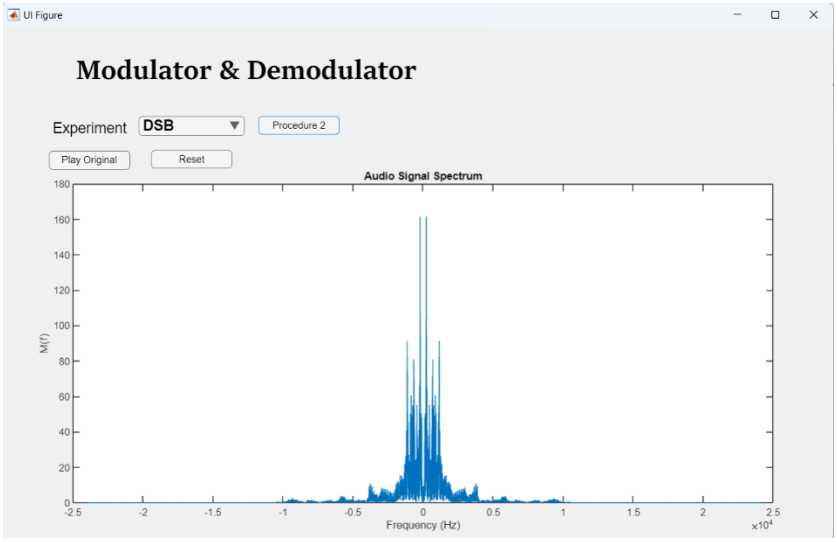
1- Double Side Band Modulation:

Procedure:

1- We Plotting the spectrum of the audio signal
Code:

```
if app.counter == 1
        app.fs_y = app.n / app.ty;
        app.fshift = (app.fs_y / 2) * linspace(-1, 1, app.fs_y);
        app.freq_y = fftshift(fft(app.y));
        app.fft_y = fftshift(fft(app.y, numel(app.fshift)));

        % Plotting the spectrum of the audio signal
        plot(app.UIAxes, app.fshift, abs(app.fft_y));
        xlabel(app.UIAxes, 'Frequency (Hz)');
        ylabel(app.UIAxes, 'M(f)');
        title(app.UIAxes, 'Audio Signal Spectrum');
    end
```

Output:

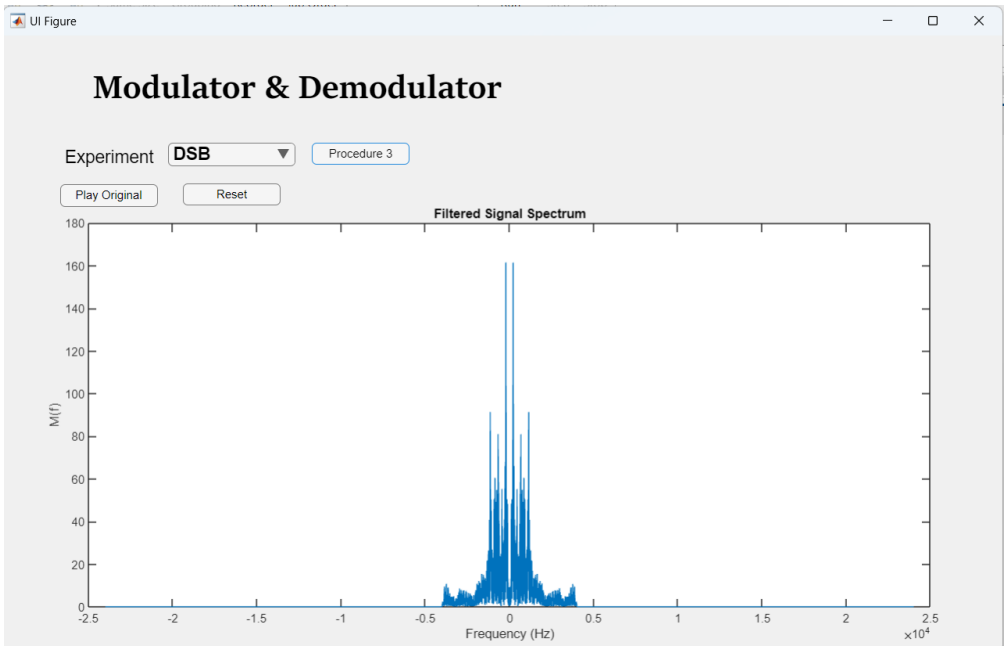


2- Get bandlimit from our Signal to remove unwanted frequencies with ideal filter from 0 to 4KHz & the actual signal used for sound & Filtered signal used for plotting

Code:

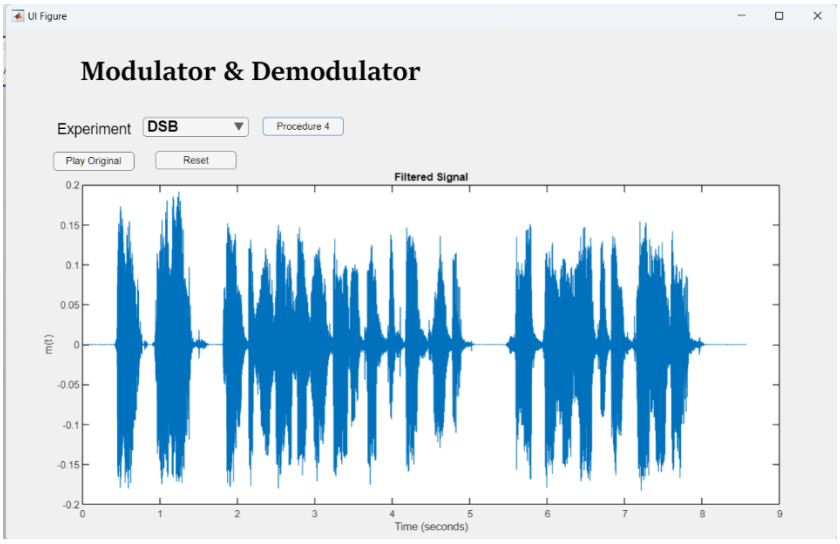
```
%%------------< EXP 1.2 >------------
    if app.counter == 2
        app.band_limit = floor(app.fn + app.ty);
        app.plotting_filter = cat(1, zeros([app.Fs / 2 - 4e3, 1]), ones([4e3, 1]),
ones([4e3, 1]), zeros([app.Fs / 2 - 4e3, 1]));
        app.filter = cat(1, zeros([app.n / 2 - app.band_limit, 1]),
ones([app.band_limit, 1]), ones([app.band_limit, 1]), zeros([app.n / 2 -
app.band_limit, 1]));
        app.filtered_signal = app.freq_y .* app.filter;
        app.plot_filtered = app.fft_y .* app.plotting_filter;
        plot(app.UIAxes, app.fshift, abs(app.plot_filtered));
        xlabel(app.UIAxes, 'Frequency (Hz)');
        ylabel(app.UIAxes, 'M(f)');
        title(app.UIAxes, 'Filtered Signal Spectrum');
        clear app.plot_filtered app.plotting_filter;
    end
```

Output:



3- Plotting the filtered signal in time domain.

```
%%------------< EXP 1.3 >------------
 if app.counter == 3
     app.time_signal = ifftshift(app.filtered_signal);
     clear app.filtered_signal;
     app.time_signal = real(ifft(app.time_signal));
     app.t = linspace(0, app.n / app.Fs, app.n);

     % Plotting the filtered signal in time domain
     plot(app.UIAxes, app.t, app.time_signal);
     xlabel(app.UIAxes, 'Time (seconds)');
     ylabel(app.UIAxes, 'm(t)');
     title(app.UIAxes, 'Filtered Signal');
 end
```



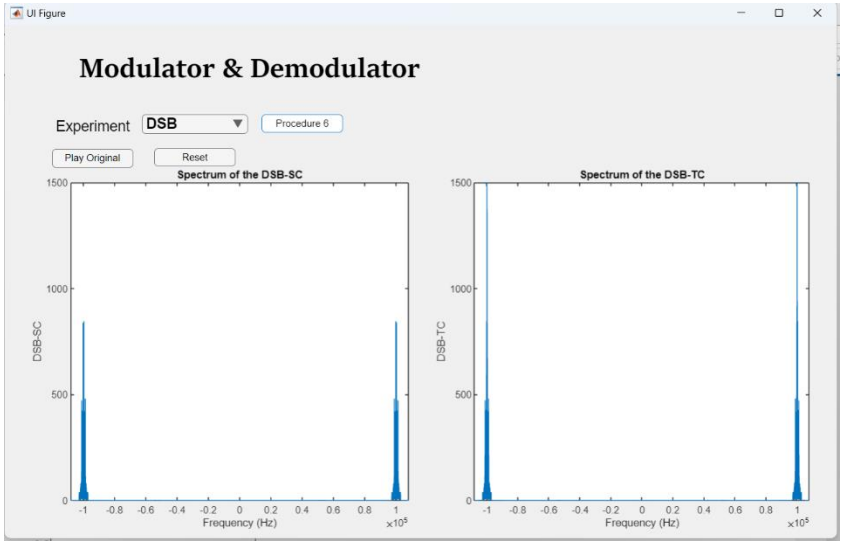
4- Sound of the filtered signal of BW=4KHz Which is not different from the original by much

Code:

```
%%------------< EXP 1.4 >------------
if app.counter == 4
   %
     app.Label_3.Text = 'Playing sound please wait.....';
     sound(app.time_signal, app.Fs);
     pause(8.5);
     app.Label_3.Text = '';
end
```

