	Communication and Electronics Department Analog Communication - EEC 381 Fall 2023 - 2024

REPORT

Name	ID
Abdel-Rahman Hesham Mohamed Desoky	20012332
Mostafa Mohamed Abdel-Azeem Hassanen	20011950
Ahmed Abdel-Hakeem Abdel-Salam Ali	20010124
Ahmed Kamel Mohamed Abdel-Gelel	20010156
Omar Ayman AbdelAal Shahin	19016059

Section:	8

Department:	**Communication and Electronics**

Experiment One: Double Sided Band Modulation: -

Signal in frequency domain:

```
function [signal, f_Sampling, S_freq, freq] = start(filename)
    % start analyzes the input audio file and computes its spectrum
    % Inputs:
    %   filename: Name of the audio file to be analyzed

    % Read the audio file and extract the signal and sampling frequency
    [signal, f_Sampling] = audioread(filename);

    % Compute the length of the signal
    len = length(signal);

    % Compute the frequency spectrum of the signal
    S_freq = fftshift(fft(signal));
    freq = (f_Sampling / 2) * linspace(-1, 1, len);

    % Plot the spectrum of the signal
    figure;
    plot(freq, abs(S_freq));
    title('Signal Spectrum'); % Set the title for the plot
end
```

Figure 1 : start function responsible for analyzes the input audio file and computes its spectrum.

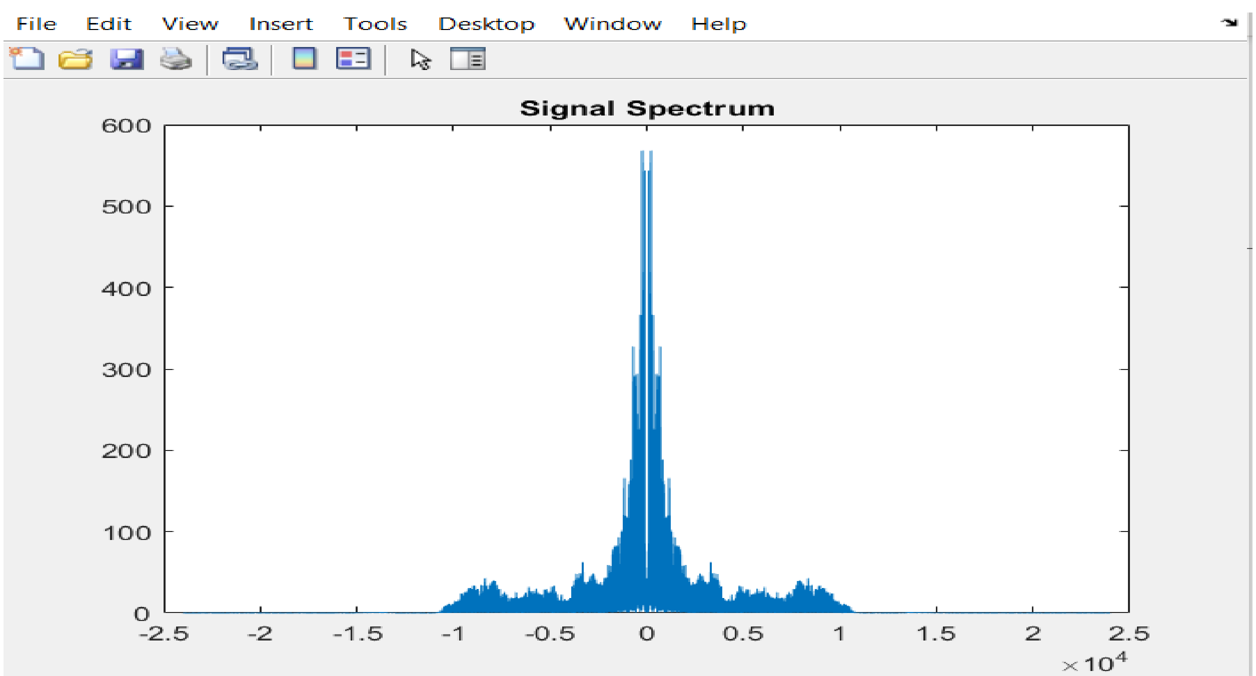


Figure 2 original spectrum of the signal

Filtered Signal:

```
function filteredSignal = filtering(cutoffFreq, S_freq, freq, f_Sampling)
    % filtering filters the input frequency-domain signal S_freq based on the cutoff frequency
    % Inputs:
    %   cutoffFreq: Frequency cutoff for filtering in Hz
    %   S_freq: Frequency domain signal
    %   freq: Frequency vector corresponding to the signal
    %   f_Sampling: Sampling frequency in Hz

    % Apply frequency domain filtering by zeroing out frequencies outside cutoff
    S_freq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Retrieve the filtered signal in time domain using inverse FFT
    filteredSignal = ifft(ifftshift(S_freq));

    % Compute the length of the filtered signal
    len = length(filteredSignal);

    % Compute the frequency spectrum of the filtered signal
    S_freq = fftshift(fft(filteredSignal));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the filtered signal
    figure;
    plot(freq, abs(S_freq));
    title('Filtered Signal Spectrum'); % Set the title for the plot
end
```

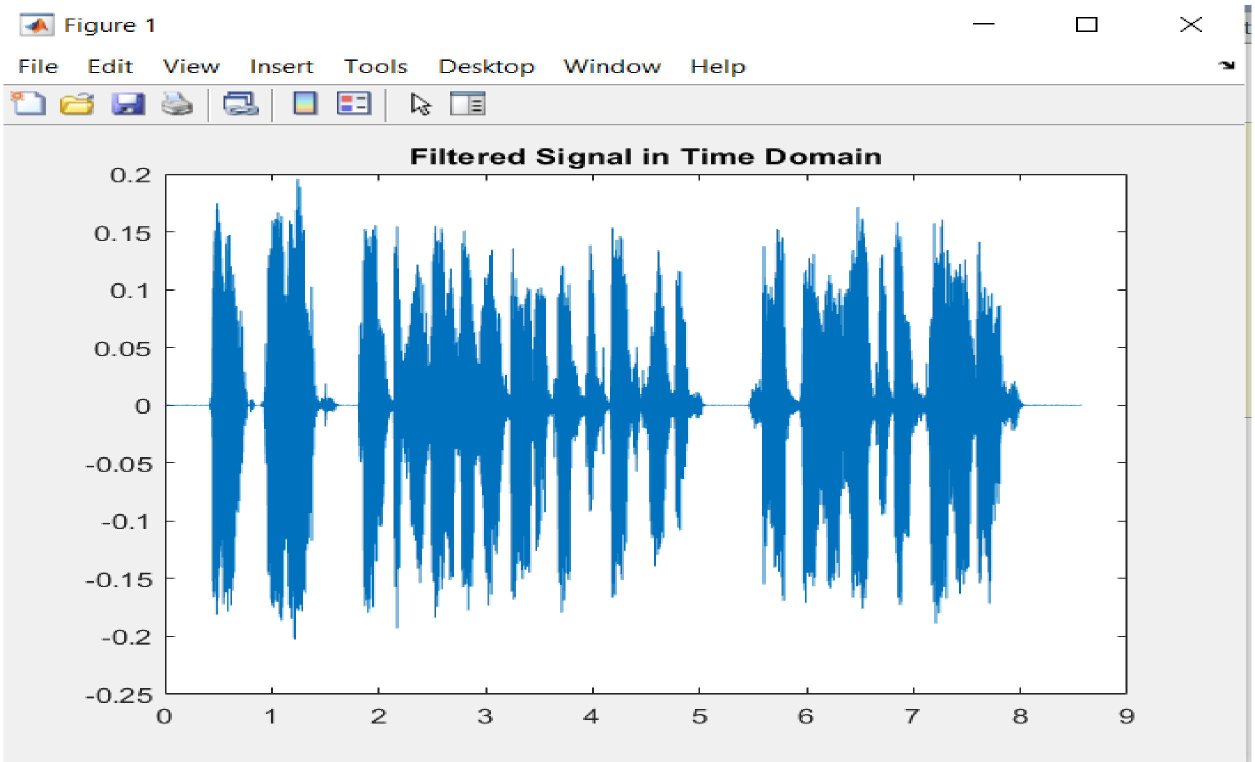


Figure 1
File Edit View Insert Tools Desktop Window Help
Filtered Signal in Time Domain
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0 1 2 3 4 5 6 7 8 9

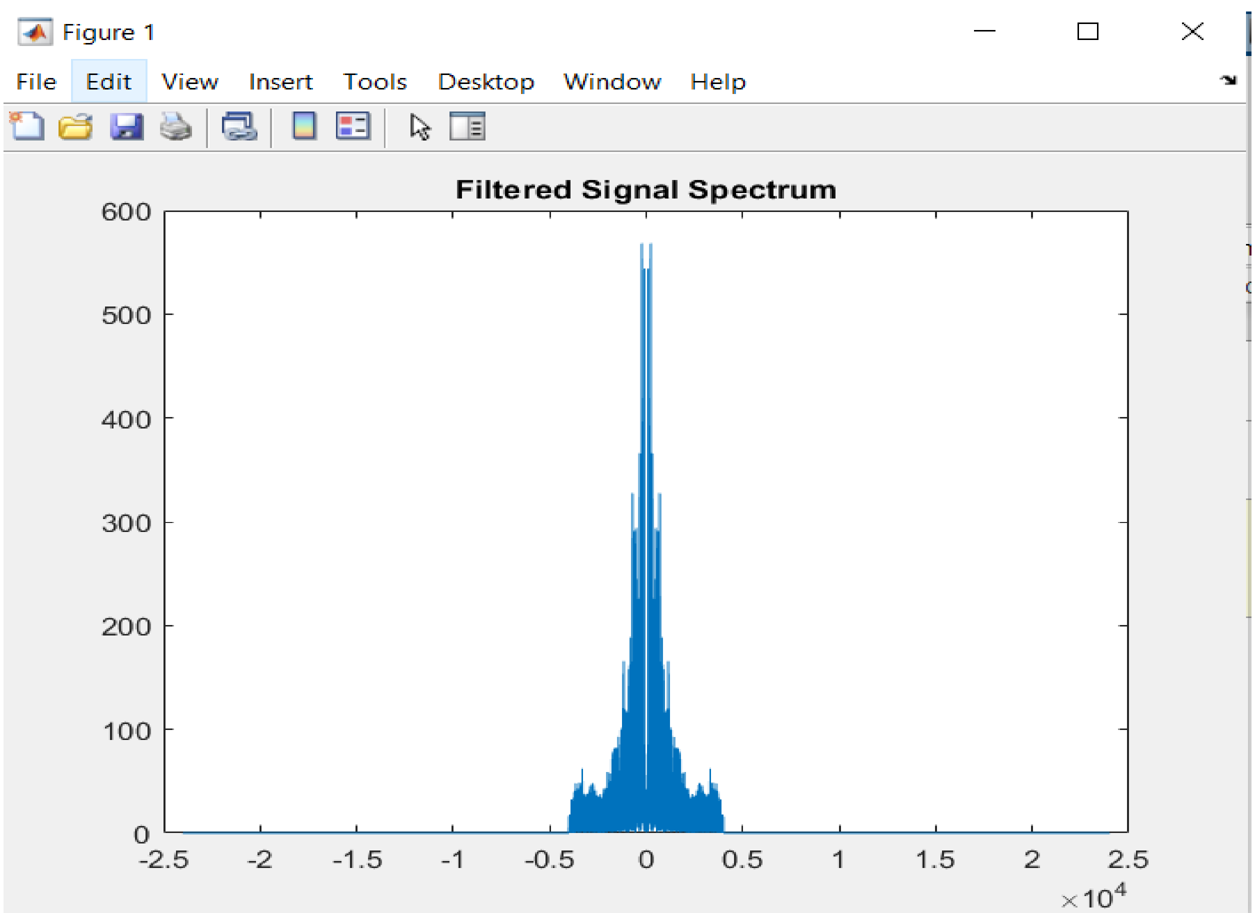
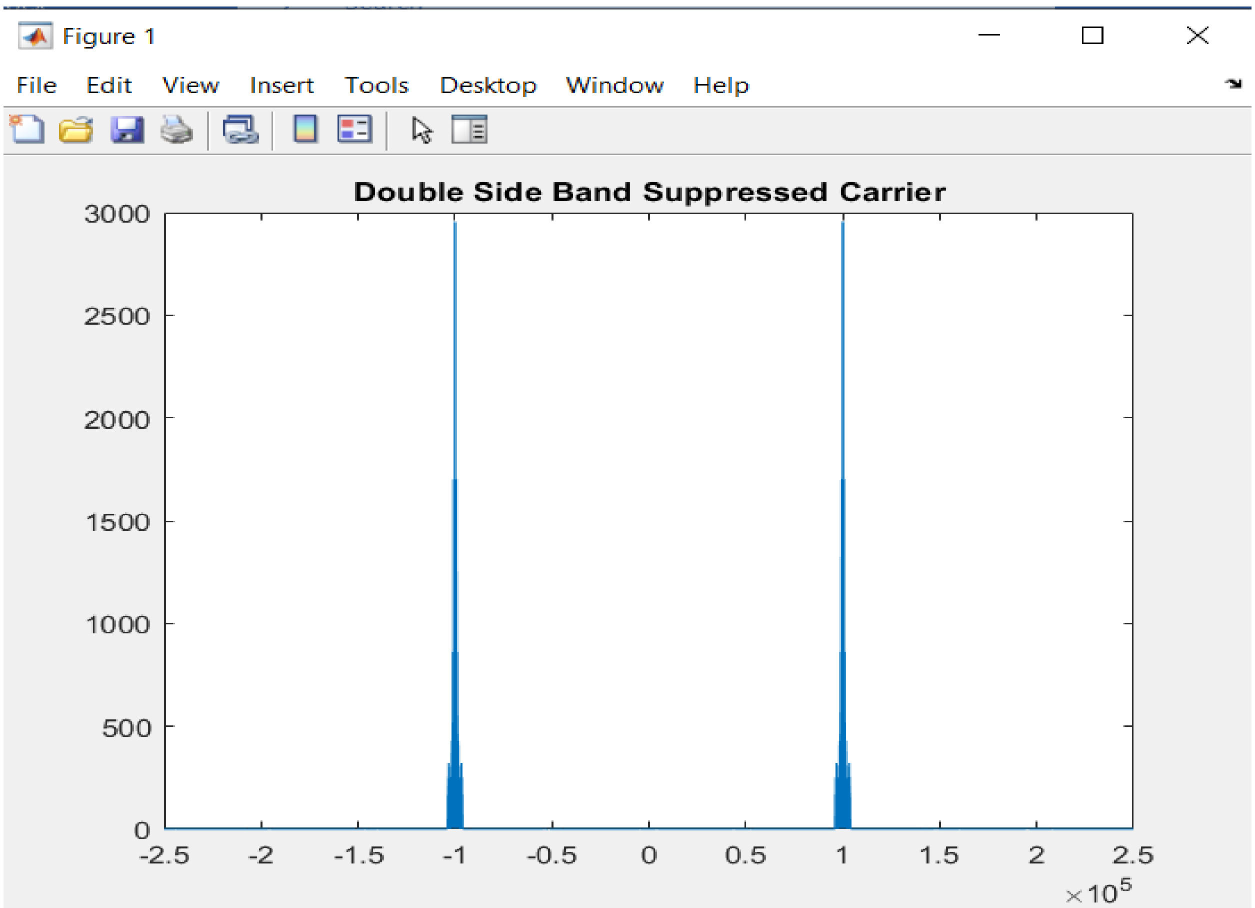


Figure 1
File Edit View Insert Tools Desktop Window Help
Filtered Signal Spectrum
600
500
400
300
200
100
0
-2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5
×10^4