5- We Resample fs to 5fc & Get new length of signal & Get new time line and frequency & Create Carrier signal & Create DSB-SC signal & Plotting the DSB-SC signal spectrum & Changing the carrier frequency to be double the maximum of the message & Plotting the DSB-TC signal spectrum

```
if app.counter == 5
        app.fc = 1e5;
        app.m_t = resample(app.time_signal, 5 * app.fc, app.Fs);
        clear app.time_signal;
        app.n = length(app.m_t);
        app.ty = app.n / (5 * app.fc);
        app.fs_y = app.n / app.ty;
        app.fshift = (app.fs_y / 2) * linspace(-1, 1, app.fs_y);
        app.t = linspace(0, app.n / (5 * app.fc), app.n);
        app.c_t = cos(2 * pi * app.fc * app.t)';
        app.DSB_SC = app.m_t .* app.c_t;
        app.F_DSB_SC = abs(fftshift(fft(app.DSB_SC, numel(app.fshift))));
        plot(app.UIAxes2, app.fshift, app.F_DSB_SC);
        xlabel(app.UIAxes2, 'Frequency (Hz)');
        ylabel(app.UIAxes2, 'DSB-SC');
        title(app.UIAxes2, 'Spectrum of the DSB-SC');
        xlim(app.UIAxes2, [-app.fc - 8e3 app.fc + 8e3]);
        ylim(app.UIAxes2, [0 1500]);
        app.Ac = 2 * max(app.m_t);
        app.DSB_TC = (app.Ac + app.m_t) .* app.c_t;
        app.F_DSB_TC = abs(fftshift(fft(app.DSB_TC, numel(app.fshift))));
        plot(app.UIAxes3, app.fshift, app.F_DSB_TC);
        clear app.F_DSB_TC app.F_DSB_SC;
        xlabel(app.UIAxes3, 'Frequency (Hz)');
        ylabel(app.UIAxes3, 'DSB-TC');
        title(app.UIAxes3, 'Spectrum of the DSB-TC');
        xlim(app.UIAxes3, [-app.fc - 8e3 app.fc + 8e3]);
        ylim(app.UIAxes3, [0 1500]);
    end
```

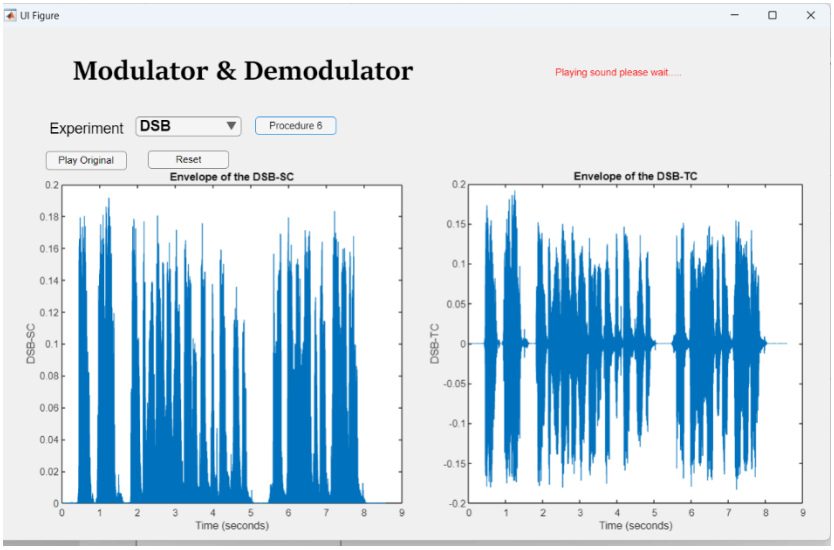


6- Getting the envelope of DSB-SC & Plotting the envelope of the DSB-SC & resampling back to Fs to play the signal & Sound of the DSB-SC after envelope & Getting the envelope of DSB-TC & Plotting the envelope of the DSB-TC & resampling back to Fs to play the signal & Sound of the DSB-TC after envelope

Code:

```
%%------------< EXP 1.6 >------------
if app.counter == 6
    app.env_DSB_SC = abs(hilbert(app.DSB_SC));
    plot(app.UIAxes2, app.t, app.env_DSB_SC);
    xlabel(app.UIAxes2, 'Time (seconds)');
    ylabel(app.UIAxes2, 'DSB-SC');
    title(app.UIAxes2, 'Envelope of the DSB-SC');
    app.Label_3.Text = 'Playing sound please wait.....';
    app.env_DSB_SC = resample(app.env_DSB_SC, app.Fs, 5 * app.fc);
    sound(app.env_DSB_SC, app.Fs);
    pause(8.5);
    app.env_DSB_TC = abs(hilbert(app.DSB_TC));
    app.env_DSB_TC = app.env_DSB_TC - mean(app.env_DSB_TC);
    plot(app.UIAxes3, app.t, app.env_DSB_TC);
    xlabel(app.UIAxes3, 'Time (seconds)');
    ylabel(app.UIAxes3, 'DSB-TC');
    title(app.UIAxes3, 'Envelope of the DSB-TC');
    app.env_DSB_TC = resample(app.env_DSB_TC, app.Fs, 5 * app.fc);
    sound(app.env_DSB_TC, app.Fs);
    pause(8.5);
    app.Label_3.Text = '';
end
```

Output:



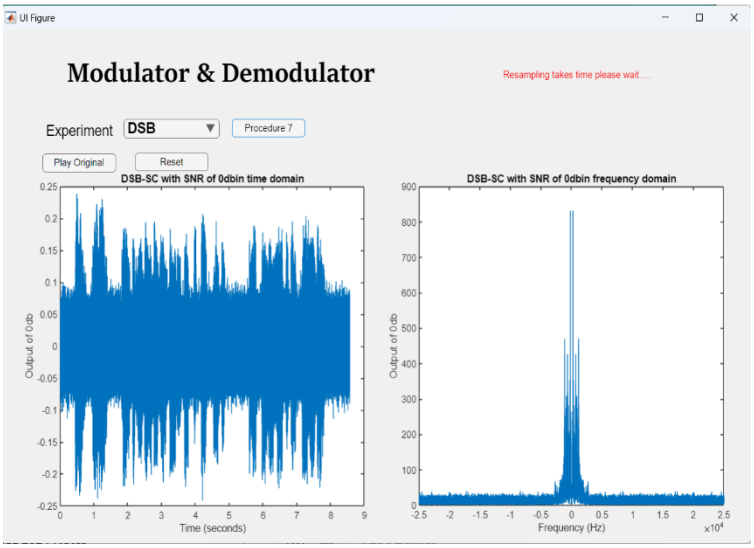
7- U Create vector for each SNR in db & Adding guassian distributed noise using awgn (from Communication Toolbox) & Vector for output & Vector for demodulated signal in time domain & Vector for demodulated signal in frequency domain & Adding noise to DSB-SC signal & Coherent detection & Plotting detected signal in time domain & Getting signal in the frequency domain & Plotting detected signal in frequency domain

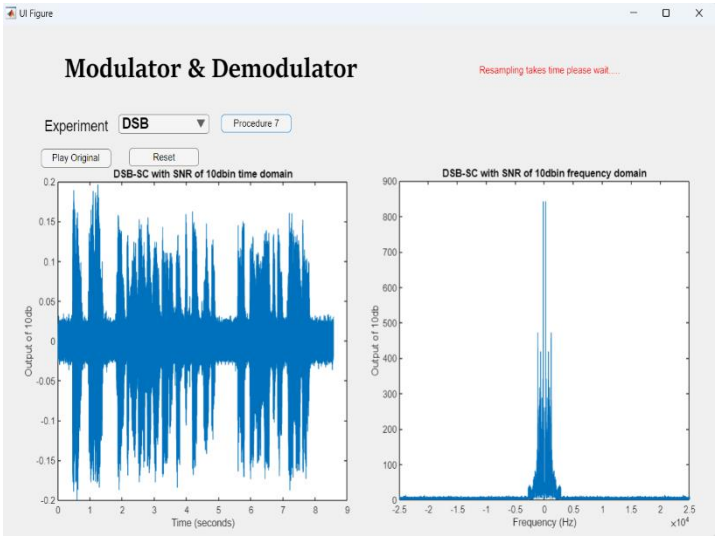
Code:

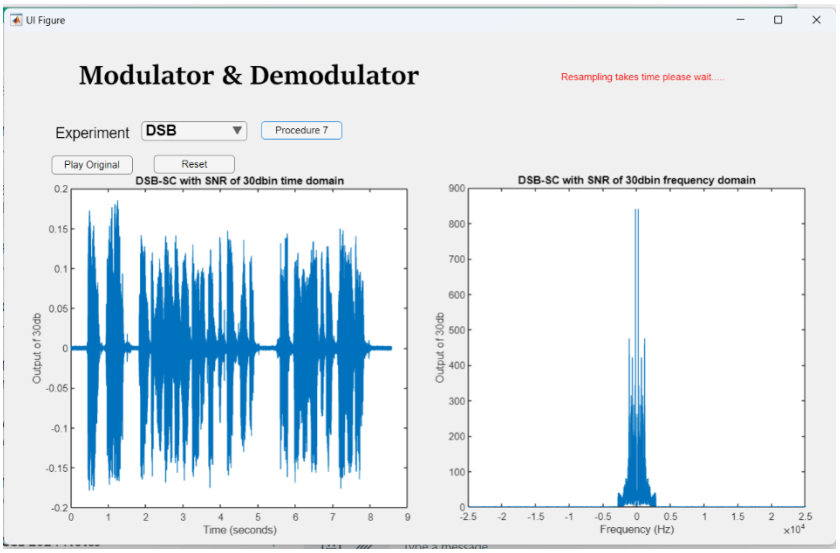
```
%%------------< EXP 1.7 >------------

    if app.counter == 7
        app.db_vec = [0, 10, 30];
        app.out_db(:, 1) = awgn(app.DSB_SC, app.db_vec(1), 'measured');
        app.out_db = zeros(length(app.DSB_SC), 3);
        app.demodulated = zeros(length(app.DSB_SC), 3);
        app.demod_freq_domain = zeros(5 * app.fc, 3);
             app.Label_3.Text = 'Resampling takes time please wait.....';
        for i = 1:3
            cla(app.UIAxes3, 'reset');
            cla(app.UIAxes2, 'reset');
            app.out_db(:, i) = awgn(app.DSB_SC, app.db_vec(i), 'measured');
            app.demodulated(:, i) = app.out_db(:, i) .* app.c_t;
            plot(app.UIAxes2, app.t, app.demodulated(:, i));
            xlabel(app.UIAxes2, 'Time (seconds)');
            ylabel(app.UIAxes2, sprintf('Output of %ddb', app.db_vec(i)));
            title(app.UIAxes2, sprintf('DSB-SC with SNR of %ddbin time domain',
app.db_vec(i)));

            app.demod_freq_domain(:, i) = abs(fftshift(fft(app.demodulated(:, i),
numel(app.fshift))));
            plot(app.UIAxes3, app.fshift, app.demod_freq_domain(:, i));
            xlim(app.UIAxes3, [-app.band_limit - 1e3 app.band_limit + 1e3]);
            xlabel(app.UIAxes3, 'Frequency (Hz)');
            ylabel(app.UIAxes3, sprintf('Output of %ddb', app.db_vec(i)));
            title(app.UIAxes3, sprintf('DSB-SC with SNR of %ddbin frequency domain',
app.db_vec(i)));
            pause(8.5);
        end
       app.Label_3.Text = 'Playing sound SNR of 0db please wait.....';
        app.s1 = resample(app.demodulated(:, 1), app.Fs, 5 * app.fc);
        app.s2 = resample(app.demodulated(:, 2), app.Fs, 5 * app.fc);
        app.s3 = resample(app.demodulated(:, 3), app.Fs, 5 * app.fc);
        sound(app.s1, app.Fs);
        pause(8.5);
        app.Label_3.Text = 'Playing sound SNR of 10db please wait.....';
        sound(app.s2, app.Fs);
        pause(8.5);
        app.Label_3.Text = 'Playing sound SNR of 30db please wait.....';
        sound(app.s3, app.Fs);
```

```
        pause(8.5);
        app.Label_3.Text = '';
        clear app.s1 app.s2 app.s3 app.out_db app.demodulated app.demod_freq_domain;
    end
```

Output:





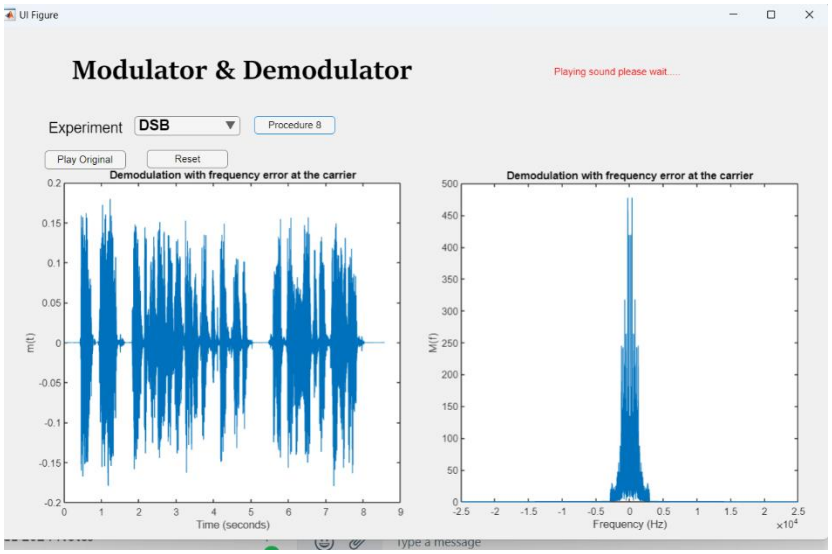


8- For DSB-SC signal the coherent detection was repeated with frequency error, F=100.1KHz instead of 100KHz. This would cause the output to be distorted the frequency error is 0.1KHz which is very small and would cause the overlapping of the 2 spectrum elements in positive and in negative by Adding frequency error & Applying the frequency error & Coherent detection with frequency error & Plotting the signal time domain

Code:

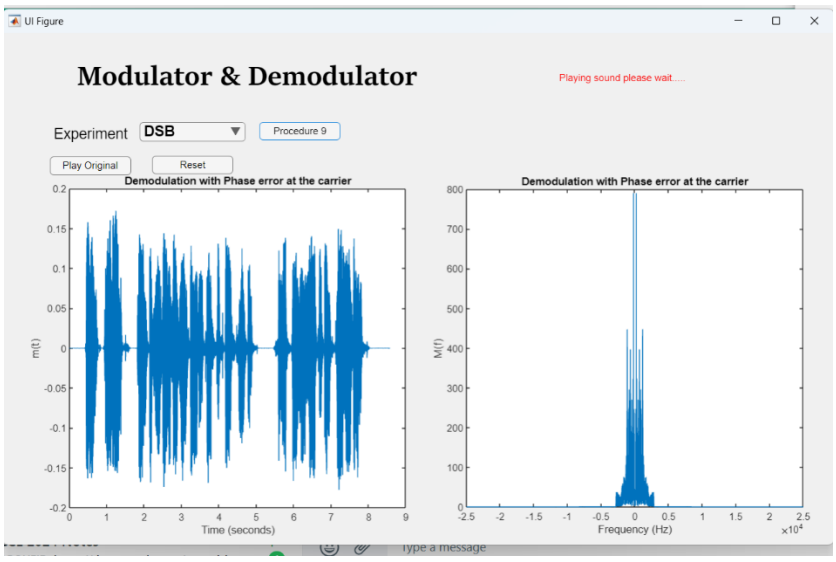
```
if app.counter == 8
    app.fc_error = app.fc + 0.1 * 1e3;
    app.c_t_freq_error = cos(2 * pi * app.fc_error * app.t)';
    app.demodulated_with_freq_error = app.DSB_SC .* app.c_t_freq_error;
    plot(app.UIAxes2, app.t, app.demodulated_with_freq_error);
    xlabel(app.UIAxes2, 'Time (seconds)');
    ylabel(app.UIAxes2, 'm(t)');
    title(app.UIAxes2, 'Demodulation with frequency error at the carrier');
    app.freq_error = abs(fftshift(fft(app.demodulated_with_freq_error,
numel(app.fshift))));
    plot(app.UIAxes3, app.fshift, app.freq_error);
    xlabel(app.UIAxes3, 'Frequency (Hz)');
    ylabel(app.UIAxes3, 'M(f)');
    title(app.UIAxes3, 'Demodulation with frequency error at the carrier');
    xlim(app.UIAxes3, [-app.band_limit - 1e3 app.band_limit + 1e3]);
    app.Label_3.Text = 'Playing sound please wait.....';
    app.s1 = resample(app.demodulated_with_freq_error, app.Fs, 5 * app.fc);
    sound(app.s1, app.Fs);
    pause(8.5);
    app.Label_3.Text = '';
end
```

Output:



9-coherent detection with phase error of 20o which caused a slight attenuation by first definning the phase error = pi / 9 and applying the phase error

```
if app.counter == 9
     phase_error = pi/9;
      carrier_with_error = cos(2 * pi * app.fc * app.t +  pi/9)';
      detected_with_error = app.DSB_SC .* carrier_with_error;
      plot(app.UIAxes2, app.t, detected_with_error);
      xlabel(app.UIAxes2, 'Time (seconds)');
      ylabel(app.UIAxes2, 'm(t)');
      title(app.UIAxes2, 'Demodulation with Phase error at the carrier');
      phase_error = abs(fftshift(fft(detected_with_error, numel(app.fshift))));
      plot(app.UIAxes3, app.fshift, phase_error);
      xlabel(app.UIAxes3, 'Frequency (Hz)');
      ylabel(app.UIAxes3, 'M(f)');
      title(app.UIAxes3, 'Demodulation with Phase error at the carrier');
      xlim(app.UIAxes3, [-app.band_limit - 1e3 app.band_limit + 1e3]);
      app.Label_3.Text = 'Playing sound please wait.....';
      app.s1 = resample(detected_with_error, app.Fs, 5 * app.fc);
      sound(app.s1, app.Fs);
      pause(8.5);
      app.Label_3.Text = '';
  end
```

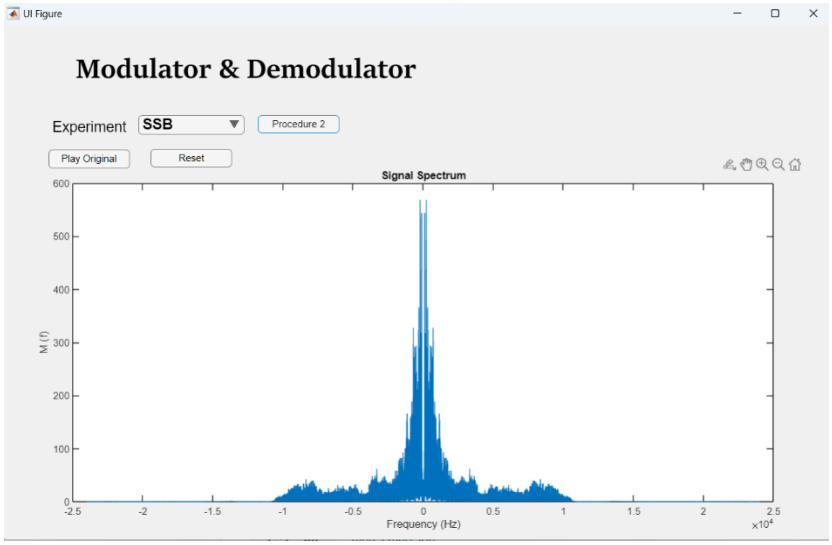


2- Single Side Band Modulation:

Procedure:

1- We are required to read audio signal, find and plot its spectrum using matlab commands (audioread, fft, fftshift, plot).

```
if app.counter == 1
  app.fvec = app.Fs
 / 2 * linspace(-1, 1, app.n);

 app.freq_y = fftshift(fft(app.y));
 app.fft_y = fftshift(fft(app.y, length(app.fvec)));
 plot(app.UIAxes, app.fvec, abs(app.fft_y));
 xlabel(app.UIAxes, 'Frequency (Hz)');
 ylabel(app.UIAxes, 'M (f)');
 title(app.UIAxes, 'Signal Spectrum');
end
```

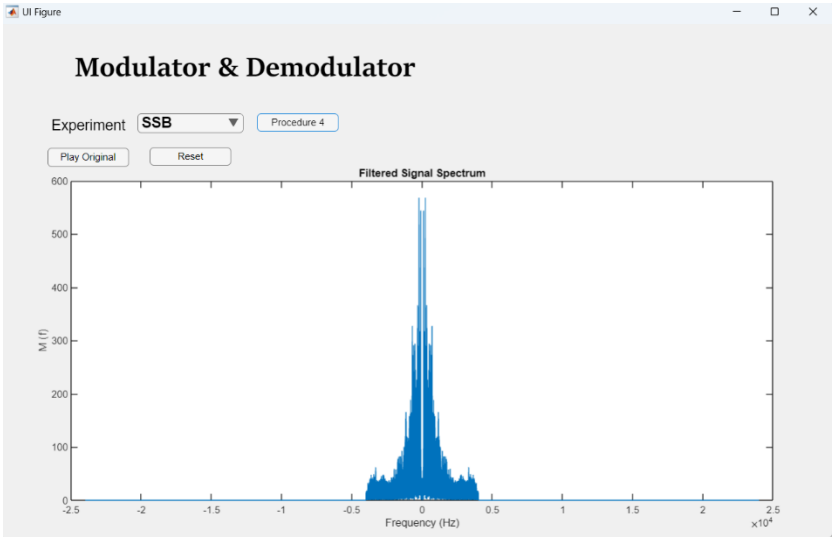


2- Just sounding the signal spectrum

```
if app.counter == 2

 app.Label_3.Text = 'Playing sound please wait.....';
 sound(app.y, app.Fs);
 pause(8.5);
 app.Label_3.Text = '';
end
```

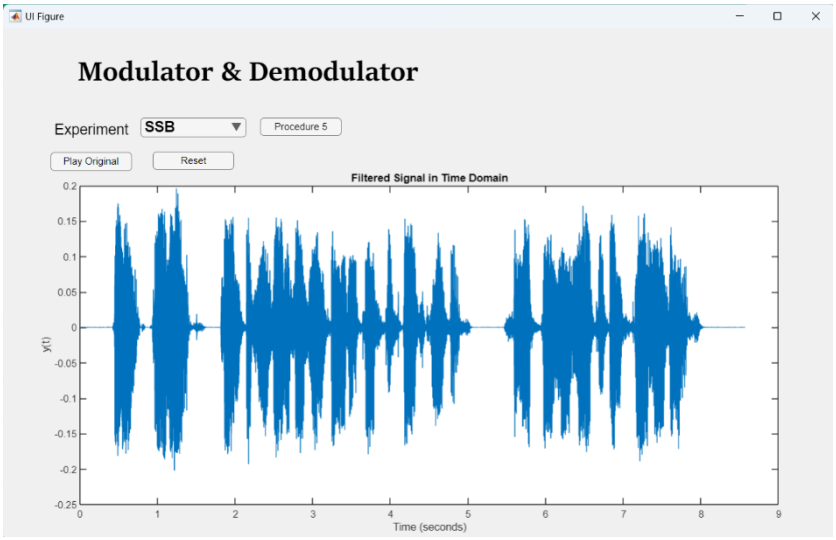
3- Using an ideal filter to remove all frequencies greater than 4 KHz & Plotting Filtered Signal Spectrum

```
if app.counter == 3

    app.filtered_signal = app.fft_y;
    app.filtered_signal(app.fvec >= 4000 | app.fvec <= -4000) = 0;
    plot(app.UIAxes, app.fvec, abs(app.filtered_signal));
    xlabel(app.UIAxes, 'Frequency (Hz)');
    ylabel(app.UIAxes, 'M (f)');
    title(app.UIAxes, 'Filtered Signal Spectrum');
    clear plot_filtered app.plotting_filter;
end
```



4- Generating Converting the signal back to time domain from frequency domain & plotting the filtered Signal in Time Domain

```
if app.counter == 4

    app.t = linspace(0, app.n / app.Fs, app.n);
    app.time_signal = real(ifft(ifftshift(app.filtered_signal)));
    plot(app.UIAxes, app.t, app.time_signal);
    xlabel(app.UIAxes, 'Time (seconds)');
    ylabel(app.UIAxes, 'm (t)');
    title(app.UIAxes, 'Filtered Signal in Time Domain');
    app.Label_3.Text = 'Playing sound please wait.....';
    sound(app.time_signal, app.Fs);
    pause(8.5);
    app.Label_3.Text = '';
end
```

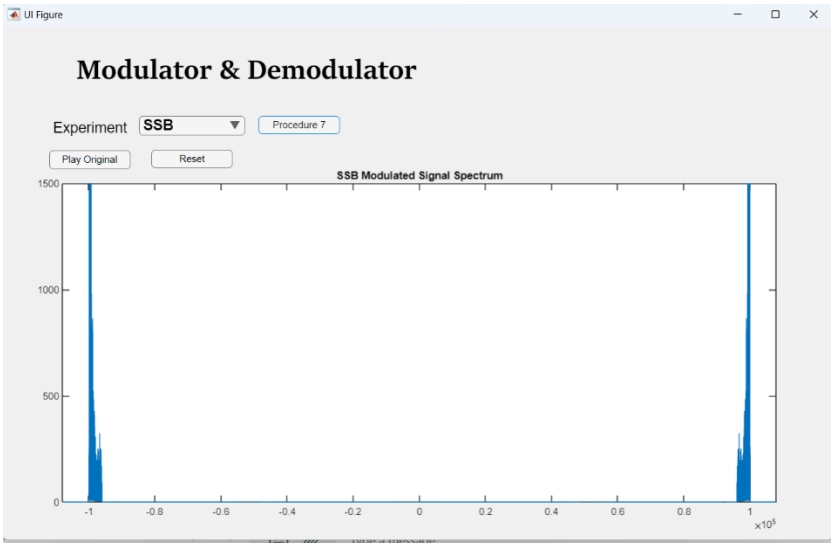


5- Changing fs to 5fc & Generating Vectors for defining time and frequency axis & Creating carrier signal & Getting DSB-SC signal & Plotting DSB-SC & Changing amplitude of carrier to double max of message

```
if app.counter == 5

    app.fc = 100000;
    app.m_t = resample(app.time_signal, 5 * app.fc, app.Fs);
    clear app.time_signal;
    app.n = length(app.m_t);
    app.fs_y = 5 * app.fc;
    app.t = linspace(0, app.n / (5 * app.fc), app.n);
    app.fshift = (app.fs_y / 2) * linspace(-1, 1, app.fs_y);
    app.c_t = cos(2 * pi * app.fc * app.t)';
    app.DSB_SC = app.m_t .* app.c_t;
    app.F_DSB_SC = abs(fftshift(fft(app.DSB_SC, length(app.fshift))));
    plot(app.UIAxes2, app.fshift, app.F_DSB_SC);
    xlabel(app.UIAxes2, 'Frequency (Hz)');
    ylabel(app.UIAxes2, 'DSB-SC');
    title(app.UIAxes2, 'DSB-SC Spectrum');
    xlim(app.UIAxes2, [-app.fc - 8000 app.fc + 8000]);
    ylim(app.UIAxes2, [0 1500]);
    app.Ac = 2 * max(app.m_t);
    app.DSB_TC = (app.Ac + app.m_t) .* app.c_t;
    app.F_DSB_TC = abs(fftshift(fft(app.DSB_TC, length(app.fshift))));
    plot(app.UIAxes3, app.fshift, app.F_DSB_TC);
    xlabel(app.UIAxes3, 'Frequency (Hz)');
    ylabel(app.UIAxes3, 'DSB-TC');
    title(app.UIAxes3, 'DSB-TC Spectrum');
    xlim(app.UIAxes3, [-app.fc - 8000 app.fc + 8000]);
    ylim(app.UIAxes3, [0 1500]);
end
```

6- Using Getting SSB-SC signal using ideal filter & Getting unit step function implementation & Plotting SSB modulated Signal Spectrum

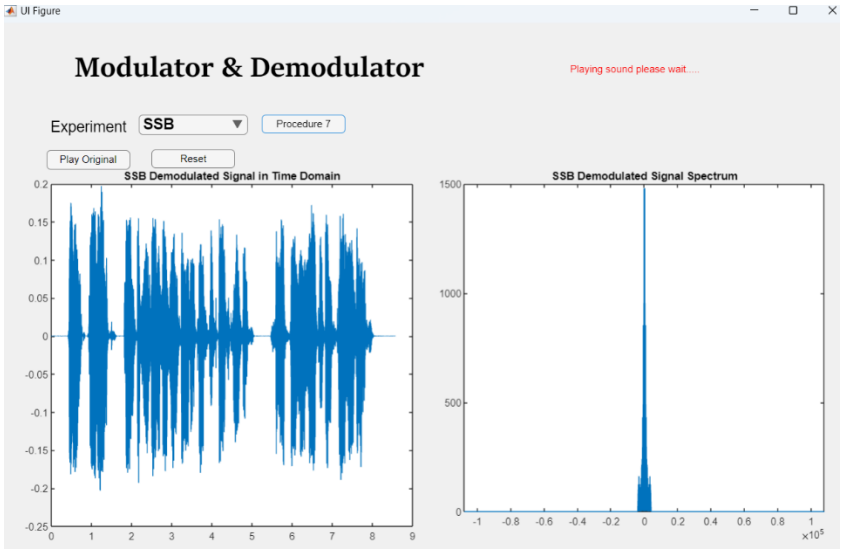
```
if app.counter == 6

    app.L = length(app.DSB_SC);
    app.freq = app.fs_y / 2 * linspace(-1, 1, app.L);
    app.SSB_f = fftshift(fft(app.DSB_SC, length(app.freq)));
    app.SSB_f(app.freq >= app.fc | app.freq <= -app.fc) = 0;
    app.SSB_t = ifft(ifftshift(app.SSB_f));

    plot(app.UIAxes, app.freq, abs(app.SSB_f));
    title(app.UIAxes, 'SSB Modulated Signal Spectrum');
    xlim(app.UIAxes, [-app.fc - 8000 app.fc + 8000]);
    ylim(app.UIAxes, [0 1500]);
end
```