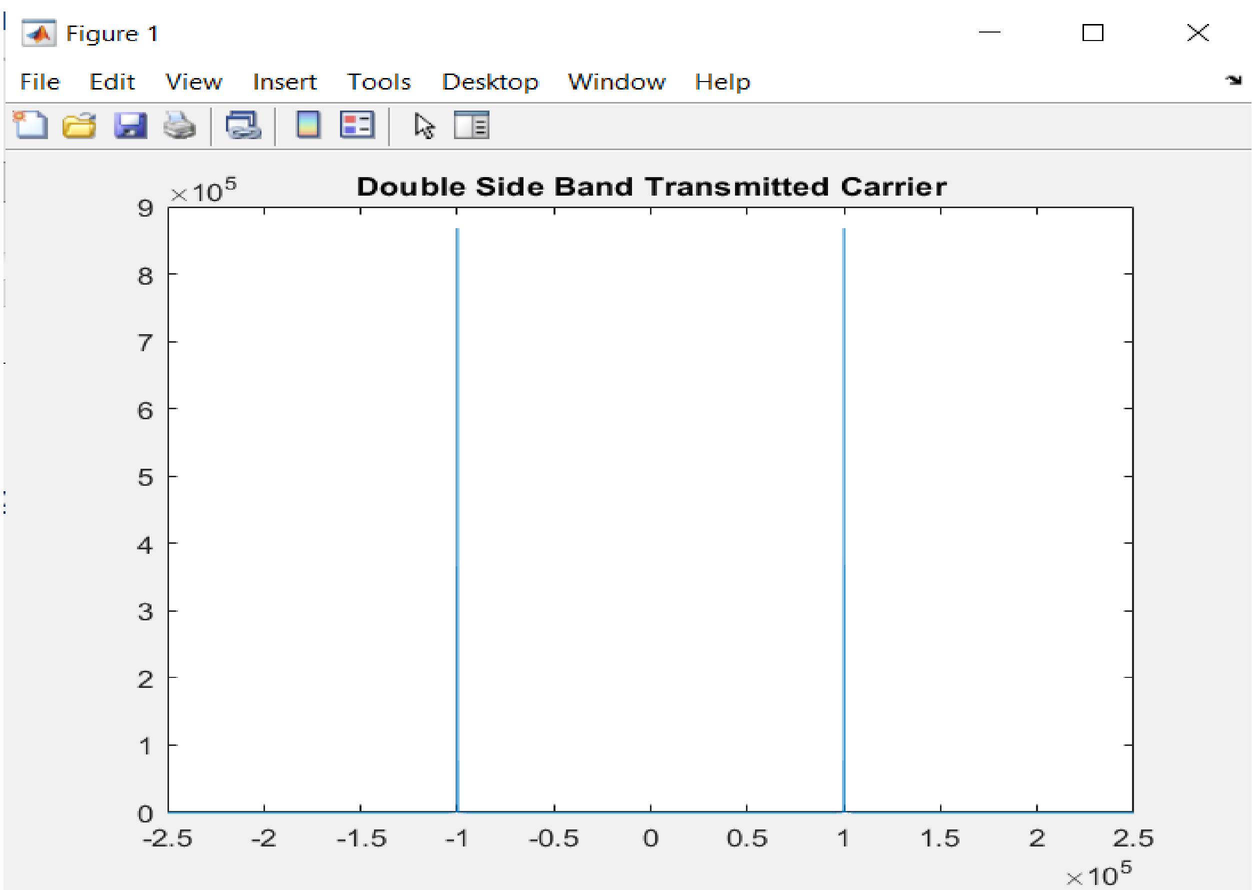
Double Sideband Suppressed Carrier:

```
function DSB_SC = suppressedCarrier(carrier, signal, f_Sampling)
    % suppressedCarrier generates a Double Side Band Suppressed Carrier (DSB-SC) signal
    % Inputs:
    %   carrier: Carrier signal
    %   signal: Input signal
    %   f_Sampling: Sampling frequency in Hz
    modIndex = 0.5;
    amplitude = max(signal);
    % Generate the DSB-SC signal by modulating the input signal with the carrier
    DSB_SC = modIndex * signal/amplitude .* carrier;

    % Compute the length of the DSB-SC signal
    len = length(DSB_SC);

    % Compute the frequency spectrum of the DSB-SC signal
    DSB_SC_Freq = fftshift(fft(DSB_SC));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the DSB-SC signal
    figure;
    plot(freq, abs(DSB_SC_Freq));
    title('Double Side Band Suppressed Carrier'); % Set the title for the plot
end
```



Double Sideband Transmitted Carrier:

```
function DSB_TC = transmittedCarrier(carrier, signal, modIndex, f_Sampling)
    % transmittedCarrier generates a Double Side Band Transmitted Carrier (DSB-TC) signal
    % Inputs:
    %   carrier: Carrier signal
    %   signal: Input signal
    %   modIndex: Modulation index
    %   f_Sampling: Sampling frequency in Hz

    % Determine the amplitude of the input signal
    amplitude = max(signal);

    % Generate the DSB-TC signal by modulating the carrier with the input signal
    DSB_TC = (1 + modIndex * signal / amplitude) .* carrier;

    % Compute the length of the DSB-TC signal
    len = length(DSB_TC);

    % Compute the frequency spectrum of the DSB-TC signal
    DSB_TC_Freq = fftshift(fft(DSB_TC));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the DSB-TC signal
    figure;
    plot(freq, abs(DSB_TC_Freq));
    title('Double Side Band Transmitted Carrier'); % Set the title for the plot
end
```

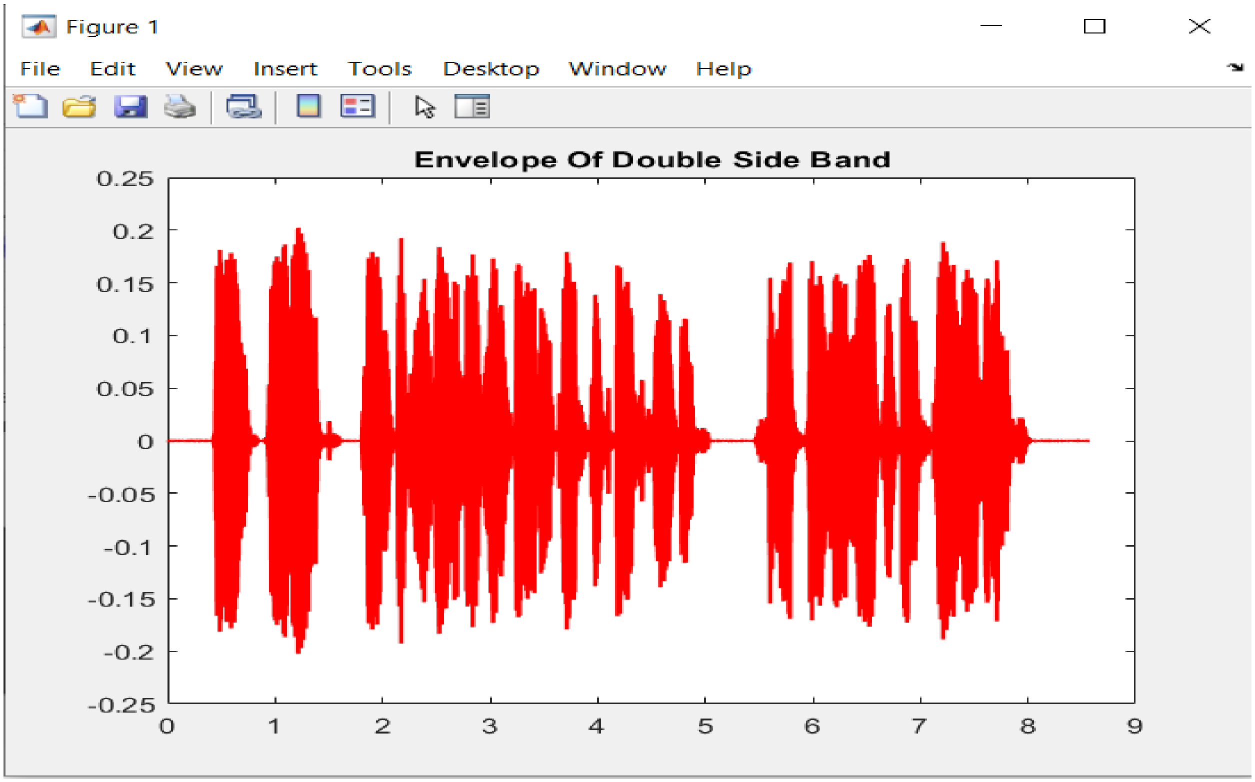


➢ Envelop Detection: -

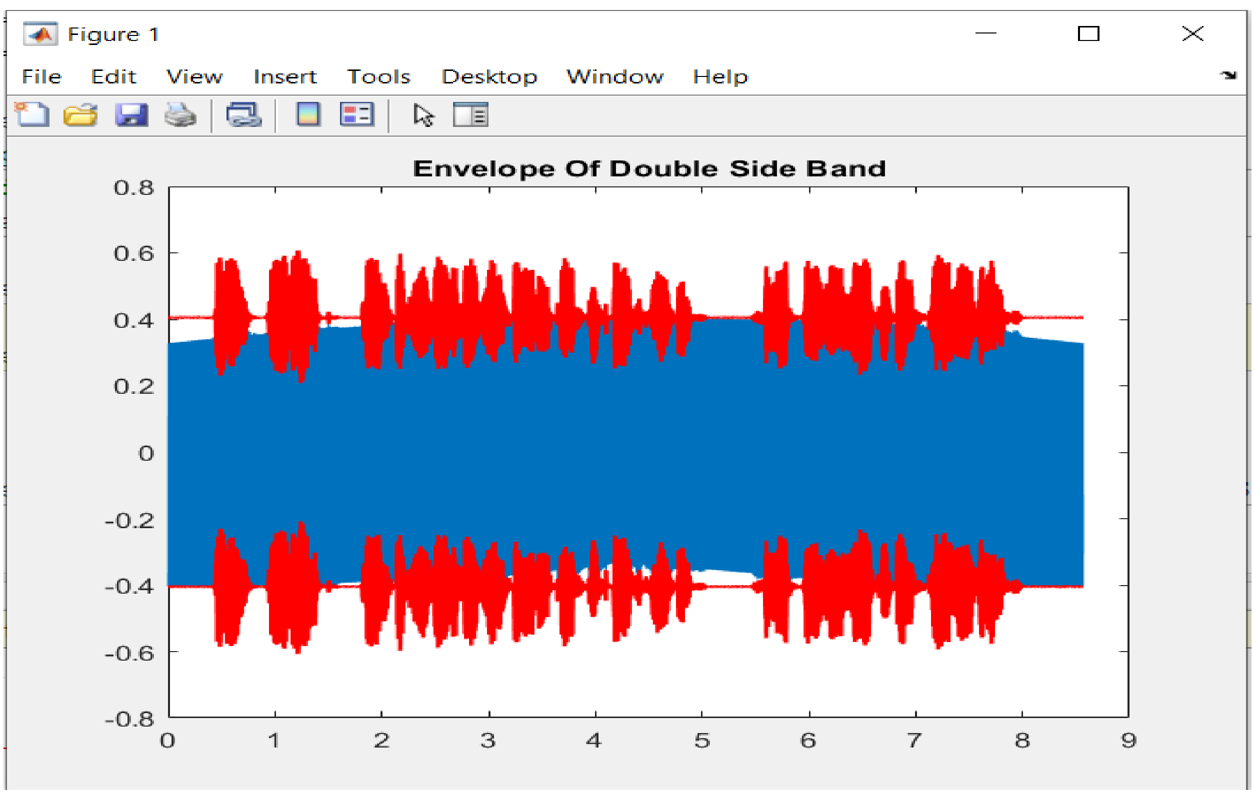
```matlab
function envelopeDetection(signal, time, f_Sampling)
    % envelopeDetection computes and plots the envelope of a signal
    % Inputs:
    %   signal: Input signal for envelope detection
    %   time: Time vector corresponding to the signal
    %   f_Sampling: Sampling frequency in Hz

    % Compute the envelope of the signal using Hilbert transform
    signalEnvelope = abs(hilbert(signal));

    % Plot the original signal and its envelope
    figure;
    plot(time, signal); % Plot the original signal
    hold on;
    plot(time, -signalEnvelope, '-r', time, signalEnvelope, '-r', 'Linewidth', 1.5); % Plot the envelope
    hold off;
    title('Envelope Of Double Side Band'); % Set the title for the plot

    % Resample the signal envelope and play the audio
    signalEnvelope = resample(abs(signalEnvelope), f_Sampling / 5, f_Sampling);
    sound(abs(signalEnvelope), f_Sampling / 5); % Play the resampled signal envelope
end
```

-In this point our observations are while playing the sound in case of DSB-SC the sound wasn't clear and good enough to hear the whole message as there is a lot of attenuation, distortion and phase reversal, in contrast to DSB-TC case the sound was excellent and we can hear the message clearly without distortion or attenuation or phase reversal because of the modulation index which is less than 1, so in summary the envelop detector receiver is much better with DSB-TC but in cases where modulation index is less than (Under modulation) or equal (Critical modulation) 1 other than these cases the envelop detector will be bad choice to receive a DSB signal and coherent detector will be better.

A) Envelop Detection of Double Sideband Suppressed Carrier:



B) Envelop Detection of Double Sideband Transmitted Carrier:



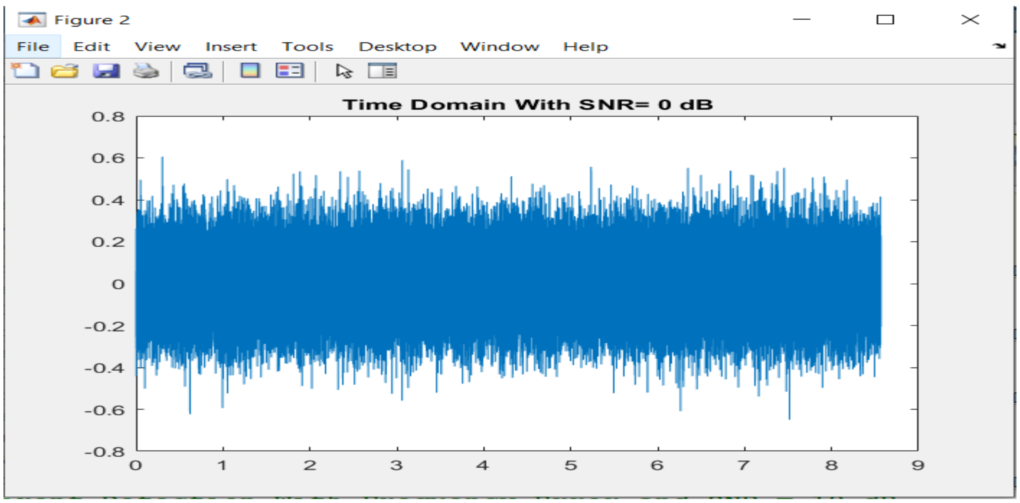
➢ Coherent Detection:

```
function coherentDetection(dB, signal, carrierFreq, time, cutoffFreq, f_Sampling, phase)
    % coherentDetection performs coherent detection of a modulated signal
    % Inputs:
    %   dB: Signal-to-noise ratio in decibels
    %   signal: Input signal to be detected
    %   carrierFreq: Carrier frequency of the signal
    %   time: Time vector corresponding to the signal
    %   cutoffFreq: Frequency cutoff for filtering in Hz
    %   f_Sampling: Sampling frequency in Hz
    %   phase: Phase of the carrier signal in radians

    % Add white Gaussian noise to the input signal based on given SNR
    snr_dB = awgn(signal, dB);
    modIndex = 0.5;
    % Demodulate the signal using coherent detection
    demodSignal = snr_dB/modIndex .* cos(2 * pi * carrierFreq * time + phase);

    % Compute the frequency spectrum of the demodulated signal
    demodSignalFreq = fftshift(fft(demodSignal));
    len = length(demodSignal);
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Apply frequency domain filtering by zeroing out frequencies outside cutoff
    demodSignalFreq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Plot the frequency spectrum
    figure;
    plot(freq, abs(demodSignalFreq));

    % Set title based on the presence of frequency or phase error
    if (carrierFreq ~= 100000)
        title(sprintf("Spectrum With SNR= %d dB and Frequency Error", dB));
    elseif (phase ~= 0)
        title(sprintf("Spectrum With SNR= %d dB and Phase Error", dB));
    else
        title(sprintf("Spectrum With SNR= %d dB", dB));
    end

    % Retrieve the demodulated signal from frequency domain
    demodSignal = ifft(ifftshift(demodSignalFreq));

    % Plot the demodulated signal in time domain
    figure;
    plot(time, demodSignal);

    % Set title based on the presence of frequency or phase error in time domain
    if (carrierFreq ~= 100000)
        title(sprintf("Time Domain With SNR= %d dB and Frequency Error", dB));
    elseif (phase ~= 0)
        title(sprintf("Time Domain With SNR= %d dB and Phase Error", dB));
    else
        title(sprintf("Time Domain With SNR= %d dB", dB));
    end

    % Resample the demodulated signal and play the audio
    demodSignal = resample(demodSignal, f_Sampling / 5, f_Sampling);
    sound(abs(demodSignal), f_Sampling / 5);
end
```

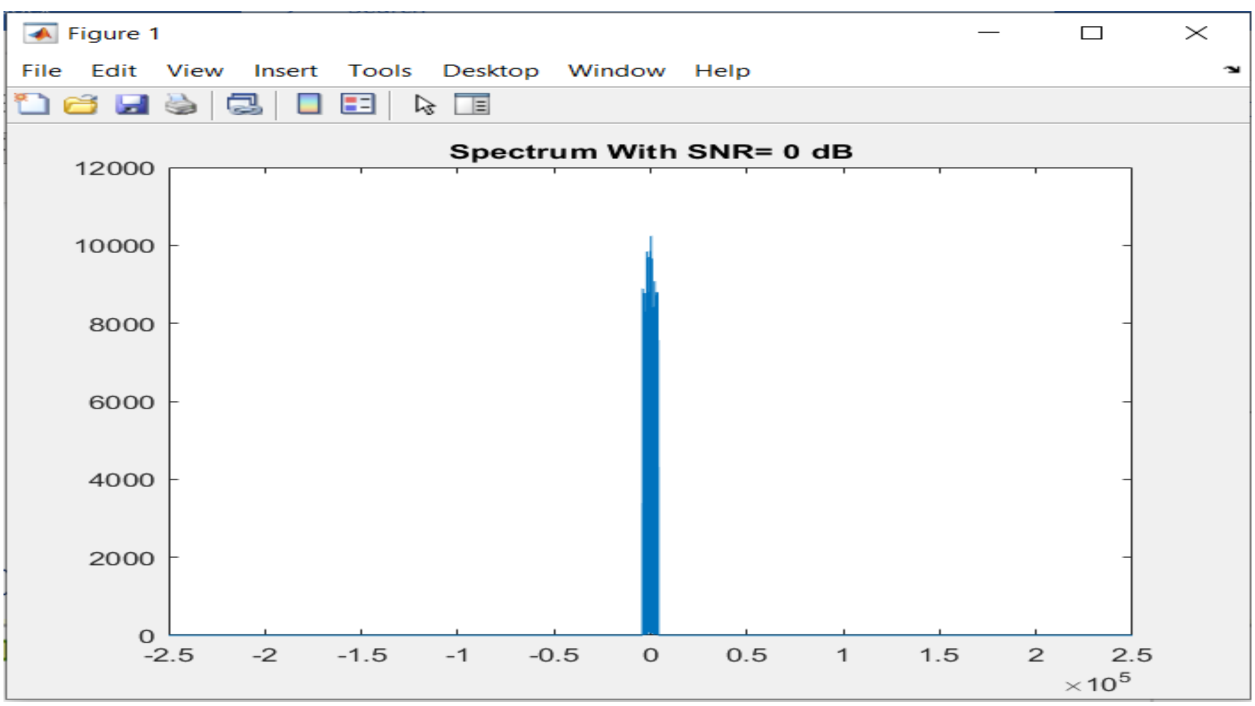
A) SNR = 0:

```
%% Coherent Detection with SNR = 0 dB
dB = 0;
phase = 0;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



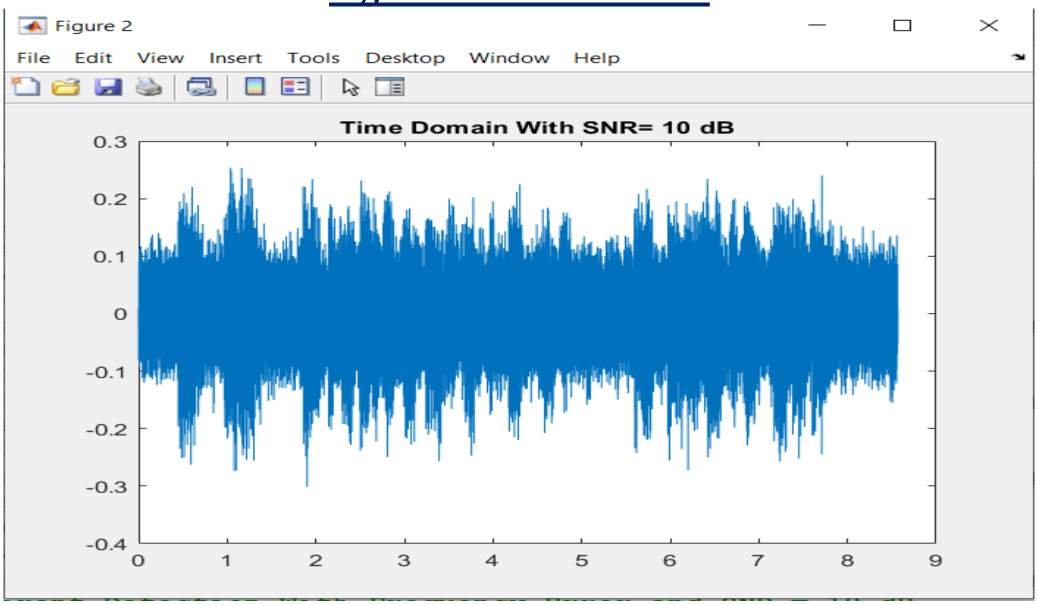
Signal in Frequency Domain:



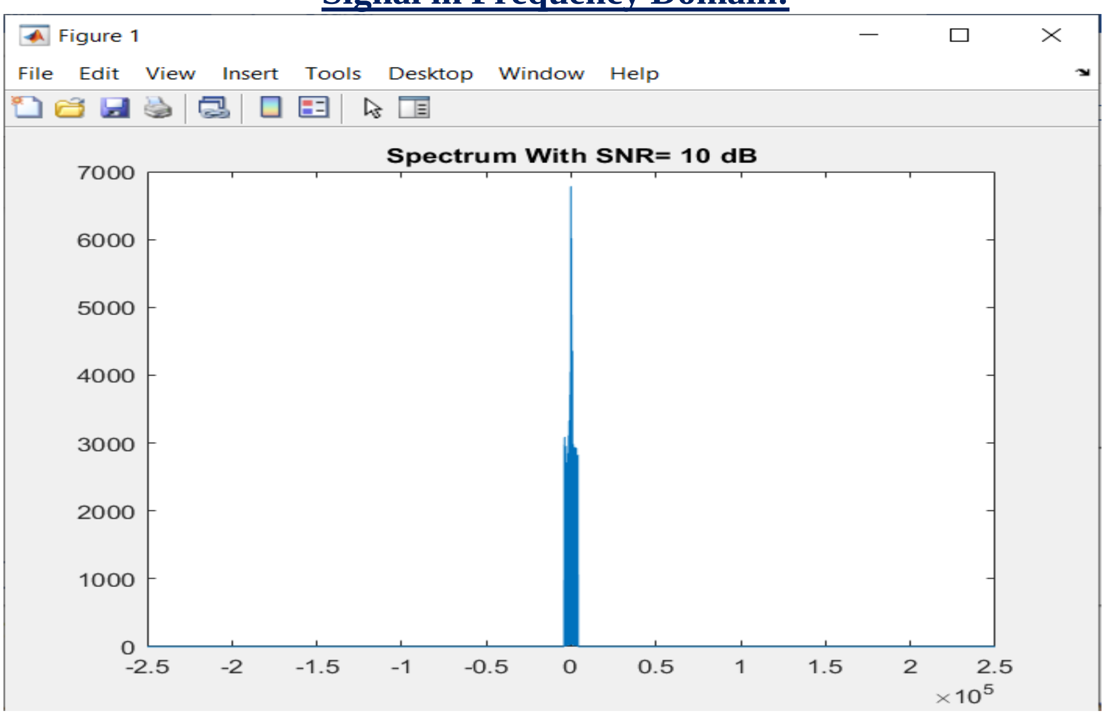
B) SNR = 10:

```
%% Coherent Detection with SNR = 10 dB
dB = 10;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



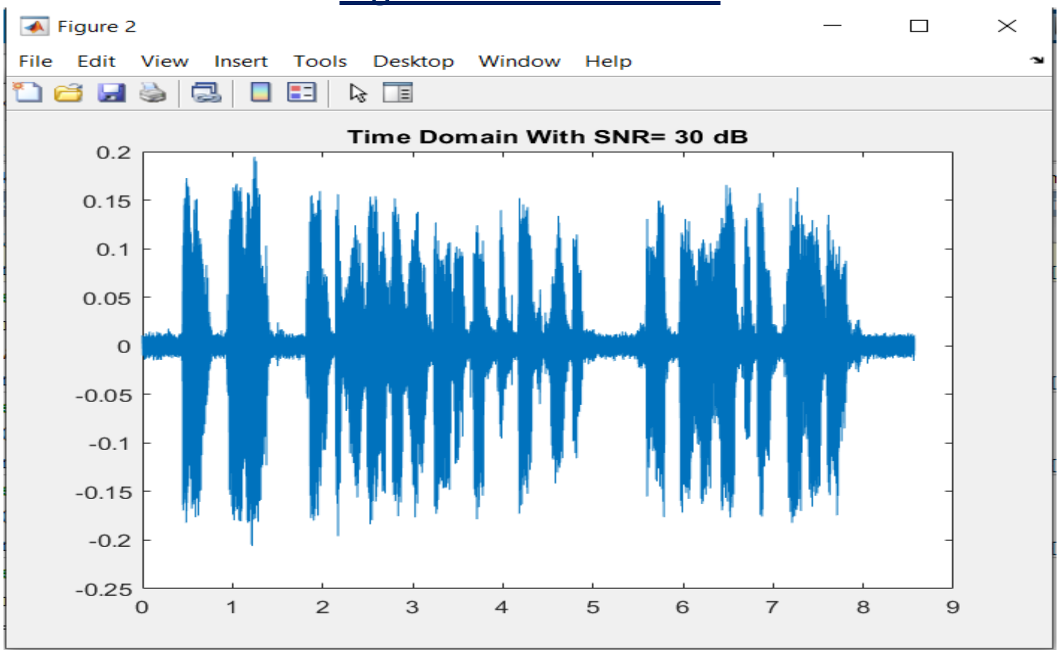
Signal in Frequency Domain:



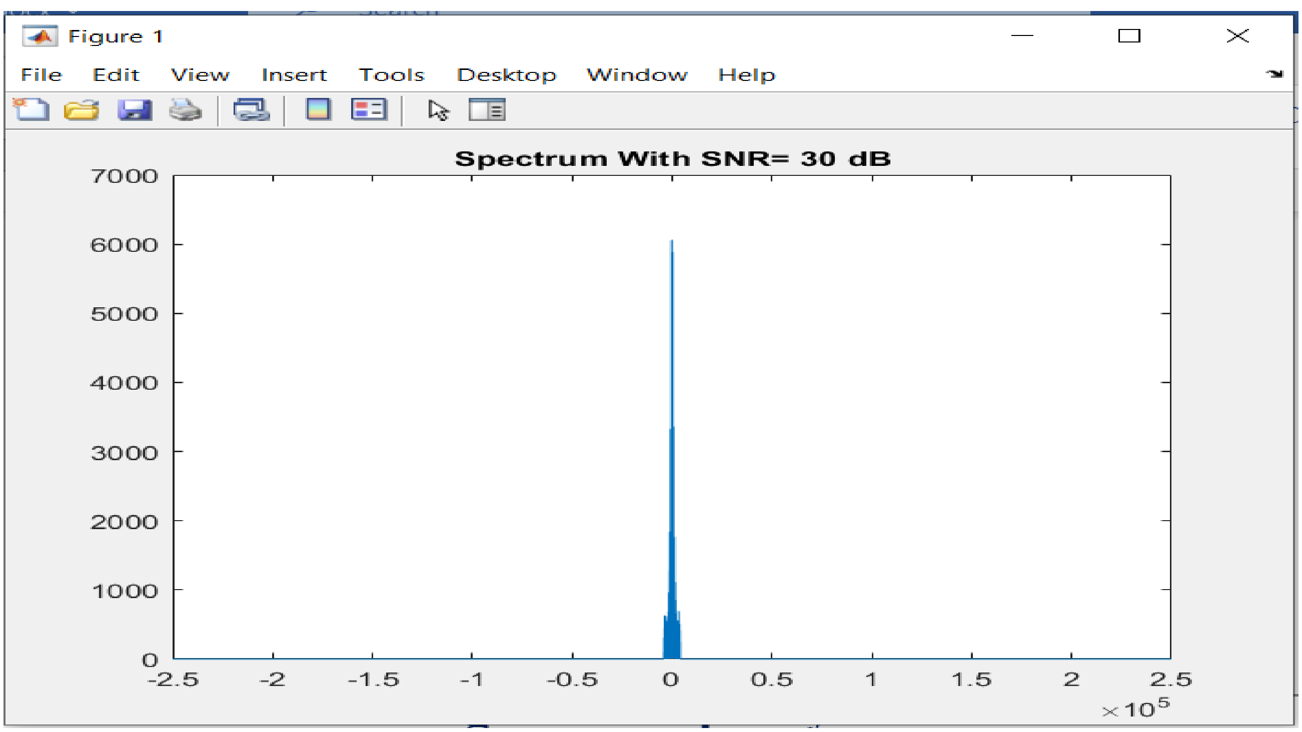
C) SNR=30:

```
%% Coherent Detection with SNR = 30 dB
dB = 30;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



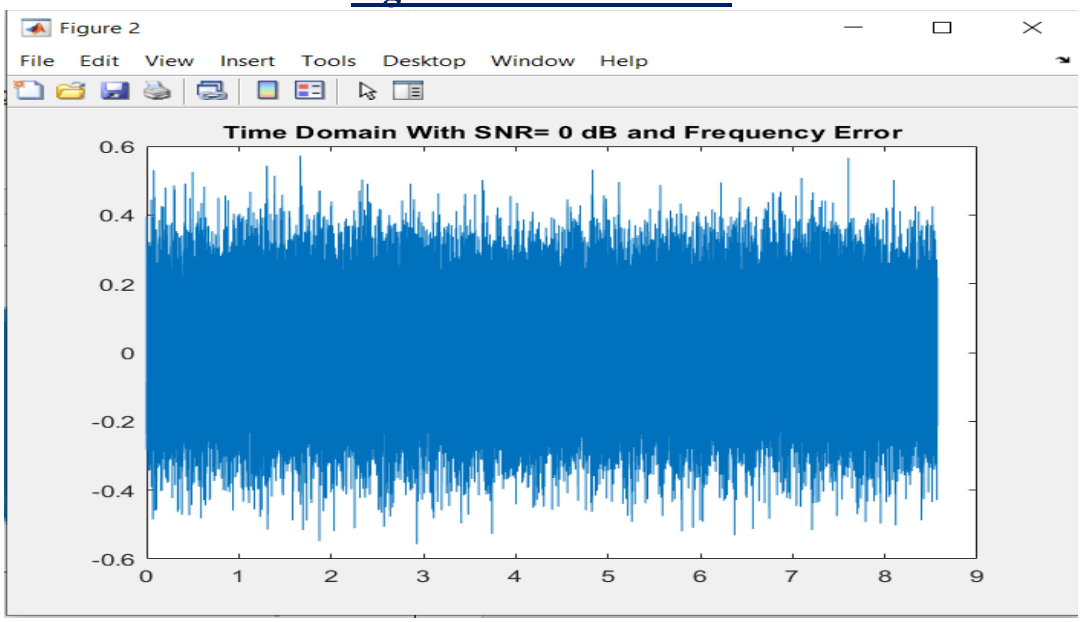
Signal in Frequency Domain:



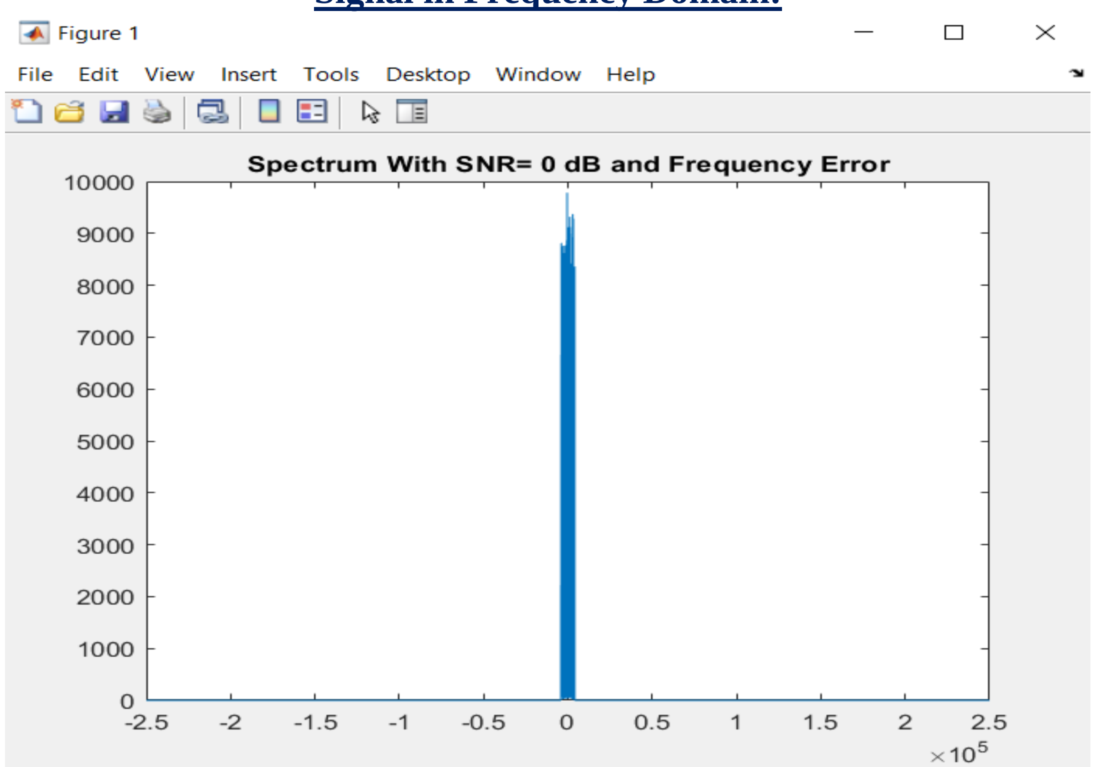
D) Frequency Shift and SNR=0:

```
%% Coherent Detection With Frequency Error and SNR = 0 dB
carrierFreq = 100100;
dB = 0;
coherentDetection(dB, DSB SC, carrierFreq, timeVector, cutoffFreq, f S, phase);
```

Signal in Time Domain:



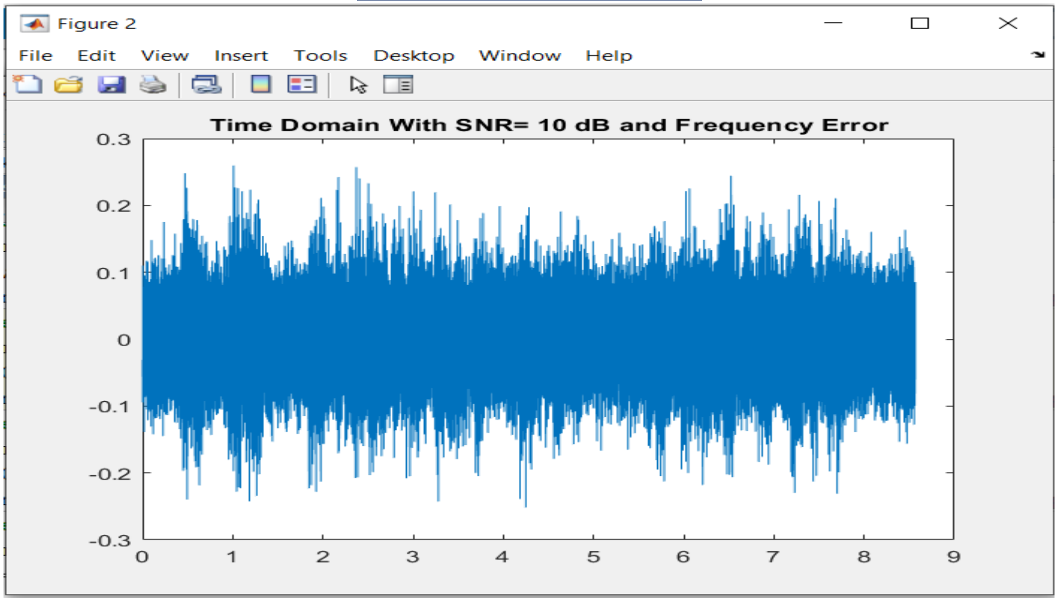
Signal in Frequency Domain:



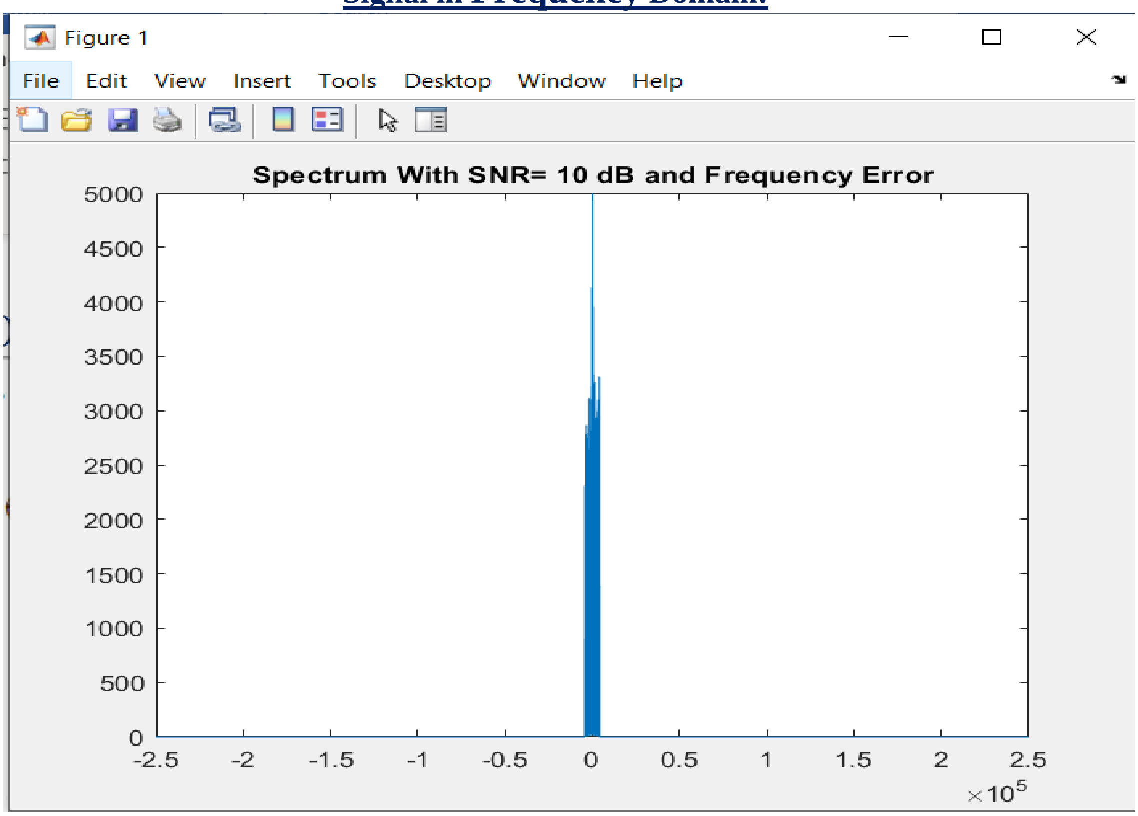
E) Frequency Shift and SNR=10:

```
%% Coherent Detection With Frequency Error and SNR = 10 dB
dB = 10;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



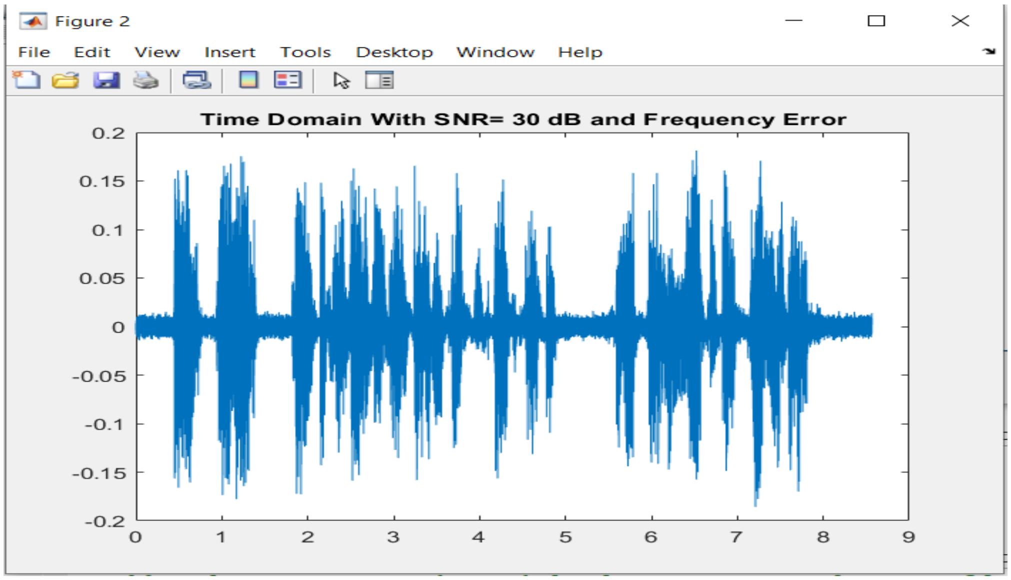
Signal in Frequency Domain:



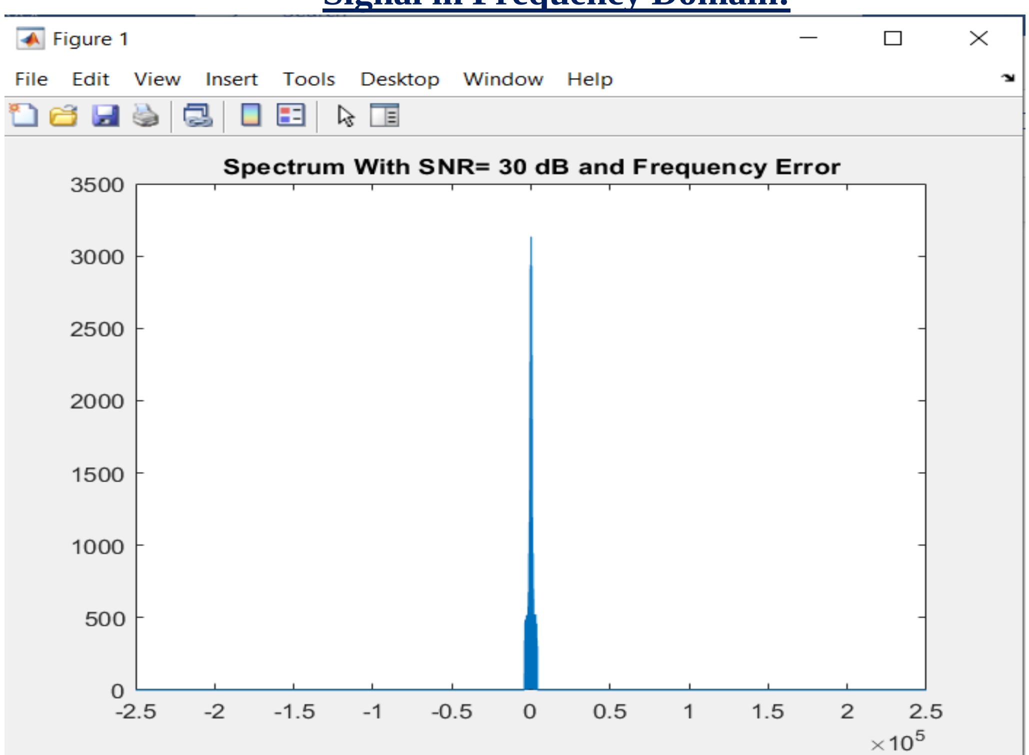
F) Frequency Shift and SNR=30:

```
%% Coherent Detection With Frequency Error and SNR = 30 dB
dB = 30;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



Signal in Frequency Domain:



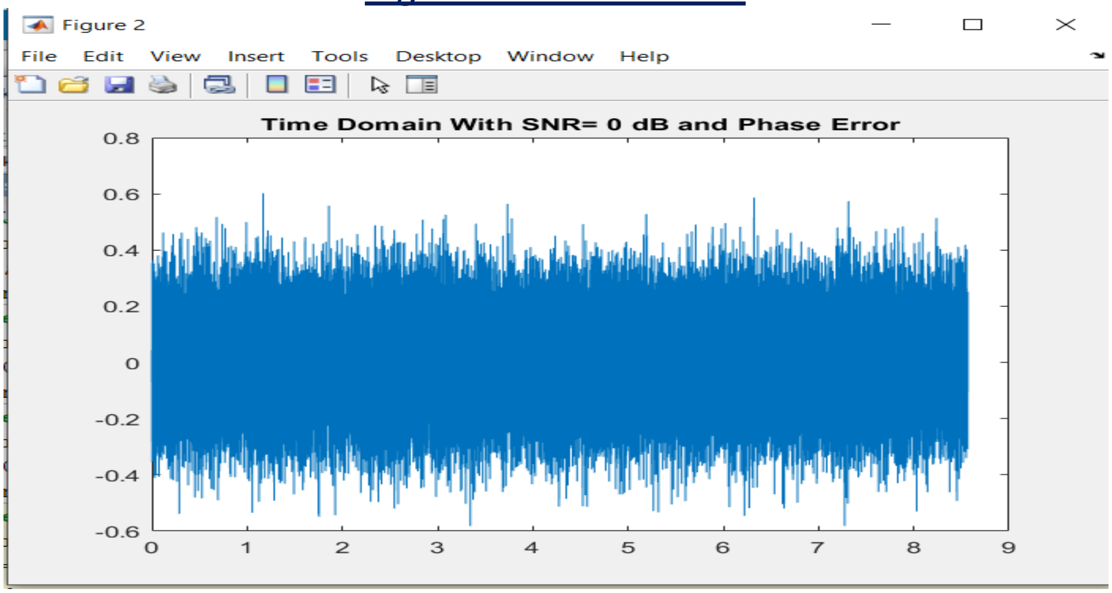
-For proper detection with coherent detector it shouldn't be any phase or frequency errors between the transmitter and the receiver so in our case we have here shift in frequency which leads to distortion in the received message and this phenomenon is called Beat effect.

The rest of the page is left blank intentionally.

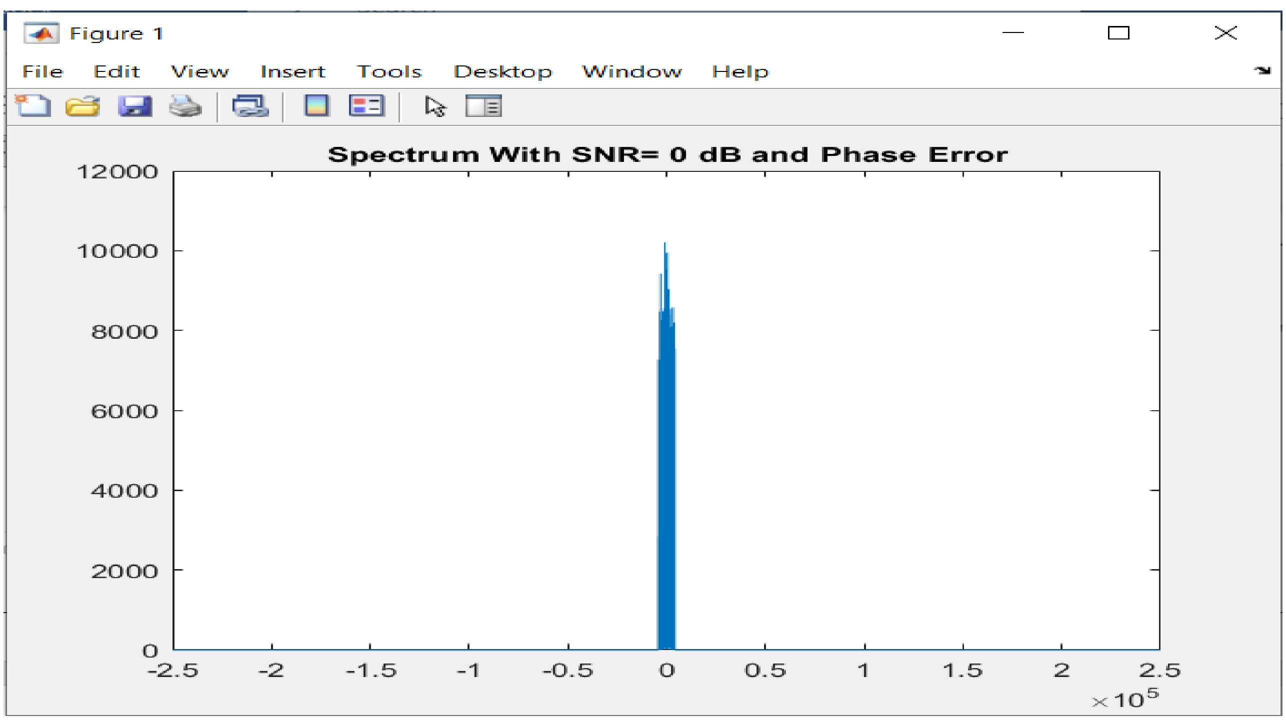
G)Phase Error and SNR=0:

```
%% Coherent Detection With Phase Error and SNR = 0 dB
carrierFreq = 100000;
phase = 20 * pi/180;
dB = 0;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



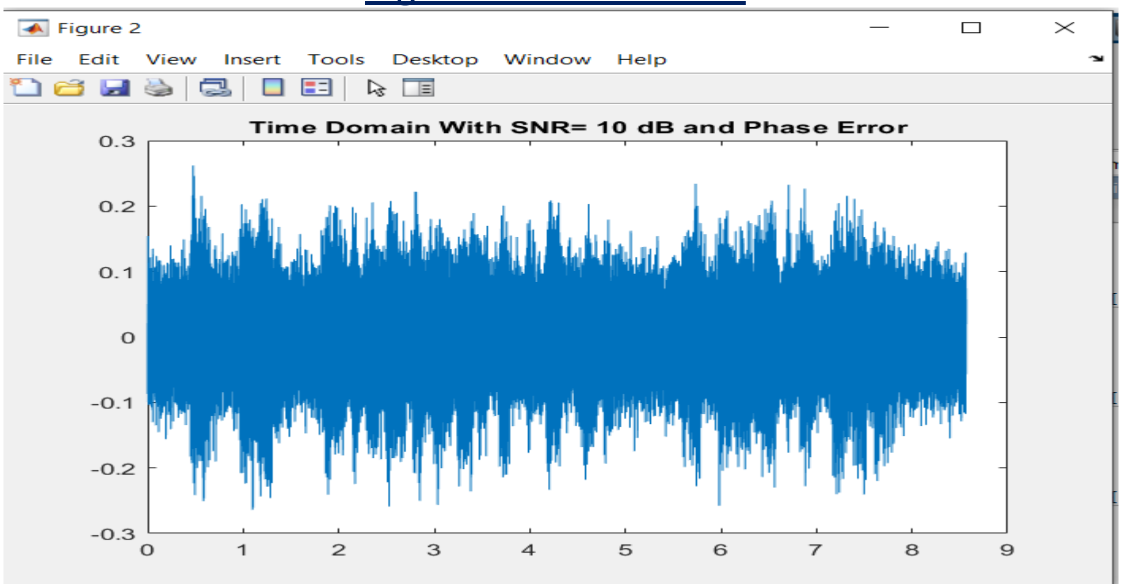
Signal in Frequency Domain:



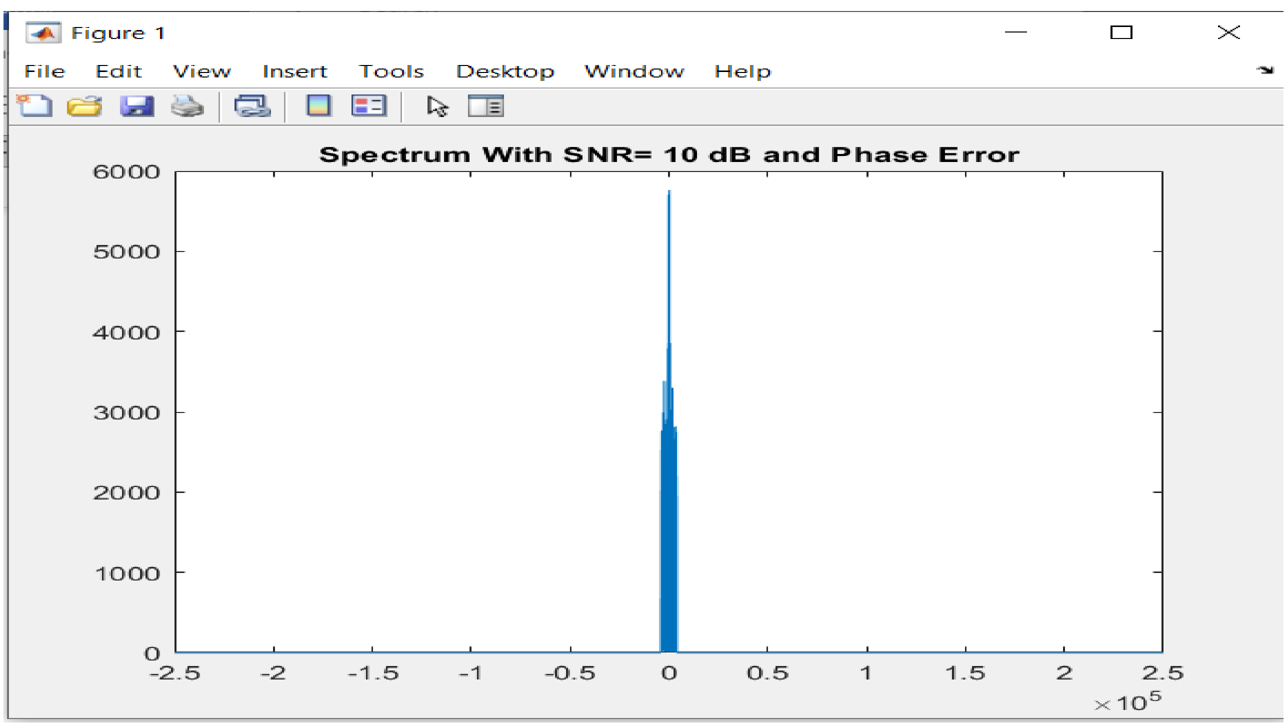
H)Phase Error and SNR=10:

```
%% Coherent Detection With Phase Error and SNR = 10 dB
carrierFreq = 100000;
phase = 20 * pi/180;
dB = 10;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



Signal in Frequency Domain:

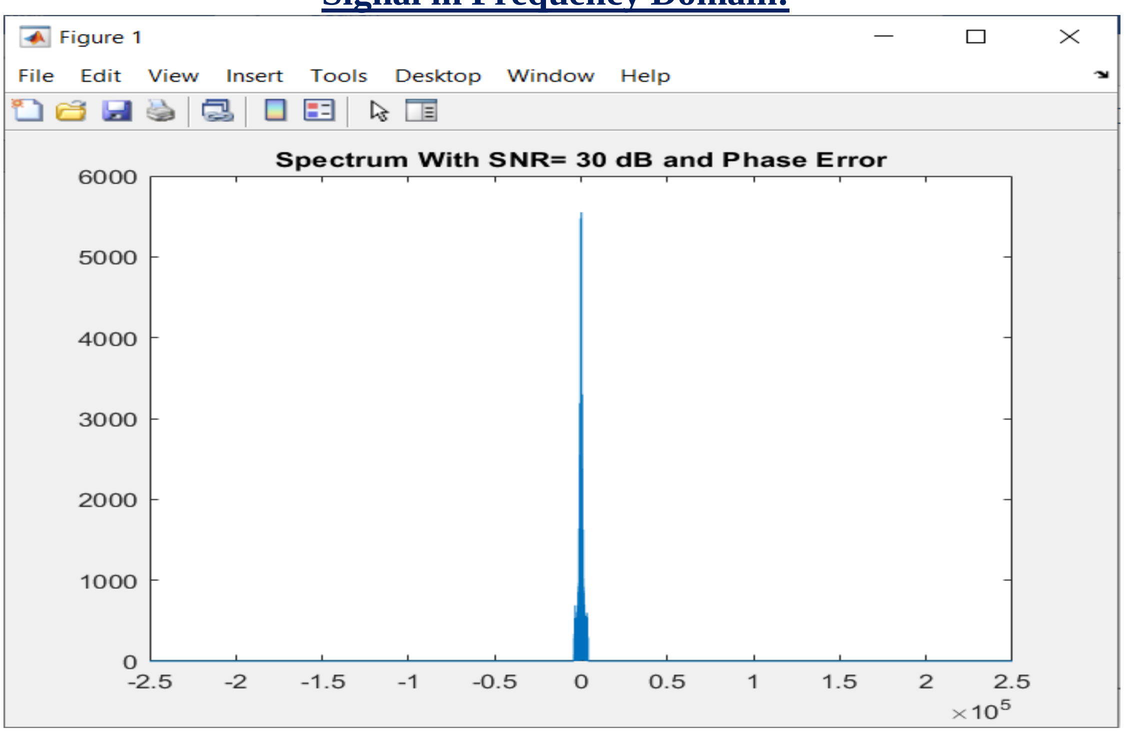


I) Phase Error and SNR=30:

```
%% Coherent Detection With Phase Error and SNR = 30 dB
carrierFreq = 100000;
phase = 20 * pi/180;
dB = 30;
coherentDetection(dB, DSB_SC, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:

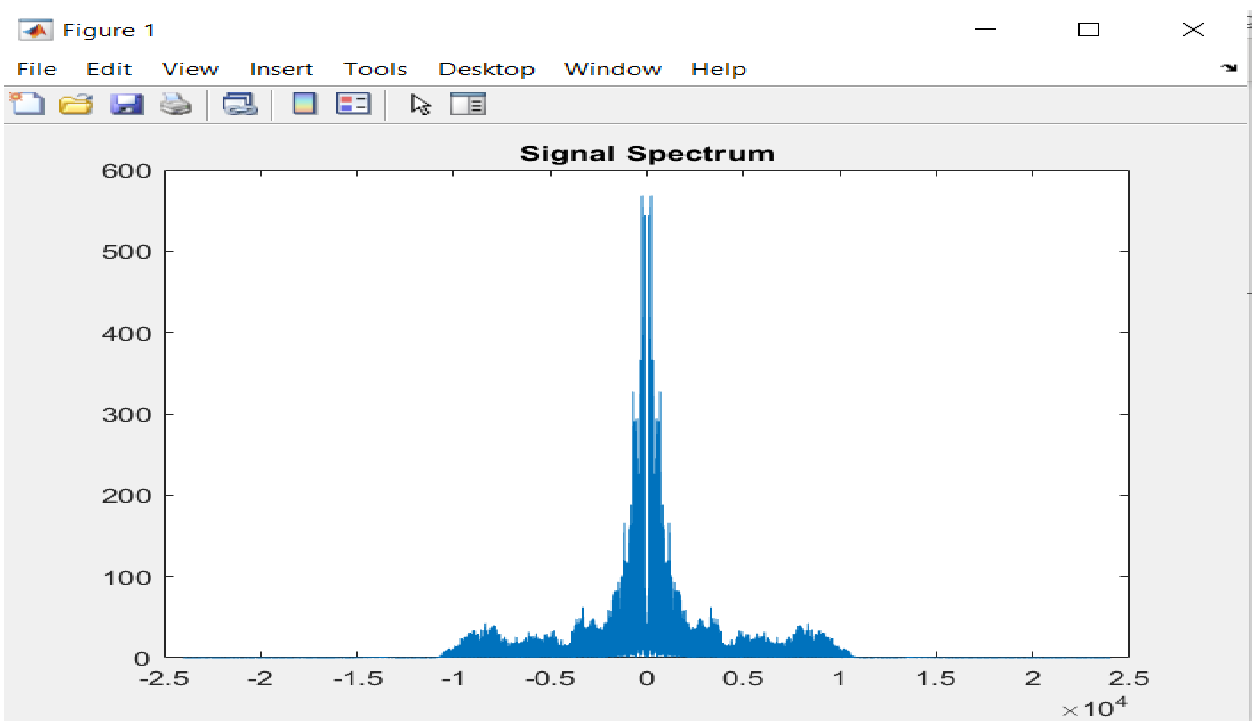
Signal in Frequency Domain:



Experiment Two: Single Side Band Modulation: -

```
function [signal, f_Sampling, S_freq, freq] = start(filename)
    % start analyzes the input audio file and computes its spectrum
    % Inputs:
    %   filename: Name of the audio file to be analyzed

    % Read the audio file and extract the signal and sampling frequency
    [signal, f_Sampling] = audioread(filename);

    % Compute the length of the signal
    len = length(signal);

    % Compute the frequency spectrum of the signal
    S_freq = fftshift(fft(signal));
    freq = (f_Sampling / 2) * linspace(-1, 1, len);

    % Plot the spectrum of the signal
    figure;
    plot(freq, abs(S_freq));
    title('Signal Spectrum'); % Set the title for the plot
end
```

Signal in frequency domain:

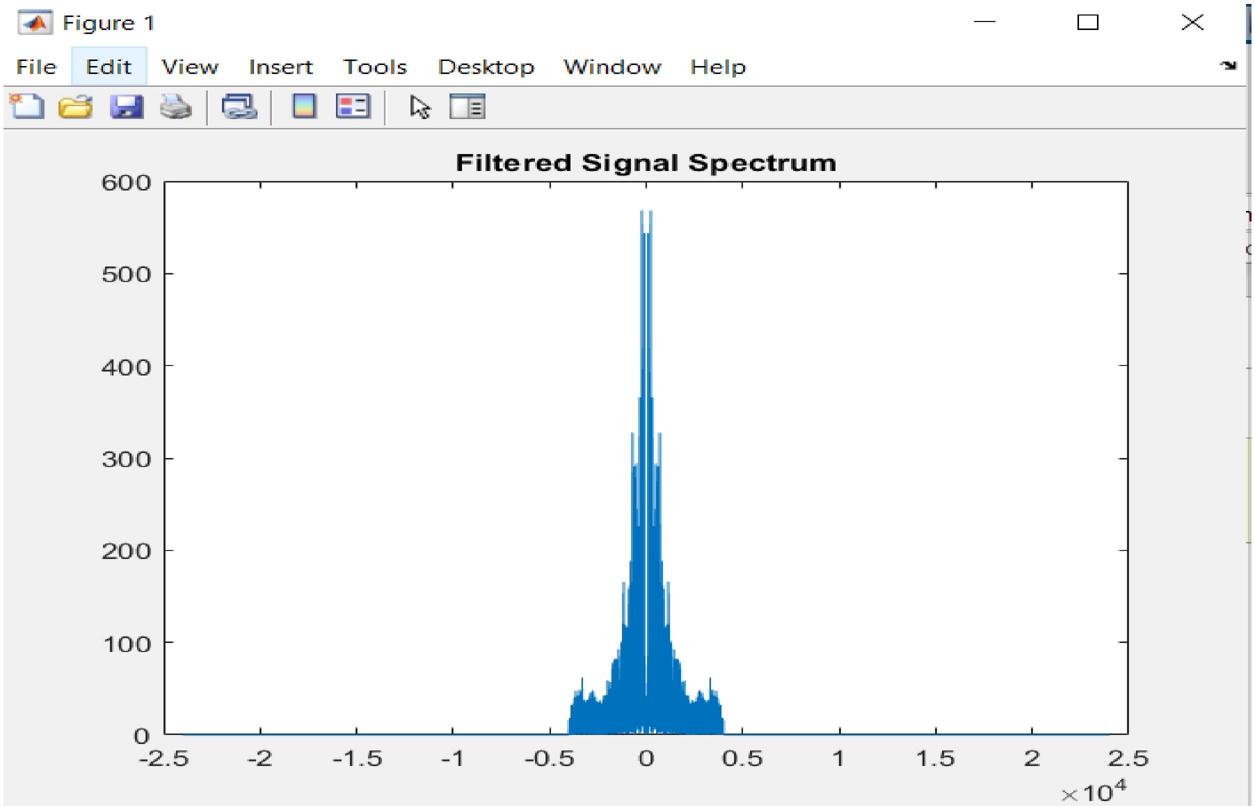


Filtered Signal:

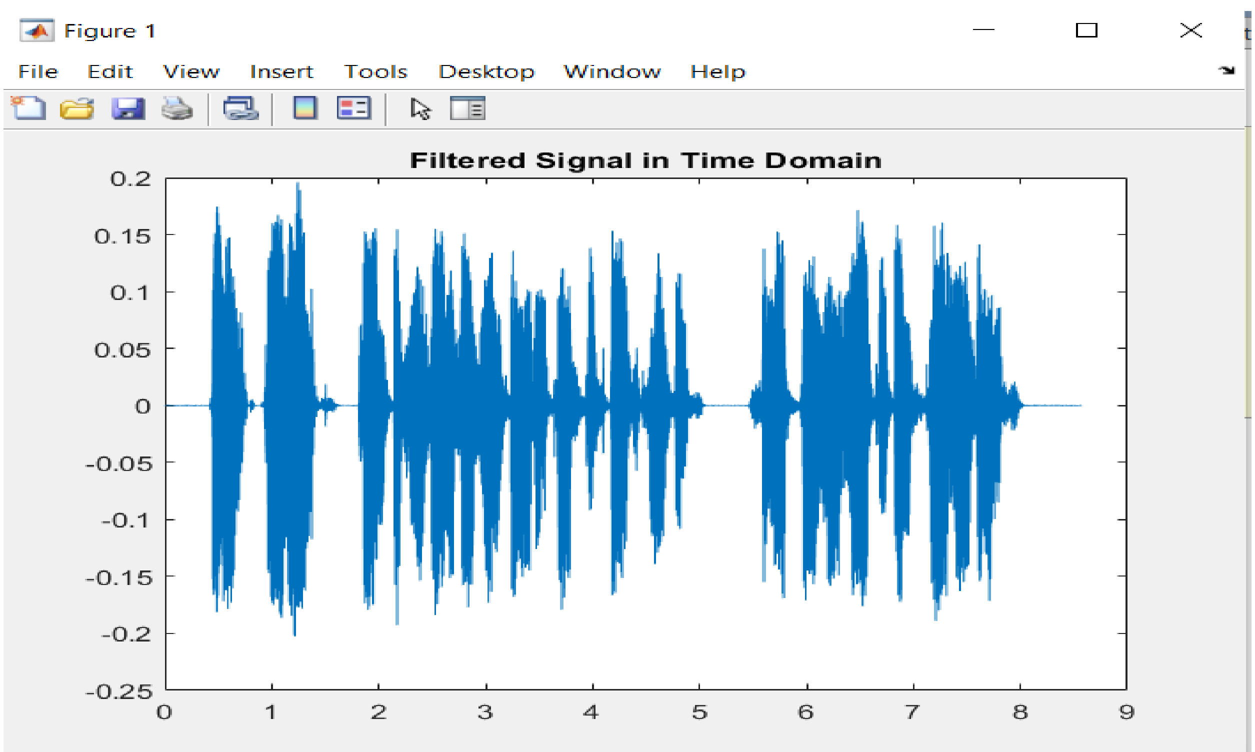
```
%% Filtering the Input Signal
cutoffFreq = 4000;
filteredSignal = filtering(cutoffFreq, S_freq, freq, f_S);

startTime = 0;
endTime = startTime + length(filteredSignal) / f_S;
timeVector = linspace(startTime, endTime, length(filteredSignal));
timeVector = timeVector';

figure;
plot(timeVector, filteredSignal);
title('Filtered Signal Time Domain');

sound(abs(filteredSignal), f_S);
```

```
function filteredSignal = filtering(cutoffFreq, S_freq, freq, f_Sampling)
    % filtering filters the input frequency-domain signal S_freq based on the cutoff frequency
    % Inputs:
    %   cutoffFreq: Frequency cutoff for filtering in Hz
    %   S_freq: Frequency domain signal
    %   freq: Frequency vector corresponding to the signal
    %   f_Sampling: Sampling frequency in Hz

    % Apply frequency domain filtering by zeroing out frequencies outside cutoff
    S_freq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Retrieve the filtered signal in time domain using inverse FFT
    filteredSignal = ifft(ifftshift(S_freq));

    % Compute the length of the filtered signal
    len = length(filteredSignal);

    % Compute the frequency spectrum of the filtered signal
    S_freq = fftshift(fft(filteredSignal));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the filtered signal
    figure;
    plot(freq, abs(S_freq));
    title('Filtered Signal Spectrum'); % Set the title for the plot
end
```

Filtered Signal in frequency domain:



Filtered Signal in Time Domain:



Double Sideband Suppressed Carrier:

```
function carrier = generateCarrier(carrierFreq, carrierAmp, time)
    % generateCarrier generates a carrier signal with specified parameters
    % Inputs:
    %   carrierFreq: Carrier frequency of the signal in Hz
    %   carrierAmp: Amplitude of the carrier signal
    %   time: Time vector for the carrier signal

    % Generate the carrier signal using the cosine function
    carrier = carrierAmp .* cos(2 * pi * carrierFreq * time);
end
```

```
%% Generating Double-Side-Band-Supprassed-Carrier
carrierFreq = 100000;
Amplitude = max(filteredSignal);
carrierAmp = 2 * Amplitude;

filteredSignal = resample(filteredSignal, 5 * carrierFreq, f_S);
f_S = 5 * carrierFreq;

startTime = 0;
endTime = startTime + length(filteredSignal)/f_S;
timeVector = linspace(startTime, endTime, length(filteredSignal));
timeVector = timeVector';

carrier = generateCarrier(carrierFreq, carrierAmp, timeVector);
DSB_SC = suppressedCarrier(carrier, filteredSignal, f_S);
```

```
function DSB_SC = suppressedCarrier(carrier, signal, f_Sampling)
    % suppressedCarrier generates a Double Side Band Suppressed Carrier (DSB-SC) signal
    % Inputs:
    %   carrier: Carrier signal
    %   signal: Input signal
    %   f_Sampling: Sampling frequency in Hz
    amp = max(signal);
    modIndex = 0.5;
    % Generate the DSB-SC signal by modulating the input signal with the carrier
    DSB_SC = modIndex * signal/amp .* carrier;

    % Compute the length of the DSB-SC signal
    len = length(DSB_SC);

    % Compute the frequency spectrum of the DSB-SC signal
    DSB_SC_Freq = fftshift(fft(DSB_SC));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the DSB-SC signal
    figure;
    plot(freq, abs(DSB_SC_Freq));
    title('Double Side Band Suppressed Carrier'); % Set the title for the plot
end

```

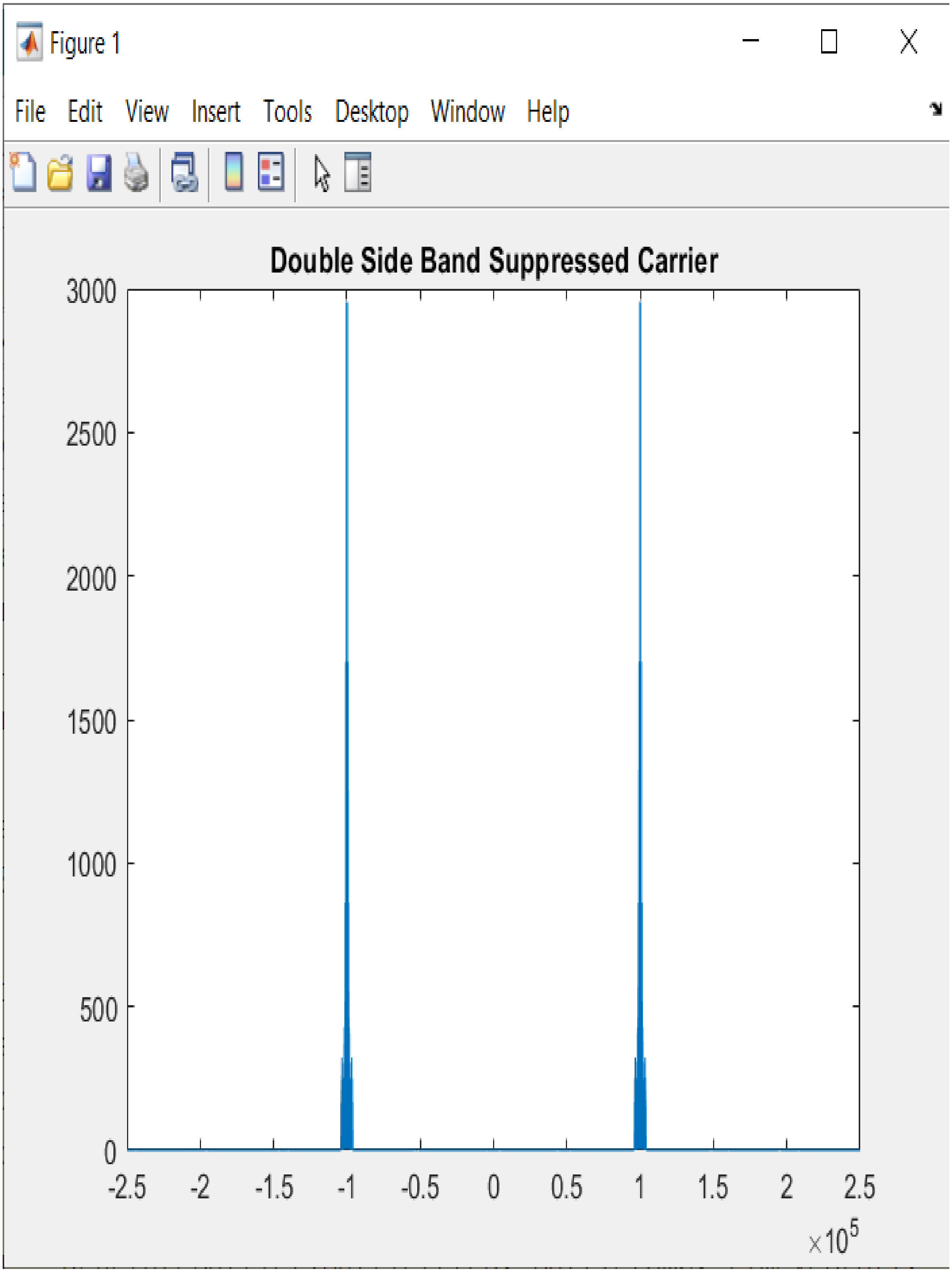
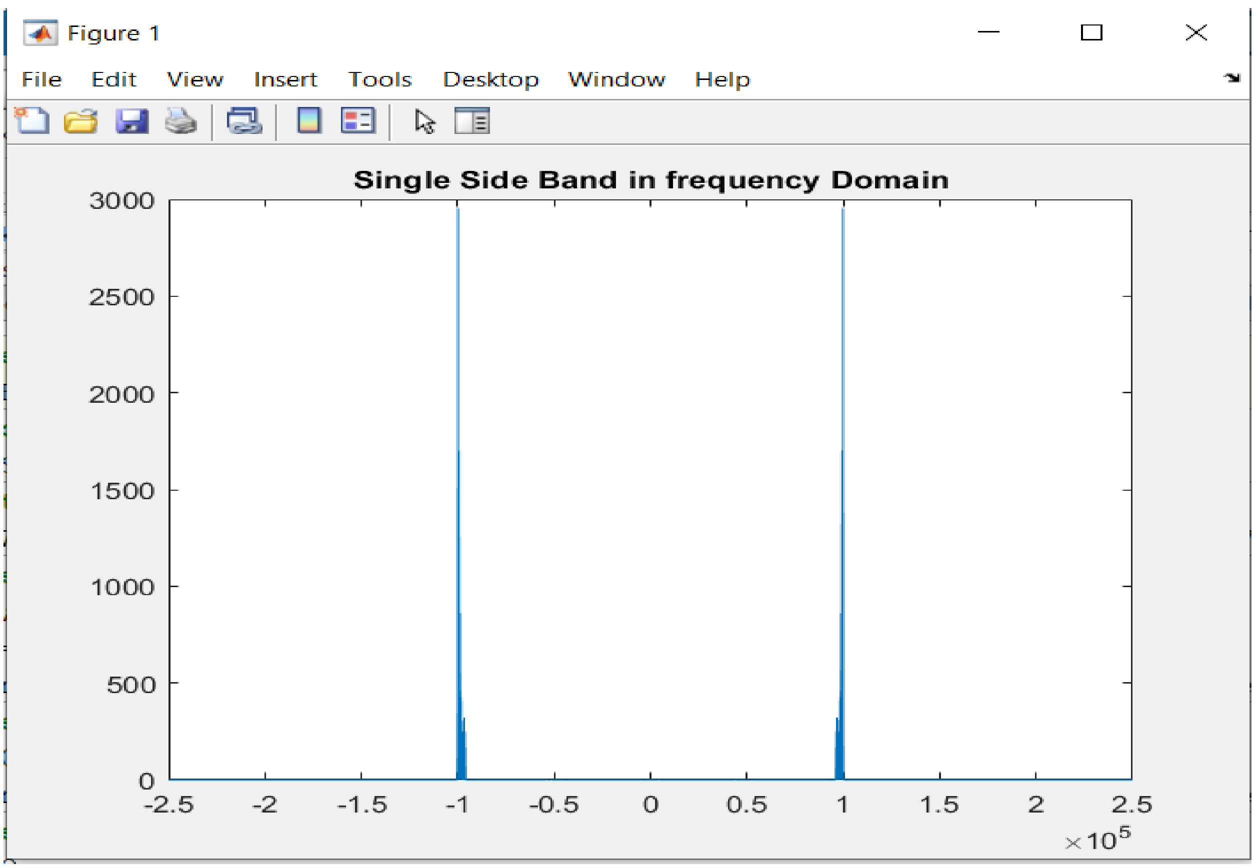