7- Use Demodulating SSB-SC signal using coherent detection & Getting demodulated SSB-SC signal in time domain & Playing demodulated SSB-SC signal

```
if app.counter == 7

    app.Demodulated_SSB_t = app.SSB_t .* app.c_t;
    app.Demodulated_SSB_f = fftshift(fft(app.Demodulated_SSB_t));
    app.L = length(app.Demodulated_SSB_t);
    app.freq = app.fs_y / 2 * linspace(-1, 1, app.L);
    app.Demodulated_SSB_f(app.freq >= 4000 | app.freq <= -4000) = 0;
    app.Demodulated_SSB_t = 4 * real(ifft(ifftshift(app.Demodulated_SSB_f)));
    plot(app.UIAxes2, app.t, app.Demodulated_SSB_t);
    title(app.UIAxes2, 'SSB Demodulated Signal in Time Domain');
    plot(app.UIAxes3, app.freq, abs(app.Demodulated_SSB_f));
    title(app.UIAxes3, 'SSB Demodulated Signal Spectrum');
    xlim(app.UIAxes3, [-app.fc - 8000 app.fc + 8000]);
    ylim(app.UIAxes3, [0 1500]);
    app.Demodulated_SSB_t = resample(app.Demodulated_SSB_t, app.Fs, 5 * app.fc);
    app.Label_3.Text = 'Playing sound please wait.....';
```

```
        sound(app.Demodulated_SSB_t, app.Fs);
        pause(8.5);
        app.Label_3.Text = '';
    end
```

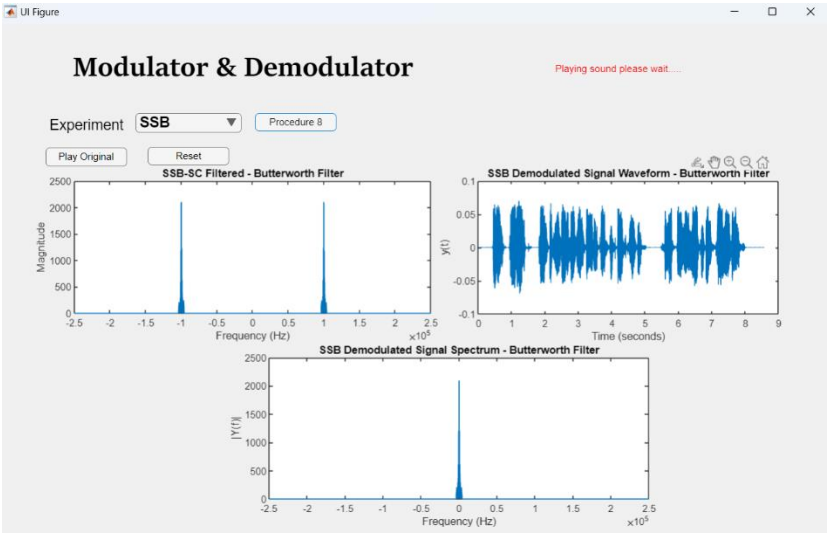


8- Using a butterworth filter of order 4 to filter DSB-SC signal to SSB-SC & Demodulating signal and obtaining spectrum & Getting demodulated SSB-SC signal in time and frequency domains & Plotting the Waveform and Spectrum of SSB Demodulated Butterworth-filtered & Playing Signal after butterworth filter

```
if app.counter == 8
        [app.b, app.a] = butter(4, app.fc / (app.fs_y / 2));
        app.SSB_practical_t = filter(app.b, app.a, app.DSB_SC);
        app.freq = app.fs_y / 2 * linspace(-1, 1, length(app.DSB_SC));

        plot(app.UIAxes4, app.freq, abs(fftshift(fft(app.SSB_practical_t,
length(app.freq)))));
        xlabel(app.UIAxes4, 'Frequency (Hz)');
        ylabel(app.UIAxes4, 'Magnitude');
        title(app.UIAxes4, 'SSB-SC Filtered - Butterworth Filter');
        app.Demod_SSB_butter_t = app.SSB_practical_t .* app.c_t;
        [app.b, app.a] = butter(4, 4000 / (app.fs_y / 2));
        app.Demod_SSB_butter_t = filter(app.b, app.a, app.Demod_SSB_butter_t);
        app.Demod_SSB_butter_f = fftshift(fft(app.Demod_SSB_butter_t));
        plot(app.UIAxes5, app.t, app.Demod_SSB_butter_t);
        title(app.UIAxes5, 'SSB Demodulated Signal Waveform - Butterworth Filter');
        xlabel(app.UIAxes5, 'Time (seconds)');
```

```
        ylabel(app.UIAxes5, 'm(t)');
        plot(app.UIAxes6, app.freq, abs(app.Demod_SSB_butter_f));
        title(app.UIAxes6, 'SSB Demodulated Signal Spectrum - Butterworth Filter');
        xlabel(app.UIAxes6, 'Frequency (Hz)');
        ylabel(app.UIAxes6, '|M(f)|');

        app.Demod_SSB_butter_t = resample(app.Demod_SSB_butter_t, app.Fs, 5 *
app.fc);
        app.Label_3.Text = 'Playing sound please wait.....';
        sound(app.Demod_SSB_butter_t, app.Fs);


        pause(8.5);
        app.Label_3.Text = '';
    end
```

Output:



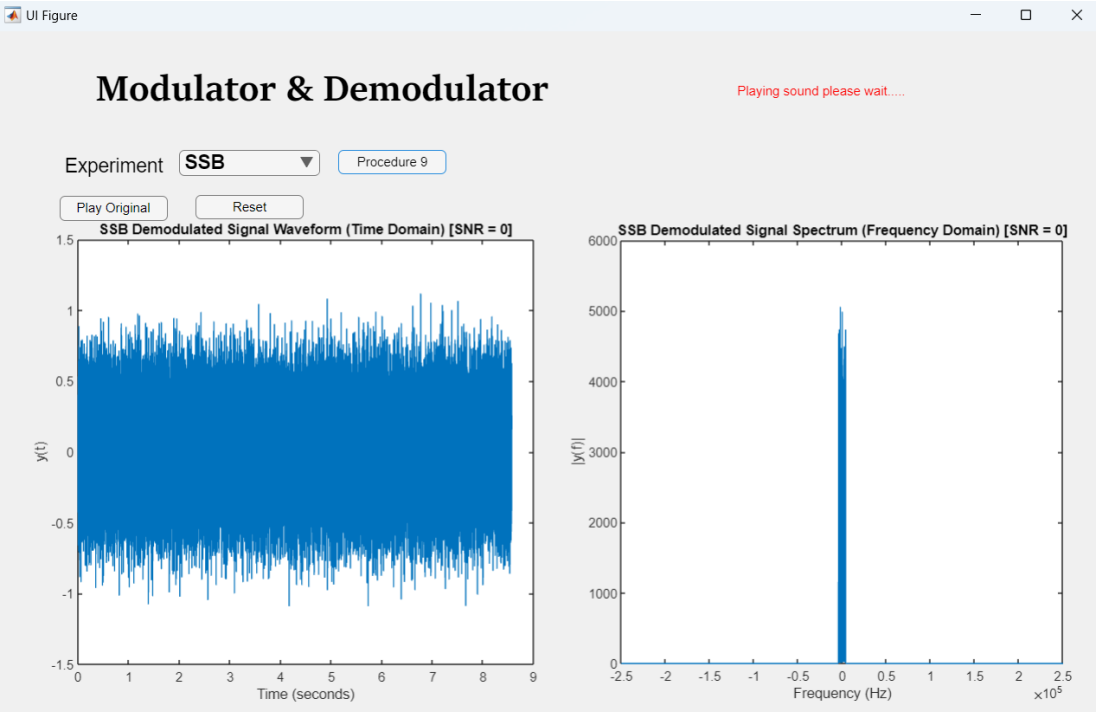
9- Adding white gaussian noise to signal & Demodulating noisy signal and obtaining spectrum & Getting noisy signal in Time Domain & Plotting the SSB Demodulated signal SNR = 0 in Time and frequency Domain and Spectrum & Playing Noisy Signal

Code:

```
    if app.counter == 9

        app.SSB_SNR0 = awgn(app.SSB_t, 0);
        app.demod_t_SNR0 = app.SSB_SNR0 .* app.c_t;
        app.demod_f_SNR0 = fftshift(fft(app.demod_t_SNR0));
        app.demod_f_SNR0(app.freq >= 4000 | app.freq <= -4000) = 0;
        app.demod_t_SNR0 = 4 * real(ifft(ifftshift(app.demod_f_SNR0)));
        plot(app.UIAxes2, app.t, app.demod_t_SNR0);
        title(app.UIAxes2, 'SSB Demodulated Signal Waveform (Time Domain) [SNR =
0]');
        xlabel(app.UIAxes2, 'Time (seconds)');
```

```
        ylabel(app.UIAxes2, 'm(t)');

        plot(app.UIAxes3, app.freq, abs(app.demod_f_SNR0));
        title(app.UIAxes3, 'SSB Demodulated Signal Spectrum (Frequency Domain) [SNR =
0]');
        xlabel(app.UIAxes3, 'Frequency (Hz)');
        ylabel(app.UIAxes3, '|M (f)|');


    % Playing Noisy Signal
        app.demod_t_SNR0 = resample(app.demod_t_SNR0, app.Fs, 5 * app.fc);
        app.Label_3.Text = 'Playing sound please wait.....';
        sound(app.demod_t_SNR0, app.Fs);
        pause(8.5);
        app.Label_3.Text = '';
    end
```