Figure 1
File Edit View Insert Tools Desktop Window Help
Double Side Band Suppressed Carrier
3000
2500
2000
1500
1000
500
0
-2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5
×10^5

Single Sided Band (LSB only):

```
%% Generating Single-Side-Band-LSB
SSB LSB = singleSideBand(DSB SC, f S, carrierFreq);
```

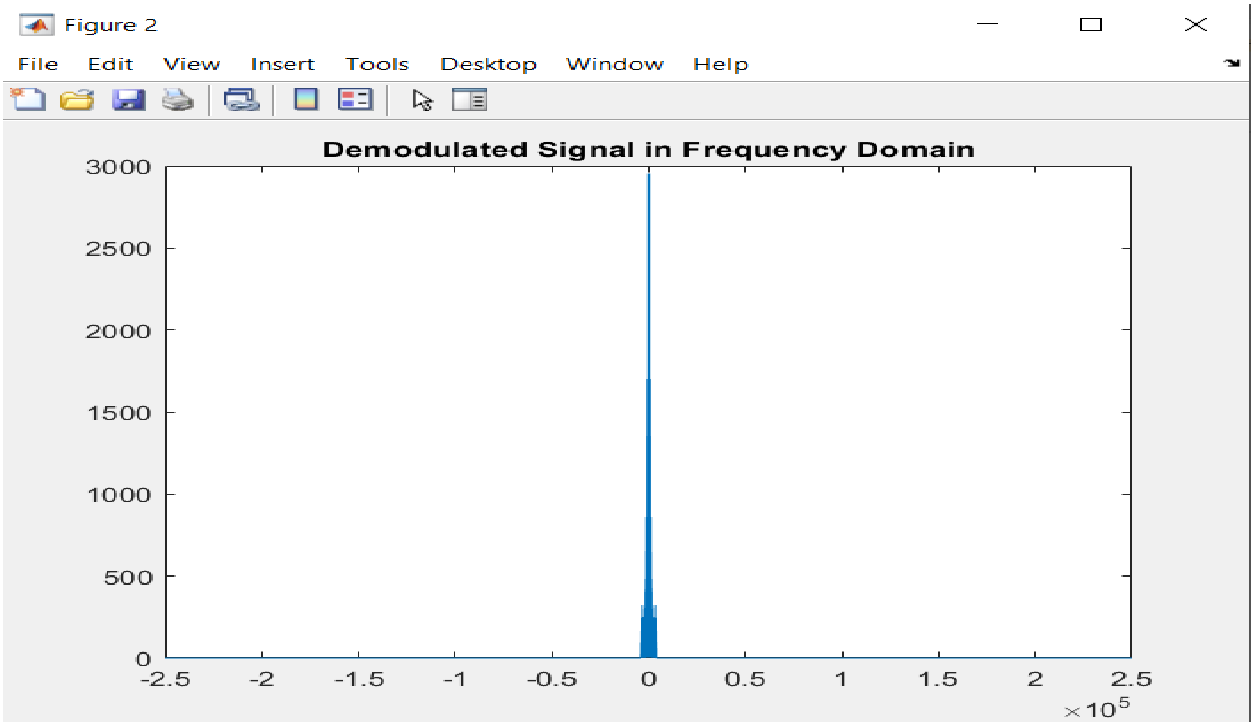
```
function SSB_LSB = singleSideBand(DSB_SC, f_Sampling, carrierFreq)
    % singleSideBand - Generates Single Side Band (SSB) modulation by suppressing one sideband.
    % Inputs:
    %   DSB_SC: Double Side Band Suppressed Carrier signal.
    %   f_Sampling: Sampling frequency of the input signal.
    %   carrierFreq: Carrier frequency used in modulation.

    % Copy the input signal to SSB_LSB
    SSB_LSB = DSB_SC;

    % Calculate length and frequency vector
    len = length(SSB_LSB);
    freq = f_Sampling/2 * linspace(-1, 1, len);

    % Compute frequency domain representation
    S_freq = fftshift(fft(SSB_LSB));

    % Suppress one sideband by setting frequencies outside the desired range to zero
    S_freq(freq >= carrierFreq | freq <= -carrierFreq) = 0;

    % Perform inverse FFT to get the time-domain signal
    SSB_LSB = ifft(ifftshift(S_freq));

    % Plot the magnitude of the frequency domain representation
    figure;
    plot(freq, abs(S_freq));
    title('Single Side Band in frequency Domain');
end
```



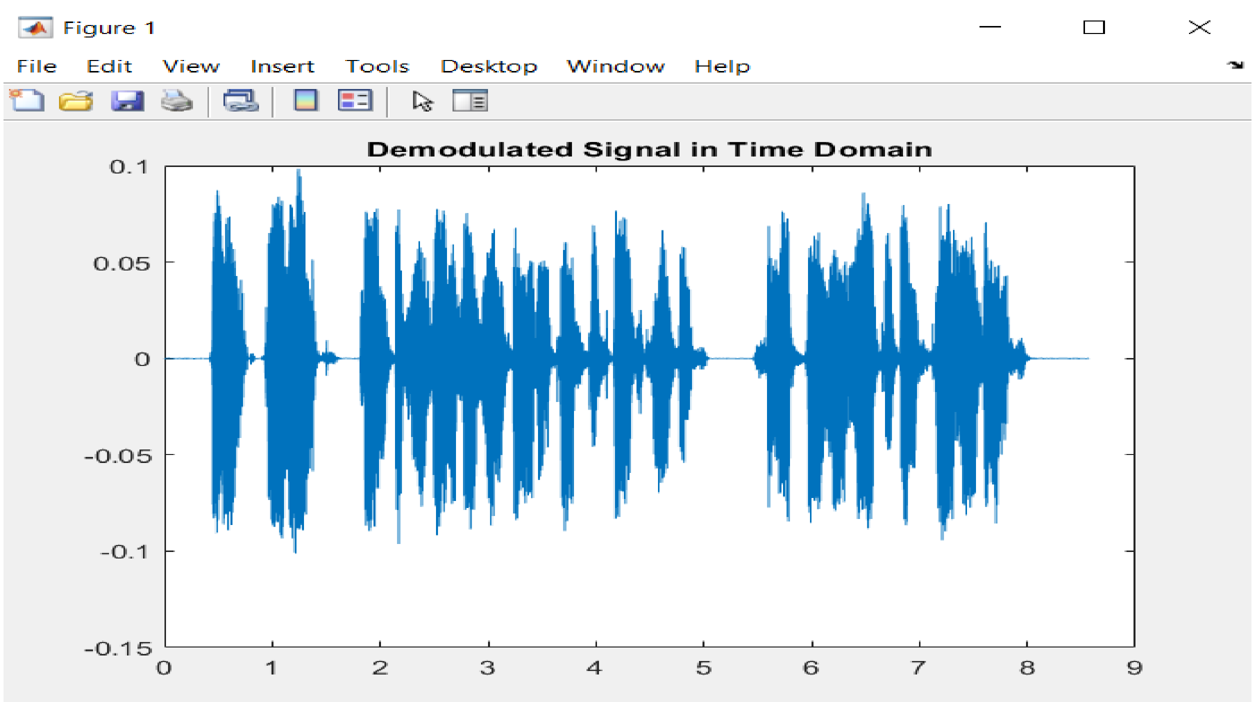
Demodulated signal spectrum:

```
%% Demodulating
singleSideBandDemodulation(SSB_LSB, carrierFreq, timeVector, f_S, cutoffFreq);
```

```
function singleSideBandDemodulation(SSB, carrierFreq, timeVector, f_S, cutoffFreq)
    % singleSideBandDemodulation - Performs demodulation of Single Side Band (SSB) signal.
    % Inputs:
    %   SSB: Single Side Band modulated signal.
    %   carrierFreq: Carrier frequency used in modulation.
    %   timeVector: Time vector corresponding to the signal.
    %   f_S: Sampling frequency of the signal.
    %   cutoffFreq: Cutoff frequency used for demodulation.

    modIndex = 0.5;
    % Demodulate the SSB signal
    demodSignal = SSB/modIndex .* cos(2 * pi * carrierFreq * timeVector);

    % Calculate length and frequency vectors
    len = length(SSB);
    freq = f_S/2 * linspace(-1, 1, len);

    % Compute frequency domain representation of the demodulated signal
    demodFreq = fftshift(fft(demodSignal));

    % Filter out frequencies outside the desired range for demodulation
    demodFreq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Perform inverse FFT to obtain the demodulated signal in time domain
    demodSignal = ifft(ifftshift(demodFreq));

    % Plot the demodulated signal in the time domain
    figure;
    plot(timeVector, demodSignal);
    title('Demodulated Signal in Time Domain');

    % Obtain the frequency domain representation of the demodulated signal
    len = length(demodSignal);
    freq = f_S/2 * linspace(-1, 1, len);
    S_freq = fftshift(fft(demodSignal));

    % Plot the demodulated signal in the frequency domain
    figure;
    plot(freq, abs(S_freq));
    title('Demodulated Signal in Frequency Domain');

    % Resample and play the demodulated signal
    demodSignal = resample(abs(demodSignal), f_S/5, f_S);
    sound(abs(demodSignal), f_S/5);
end
```

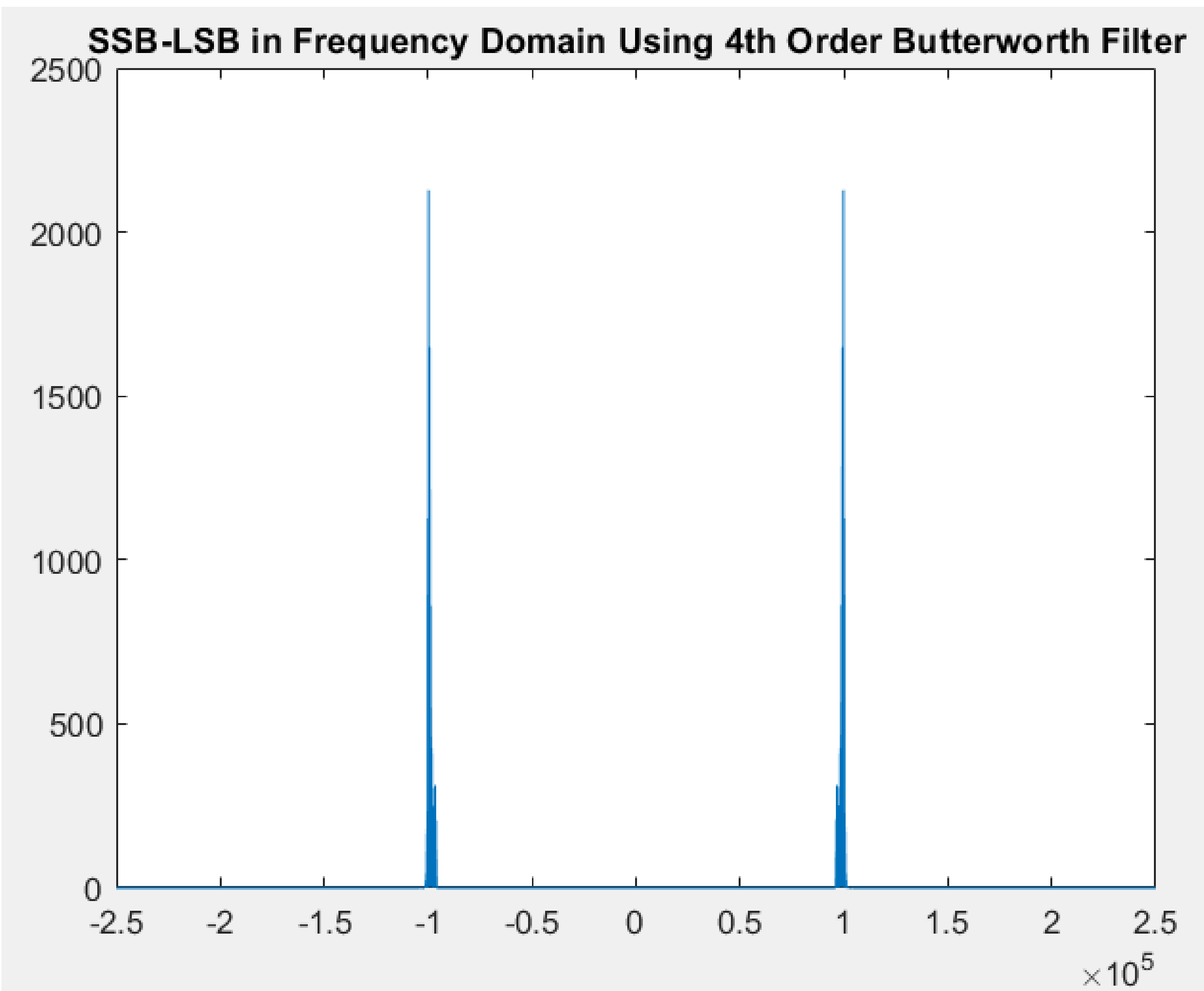


Demodulated signal in time domain:



Obtaining SSB (LSB) Using a practical 4th order Butterworth filter:

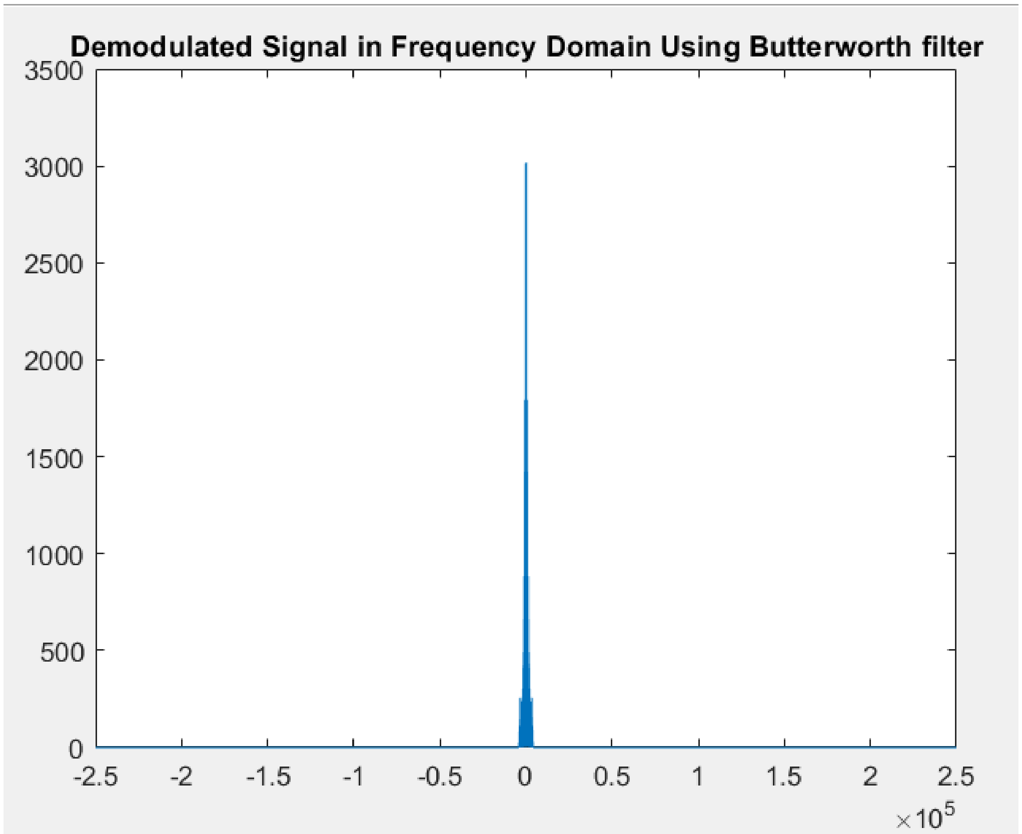
```
%% Generating Single-Side-Band-LSB Using Butterworth Filter
normalization = f_S/2;
fc1 = (carrierFreq - cutoffFreq)/normalization;
fc2 = carrierFreq/normalization;
[b, a] = butter(4,[fc1 fc2]);
SSB_LSB_BW = filtfilt(b, a, DSB_SC);
len = length(SSB_LSB_BW);
freq = f_S/2 * linspace(-1, 1, len);
S_freq = fftshift(fft(SSB_LSB_BW));

    % Plot the demodulated signal in the frequency domain
figure;
plot(freq, abs(S_freq));
title("SSB-LSB in Frequency Domain Using 4th Order Butterworth Filter");
```