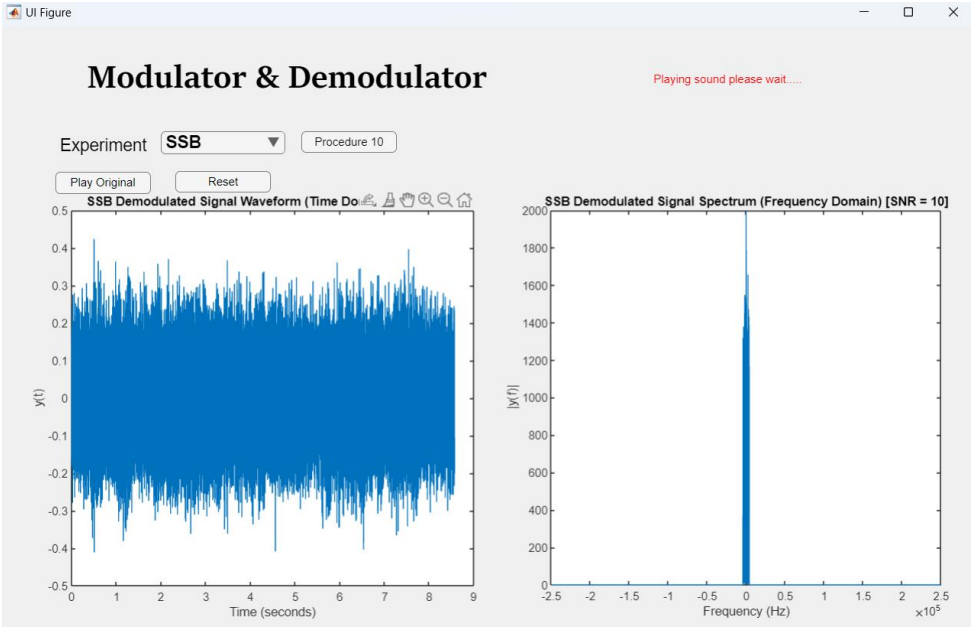
Output:



10- Adding white gaussian noise to signal & Demodulating noisy signal and obtaining spectrum & Getting noisy signal in Time Domain & Getting noisy signal in Time Domain & Plotting the SSB Demodulated

signal SNR = 10 in Time and frequency Domain and Spectrum

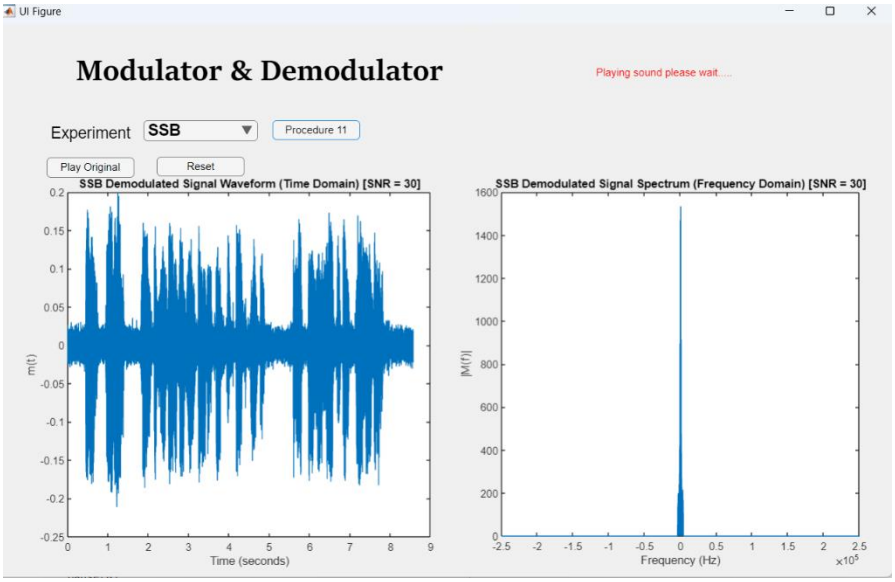
```
if app.counter == 10

        app.SSB_SNR10 = awgn(app.SSB_t, 10);
        app.demod_t_SNR10 = app.SSB_SNR10 .* app.c_t;
        app.demod_f_SNR10 = fftshift(fft(app.demod_t_SNR10));
        app.demod_f_SNR10(app.freq >= 4000 | app.freq <= -4000) = 0;
        app.demod_t_SNR10 = 4 * real(ifft(ifftshift(app.demod_f_SNR10)));
        plot(app.UIAxes2, app.t, app.demod_t_SNR10);
        title(app.UIAxes2, 'SSB Demodulated Signal Waveform (Time Domain) [SNR =
10]');
        xlabel(app.UIAxes2, 'Time (seconds)');
        ylabel(app.UIAxes2, 'm(t)');

        plot(app.UIAxes3, app.freq, abs(app.demod_f_SNR10));
        title(app.UIAxes3, 'SSB Demodulated Signal Spectrum (Frequency Domain) [SNR =
10]');
        xlabel(app.UIAxes3, 'Frequency (Hz)');
        ylabel(app.UIAxes3, '|M(f)|');
        app.demod_t_SNR10 = resample(app.demod_t_SNR10, app.Fs, 5 * app.fc);
        app.Label_3.Text = 'Playing sound please wait.....';
        sound(app.demod_t_SNR10, app.Fs);
        pause(8.5);
        app.Label_3.Text = '';
    end
```



11- Adding white gaussian noise to signal & Demodulating noisy signal and obtaining spectrum &

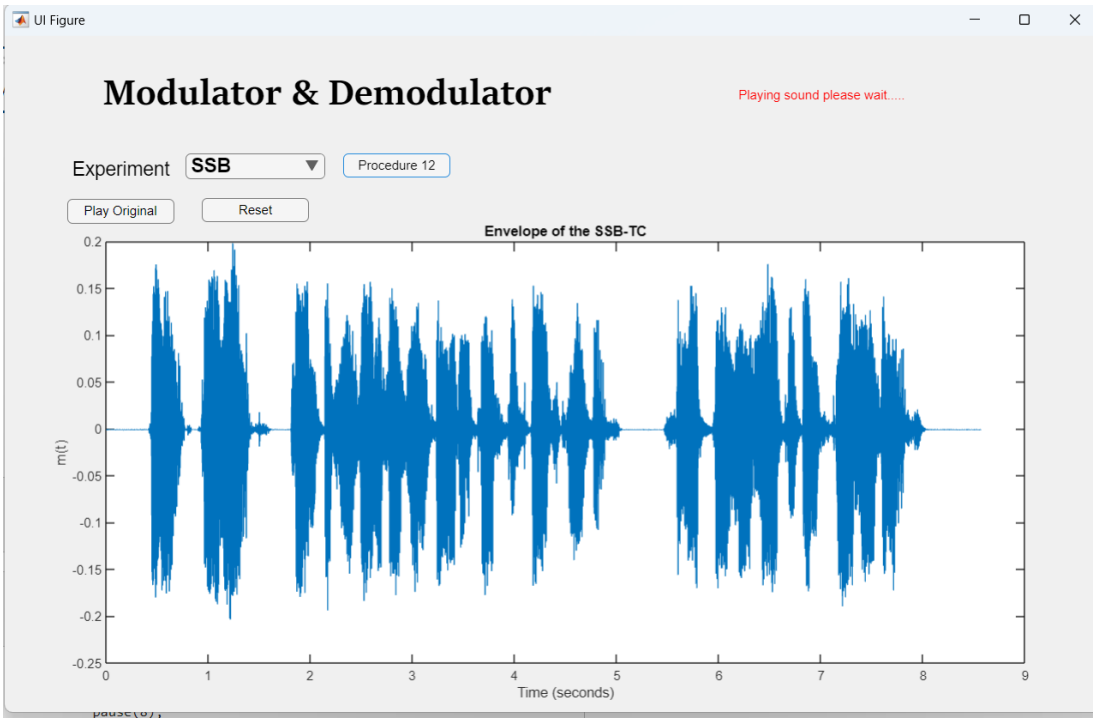
Plotting the SSB Demodulated signal SNR = 30 in Time and frequency Domain and Spectrum

```
  if app.counter == 11

        app.SSB_SNR30 = awgn(app.SSB_t, 30);
        app.demod_t_SNR30 = app.SSB_SNR30 .* app.c_t;
        app.demod_f_SNR30 = fftshift(fft(app.demod_t_SNR30));
app.demod_f_SNR30(app.freq >= 4000 | app.freq <= -4000) = 0;
        app.demod_t_SNR30 = 4 * real(ifft(ifftshift(app.demod_f_SNR30)));
        plot(app.UIAxes2, app.t, app.demod_t_SNR30);
        title(app.UIAxes2, 'SSB Demodulated Signal Waveform (Time Domain) [SNR =
30]');
        xlabel(app.UIAxes2, 'Time (seconds)');
        ylabel(app.UIAxes2, 'm(t)');
        plot(app.UIAxes3, app.freq, abs(app.demod_f_SNR30));
        title(app.UIAxes3, 'SSB Demodulated Signal Spectrum (Frequency Domain) [SNR =
30]');
        xlabel(app.UIAxes3, 'Frequency (Hz)');
        ylabel(app.UIAxes3, '|M(f)|');
        app.demod_t_SNR30 = resample(app.demod_t_SNR30, app.Fs, 5 * app.fc);
        app.Label_3.Text = 'Playing sound please wait.....';
        sound(app.demod_t_SNR30, app.Fs);
        pause(8.5);
        app.Label_3.Text = '';
    end
```



12- obtaining SSB-TC signal , Using Double the amplitude , Plotting envelope of SSB-SC

```
 if app.counter == 12

    app.SSB_t = real(app.SSB_t);
    app.SSB_t_TC = app.Ac .* app.c_t + app.SSB_t;
    app.envelopeSSB_TC = abs(hilbert(app.SSB_t_TC));
    app.envelopeSSB_TC = app.envelopeSSB_TC - mean(app.envelopeSSB_TC);
    app.envelopeSSB_TC = 2 * app.envelopeSSB_TC;
    plot(app.UIAxes, app.t, app.envelopeSSB_TC);
    title(app.UIAxes, 'Envelope of the SSB-TC');
    xlabel(app.UIAxes, 'Time (seconds)');
    ylabel(app.UIAxes, 'm(t)');
    app.envelopeSSB_TC = resample(app.envelopeSSB_TC, app.Fs, 5 * app.fc);
    app.Label_3.Text = 'Playing sound please wait.....';
    sound(app.envelopeSSB_TC, app.Fs);
    pause(8.5);
    app.Label_3.Text = '';
end
```



Conclusions:

Procedure 1 & 2:

Signal is filtered at 4 KHz to set a maximum frequency for the message signal to be used at the transmitter.

Procedure 3:

After the removal of the frequencies higher than 4 KHz, some distortion was indeed found, but not enough to make the message unintelligible when removing the higher frequencies. The message was still clear.

Procedure 4:

Sampling frequency was set to 5 times the carrier frequency to avoid the phenomenon of aliasing, as per Nyquist rule, which dictates that message's frequency must be 2 times the carrier frequency.

Procedure 5:

An ideal filter, implemented by using a version of a unit step, is used to filter out the upper side band frequencies.

Procedure 6:

After using coherent detection, it was found that the signal was still intelligible, even after removing half the bandwidth of the DSB signal

Procedure 7:

After using the butterworth filter on the DSB signal, it was found that the filter was not as effective as using the ideal filter (obviously), since frequencies above carrier frequency were still existent, though ultimately lower than they were in DSB.

Procedure 8:

Generating an SSB-TC signal of the original suppressed-carrier one proved to have no noticeable effect on the speech said in the signal, proving that transmitted carrier transmission similarly effective next to suppressed carrier, with the reduced cost

Procedure 9, 10 , 11:

Adding white gaussian noise with SNR = 0, the signal was transformed purely to said noise.

Adding white gaussian noise with SNR = 10, the speech said in the message was very faintly heard, still with a large amount of noise covering it.

Adding white gaussian noise with SNR = 30, the speech said in the message was even more clearly heard than with SNR 10, though not as clear as a zero-interference signal.

Frequency Modulation:

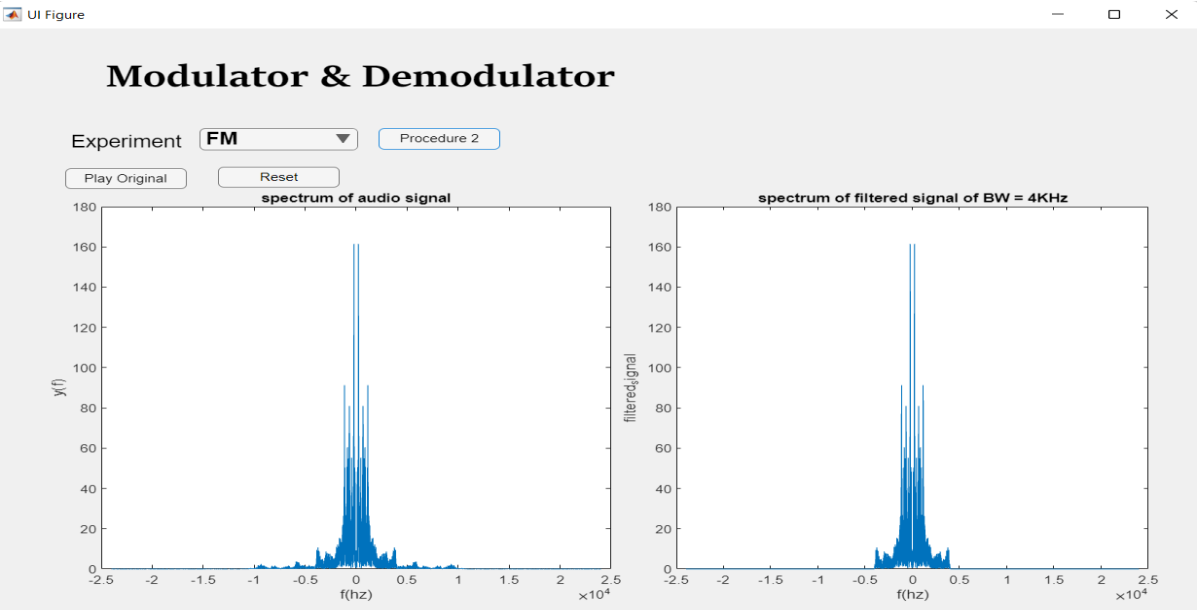
Procedure: 1 , 2 , 3 the same as DSB and SSB

1- We are required to read audio signal, find and plot its spectrum using matlab commands (audioread, fft, fftshift, plot).
2- We are required to remove all frequencies greater than 4KHz using Ideal filter.

Code:

```
%%------------< EXP 3.1 >------------
    if app.counter == 1
       % Getting Spectrum of audio signal
        app.UIAxes2.Visible  = 'on';
        app.fs_y = app.n / app.ty;
        app.fshift = (app.fs_y/2) * linspace(-1,1,app.fs_y);
        app.freq_y = fftshift(fft(app.y));
        app.fft_y = fftshift(fft(app.y, numel(app.fshift)));

        % Plotting the spectrum of the audio signal
        plot(app.UIAxes2,app.fshift, abs(app.fft_y));
        xlabel(app.UIAxes2,'Frequency (Hz)');
        ylabel(app.UIAxes2,'M (f)');
        title(app.UIAxes2,'Audio signal Spectrum');


    %%------------< EXP 3.2 >------------
        app.band_limit = floor(app.fn + app.ty);

        % Filter for the plotted signal LPF with 4KHZ cutoff frequency
        app.plotting_filter = cat(1, zeros([app.Fs/2-4e3, 1]), ones([4e3, 1]),
ones([4e3, 1]), zeros([app.Fs/2-4e3,1]));

        % Filter for the actual signal LPF with 4KHZ cutoff frequency with ideal
filter
        app.filter = cat(1, zeros([app.n/2-app.band_limit, 1]), ones([app.band_limit,
1]), ones([app.band_limit, 1]),zeros([app.n/2-app.band_limit, 1]));

        % Band Limited Signal used for Calculations
        app.filtered_signal = app.freq_y .* app.filter;

        % Band Limited Signal used for plotting
        app.plot_filtered = app.fft_y .* app.plotting_filter;
        app.UIAxes3.Visible  = 'on';

        % Plotting Filtered Signal Spectrum
        plot(app.UIAxes3,app.fshift, abs(app.plot_filtered));
        xlabel(app.UIAxes3,'Frequency (Hz)');
        ylabel(app.UIAxes3,'Filtered Signal');
        title(app.UIAxes3,'Filtered Signal Spectrum');
```

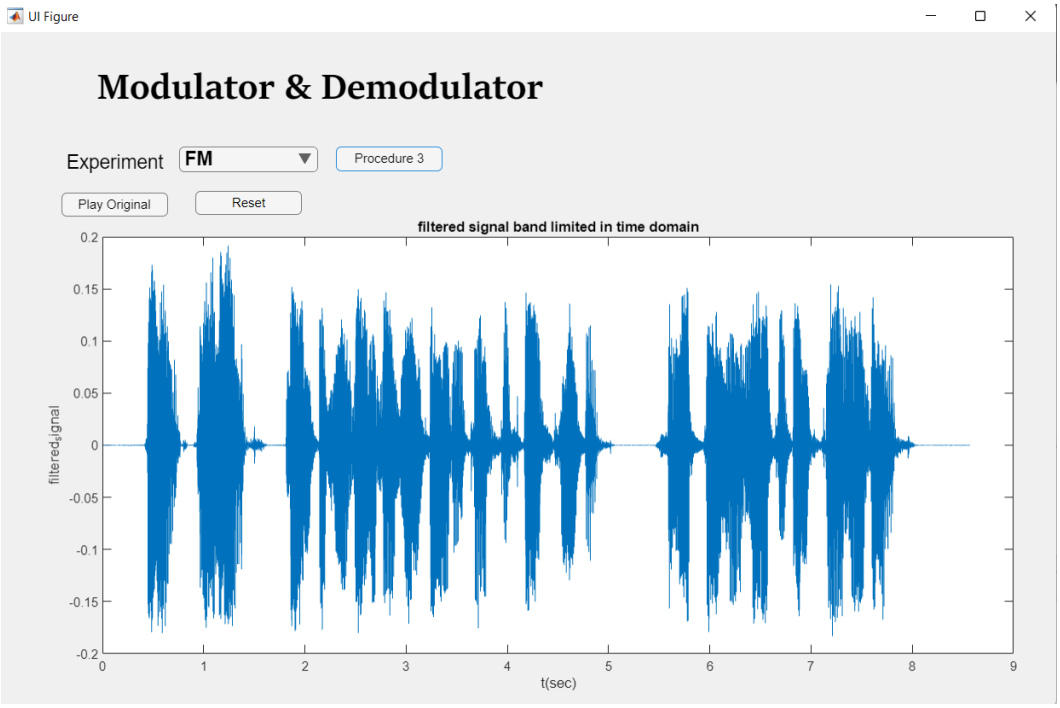
```
        clear plot_filtered plotting_filter;
    end
```



3- We are required to obtain and plot the filtered signal in time domain with limited bandwidth=4 using matlab commands (ifft, ifftshift). And sounding

After the removal of the frequencies higher than 4 KHz, some distortion was indeed found, but not enough to make the message unintelligible when removing the higher frequencies. The message was still clear.

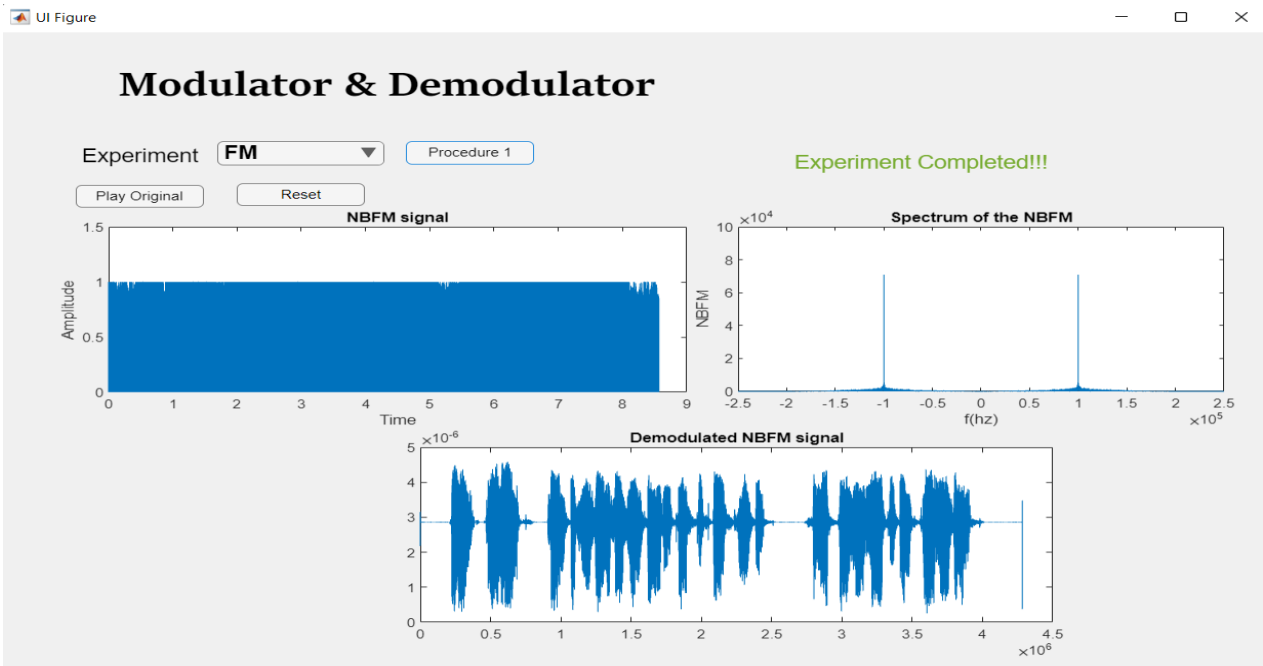
```
if app.counter == 2
      % Getting filtered signal in time domain from frequency domain
        app.time_signal = ifftshift(app.filtered_signal);
        clear filtered_signal;
        app.time_signal = real(ifft(app.time_signal));
        app.t = linspace(0,app.n/app.Fs,app.n);
        app.UIAxes.Visible  = 'on';

        % Plotting the filtered signal in time domain
        plot(app.UIAxes,app.t, app.time_signal); %
        xlabel(app.UIAxes,'Time (seconds)');
        ylabel(app.UIAxes,'Filtered Signal');
        title(app.UIAxes,'Filtered signal band limited in time domain');
    end
    if app.counter == 3
        app.Label_3.Text='Playing sound please wait.....';
        % Play filtered Signal
        sound(app.time_signal, app.Fs);
        pause(8.5);
        app.Label_3.Text='';
    end
```



3.2- Generating NBFM.

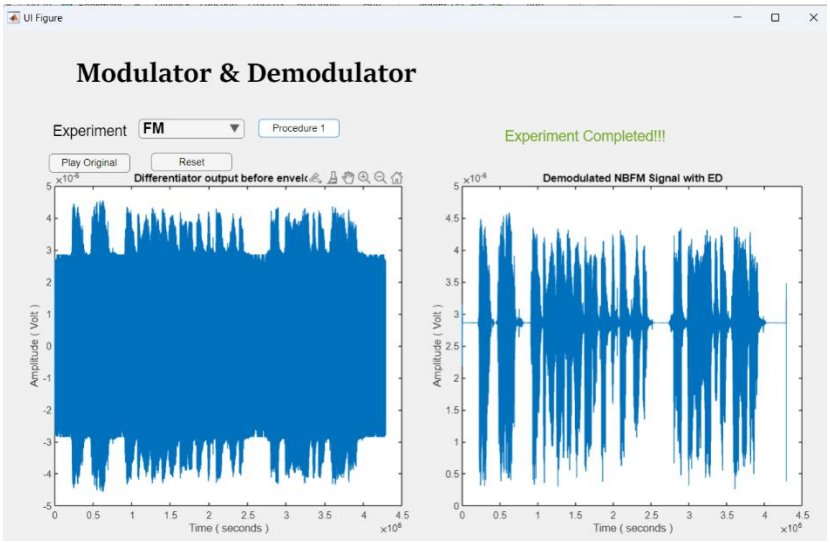
```
if app.counter == 4

      app.fc = 100000;

      % Resampling fs to 5fc
      app.m_t = resample(app.time_signal, 5*app.fc, app.Fs);
      clear time_signal;
      app.n = length(app.m_t);
      app.fs_y = 5*app.fc;
      app.fshift = (app.fs_y/2) * linspace(-1, 1, app.fs_y);
      app.t = linspace(0,app.n/app.fs_y,app.n);

      % Creating Carrier signal
      app.c_t = cos(2*pi*app.fc*app.t)';

      % Assume Kf=1 ( Small because narrow band )
      kf = 1;
      m_int=kf*2*pi*cumsum(app.m_t).';
      app.nbfm=cos(2*app.fc*pi*app.t + m_int);
      app.UIAxes2.Visible  = 'on';
      app.UIAxes3.Visible  = 'on';

      % Plotting the NBFM signal
      plot(app.UIAxes2,app.t,app.nbfm);
      title(app.UIAxes2,'NBFM Signal');
      xlabel(app.UIAxes2,'Time ( seconds )')
      ylabel(app.UIAxes2,'Amplitude ( Volt )')
      ylim(app.UIAxes2,[0 1.5]);
```

```
        F_nbfm = abs(fftshift(fft(app.nbfm, numel(app.fshift))));

        % Plotting the Spectrum of the NBFM
        plot(app.UIAxes3,app.fshift, F_nbfm);
        xlabel(app.UIAxes3,'Frequenct (Hz)');
        ylabel(app.UIAxes3,'NBFM');
        ylim(app.UIAxes3,[0 100000]);
        title(app.UIAxes3,'NBFM Spectrum');
    end
```



4- Plotting the differentiator output & Plotting the Demodulated NBFM signal with ED

```
if app.counter == 5

      am = [diff(app.nbfm) 0]./length(app.filtered_signal);
       envelope = abs(hilbert(am));
       app.UIAxes2.Visible  = 'on';
       app.UIAxes3.Visible  = 'on';
       % Plotting the differentiator output
       plot(app.UIAxes2,am);
       xlabel(app.UIAxes2,'Time ( seconds )')
       ylabel(app.UIAxes2,'Amplitude ( Volt )')
       title (app.UIAxes2,'Differentiator output before envelope');
       % Plotting the Demodulated NBFM signal with ED
       plot(app.UIAxes3,envelope);
       xlabel(app.UIAxes3,'Time ( seconds )')
       ylabel(app.UIAxes3,'Amplitude ( Volt )')
       title (app.UIAxes3,'Demodulated NBFM Signal with ED');
    end
```



What can we make out using NBFM ?

NBFM is used for voice communications in commercial and amateur radio settings. In broadcast services, where audio fidelity is important, wideband FM is generally used. In two-way radio, NBFM is used to conserve bandwidth for land mobile, marine mobile and other radio services.

3.3- what is the condition we needed to achieve NBFM ?

Using modulation index less than 1.

3.4- Demodulating the NBFM signal using a differentiator and an ED and Assuming no noise.

Conclusions:

3.1

Procedure 1 & 2:

Signal is filtered at 4 KHz to set a maximum frequency for the message signal to be used at the transmitter.

Procedure 3:

After the removal of the frequencies higher than 4 KHz, some distortion was indeed found, but not enough to make the message unintelligible when removing the higher frequencies. The message was still clear.

3.2

Sampling frequency was set to 5 times the carrier frequency to avoid the phenomenon of aliasing, as per Nyquist rule, which dictates that message's frequency must be 2 times the carrier frequency.

Narrowband FM is used for voice communications in commercial and amateur radio settings. In broadcast services, where audio fidelity is important, wideband FM is generally used. In two-way radio, narrowband FM (NBFM) is **used to conserve bandwidth for land mobile, marine mobile and other radio services**.

3.3

When value of modulation index is less than 1, then FM band is called as **Narrowband FM** (NBFM)

3.4

A frequency discriminator theoretically extracts the message from the received FM signal.

The modulated signal s(t) is:

$$\frac{ds(t)}{dt} = -A_c[2\pi f_c + K_f m(t)]sin\left(2\pi f_c t + 2\pi K_f \int_{-\infty}^{t} m(\lambda)d\lambda\right)$$

The differentiated signal is both amplitude and frequency modulated, the envelope $A_c[2\pi f_c + K_f m(t)]$ is linearly related to message signal (amplitude component) and $sin\left(2\pi f_c t + 2\pi K_f \int_{-\infty}^{t} m(\lambda)d\lambda\right)$ is high frequency component. Therefore, m(t) can be recovered by an envelope detection of $ds(t)/dt$.