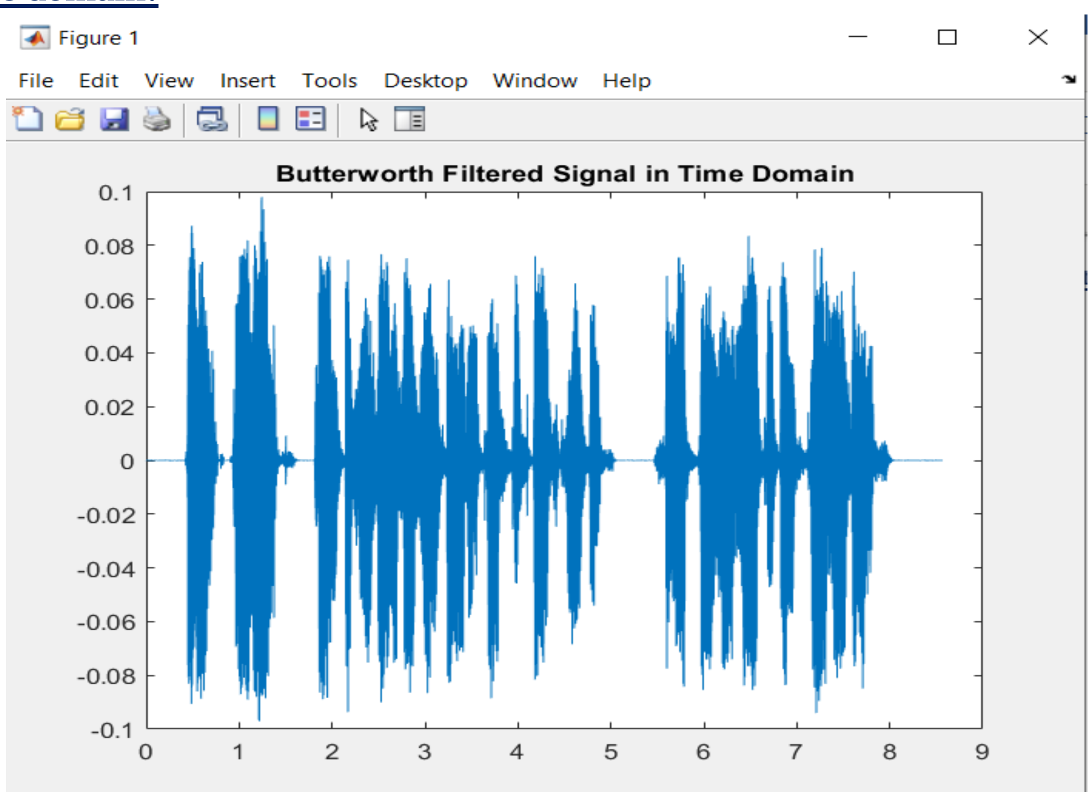
Demodulated signal spectrum Using a practical 4th order Butterworth filter:

```
%% ButterWorth Filter
butterWorthFiltering(cutoffFreq, f_S, SSB_LSB, carrierFreq, timeVector);
```

```
function butterWorthFiltering(cutoffFreq, f_S, signal, carrierFreq, timeVector)
    % butterWorthFiltering - Applies Butterworth filtering and demodulation to a signal.
    % Inputs:
    %   cutoffFreq: Cutoff frequency of the Butterworth filter.
    %   f_S: Sampling frequency of the input signal.
    %   signal: Input signal to be processed.
    %   carrierFreq: Carrier frequency for demodulation.
    %   timeVector: Time vector corresponding to the signal.

    % Design the Butterworth filter
    [b, a] = butter(4, cutoffFreq * 2/f_S);
    modIndex = 0.5;
    % Demodulate the signal
    demodSignal = signal/modIndex .* cos(2 * pi * carrierFreq * timeVector);

    % Apply the Butterworth filter using filtfilt
    demodSignal = filtfilt(b, a, demodSignal);

    % Plot the filtered signal in the time domain
    figure;
    plot(timeVector, demodSignal);
    title('Butterworth Filtered Signal in Time Domain');

    % Calculate frequency domain representation
    len = length(demodSignal);
    freq = f_S/2 * linspace(-1, 1, len);
    S_freq = fftshift(fft(demodSignal));

    % Plot the demodulated signal in the frequency domain
    figure;
    plot(freq, abs(S_freq));
    title('Demodulated Signal in Frequency Domain Using Butterworth filter');

    % Resample the signal and play the sound
    demodSignal = resample(abs(demodSignal), f_S/5, f_S);
    sound(abs(demodSignal), f_S/5);
end
```



Demodulated signal spectrum Using a practical 4th order Butterworth filter in time domain:



➢ Coherent Detection:

```
function coherentDetection(dB, signal, carrierFreq, time, cutoffFreq, f_Sampling, phase)
    % coherentDetection performs coherent detection of a modulated signal
    % Inputs:
    %   dB: Signal-to-noise ratio in decibels
    %   signal: Input signal to be detected
    %   carrierFreq: Carrier frequency of the signal
    %   time: Time vector corresponding to the signal
    %   cutoffFreq: Frequency cutoff for filtering in Hz
    %   f_Sampling: Sampling frequency in Hz
    %   phase: Phase of the carrier signal in radians

    % Add white Gaussian noise to the input signal based on given SNR
    snr_dB = awgn(signal, dB);

    % Demodulate the signal using coherent detection
    demodSignal = snr_dB .* cos(2 * pi * carrierFreq * time + phase);

    % Compute the frequency spectrum of the demodulated signal
    demodSignalFreq = fftshift(fft(demodSignal));
    len = length(demodSignal);
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Apply frequency domain filtering by zeroing out frequencies outside cutoff
    demodSignalFreq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Plot the frequency spectrum
    figure;
    plot(freq, abs(demodSignalFreq)      );

    % Set title based on the presence of frequency or phase error
    if (carrierFreq ~= 100000)
        title(sprintf("Spectrum With SNR= %d dB and Frequency Error", dB));
    elseif (phase ~= 0)
        title(sprintf("Spectrum With SNR= %d dB and Phase Error", dB));
    else
        title(sprintf("Spectrum With SNR= %d dB", dB));
    end

    % Retrieve the demodulated signal from frequency domain
    demodSignal = ifft(ifftshift(demodSignalFreq));

    % Plot the demodulated signal in time domain
    figure;
    plot(time, demodSignal);
```

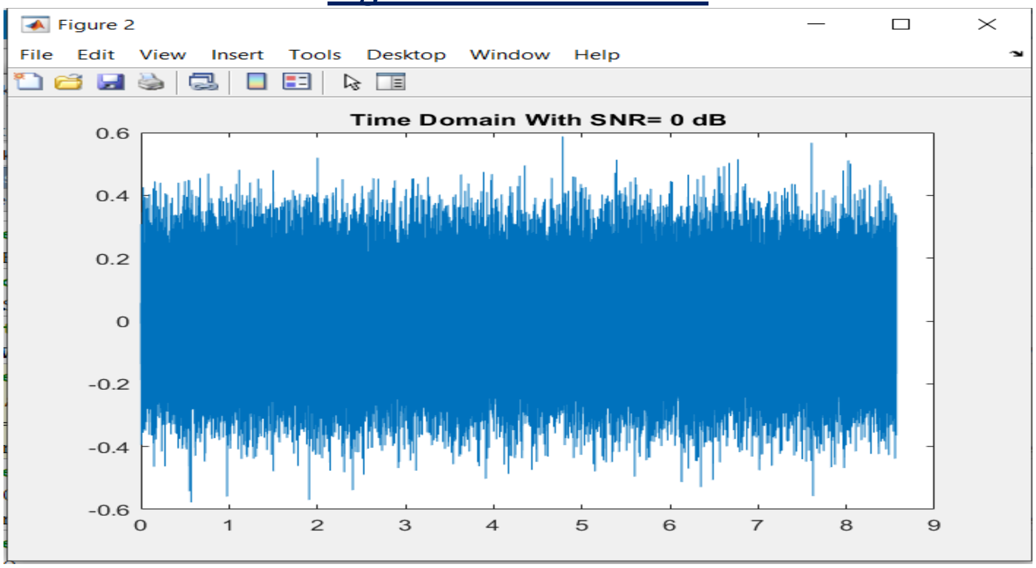
```
    % Set title based on the presence of frequency or phase error in time domain
    if (carrierFreq ~= 100000)
        title(sprintf("Time Domain With SNR= %d dB and Frequency Error", dB));
    elseif (phase ~= 0)
        title(sprintf("Time Domain With SNR= %d dB and Phase Error", dB));
    else
        title(sprintf("Time Domain With SNR= %d dB", dB));
    end

    % Resample the demodulated signal and play the audio
    demodSignal = resample(demodSignal, f_Sampling / 5, f_Sampling);
    sound(abs(demodSignal), f_Sampling / 5);
end
```

The rest of the page is left blank intentionally.

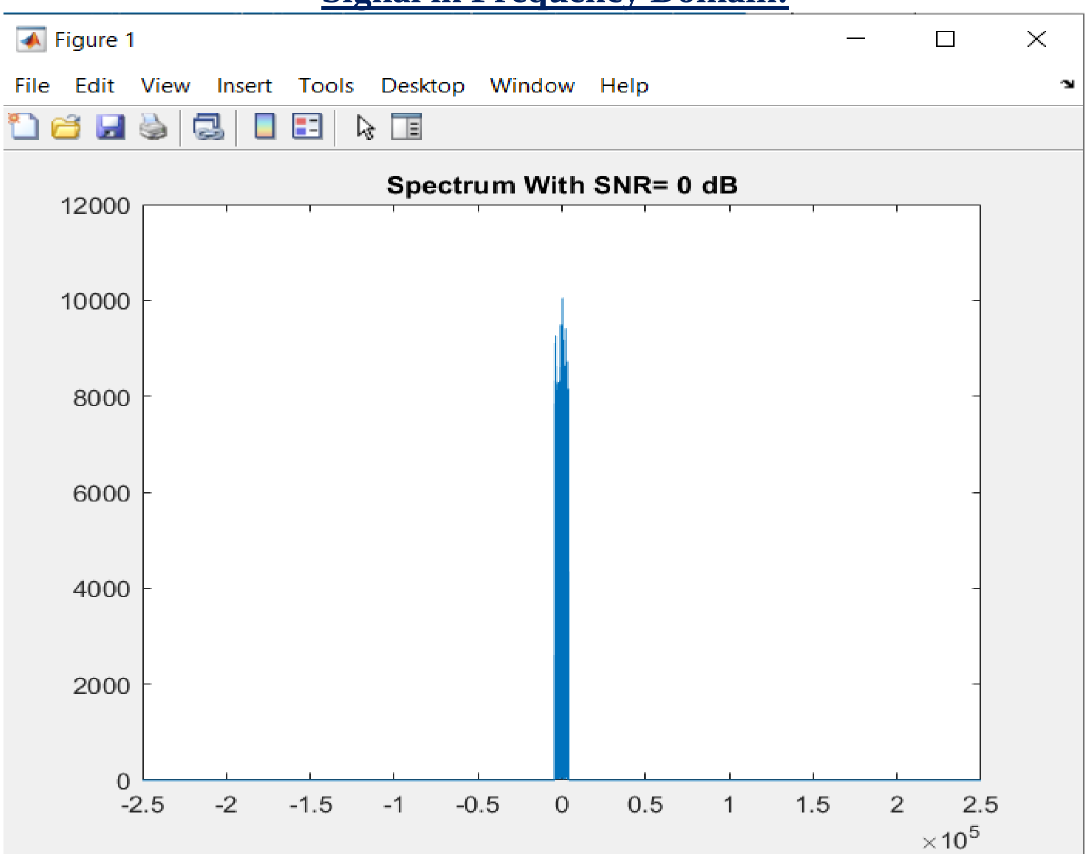
A) SNR = 0:

```
%% Coherent Detection With SNR = 0 dB
dB = 0;
phase = 0;
coherentDetection(dB, SSB_LSB, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



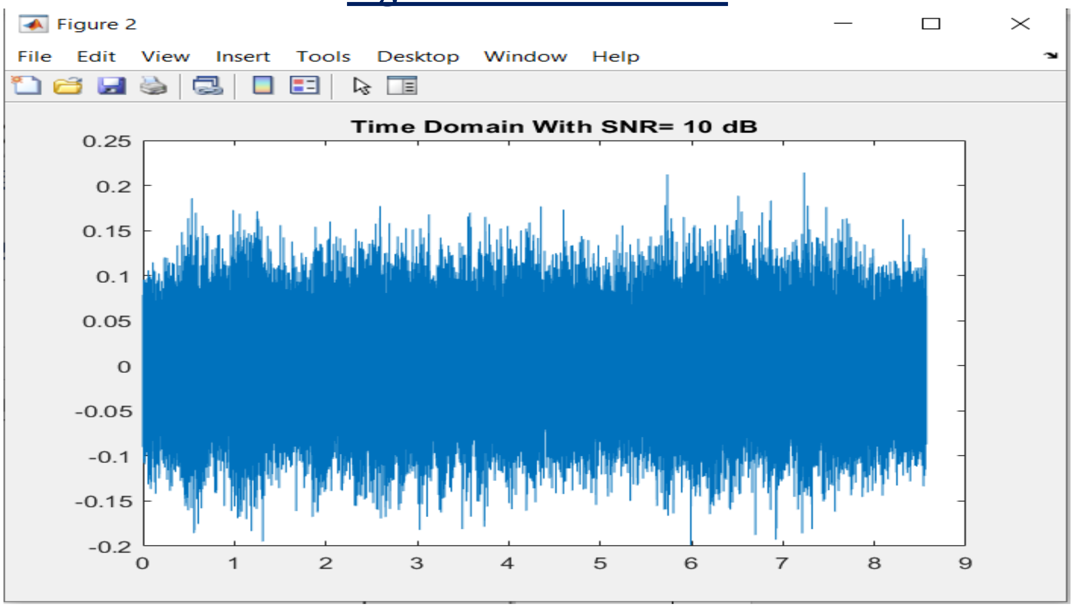
Signal in Frequency Domain:



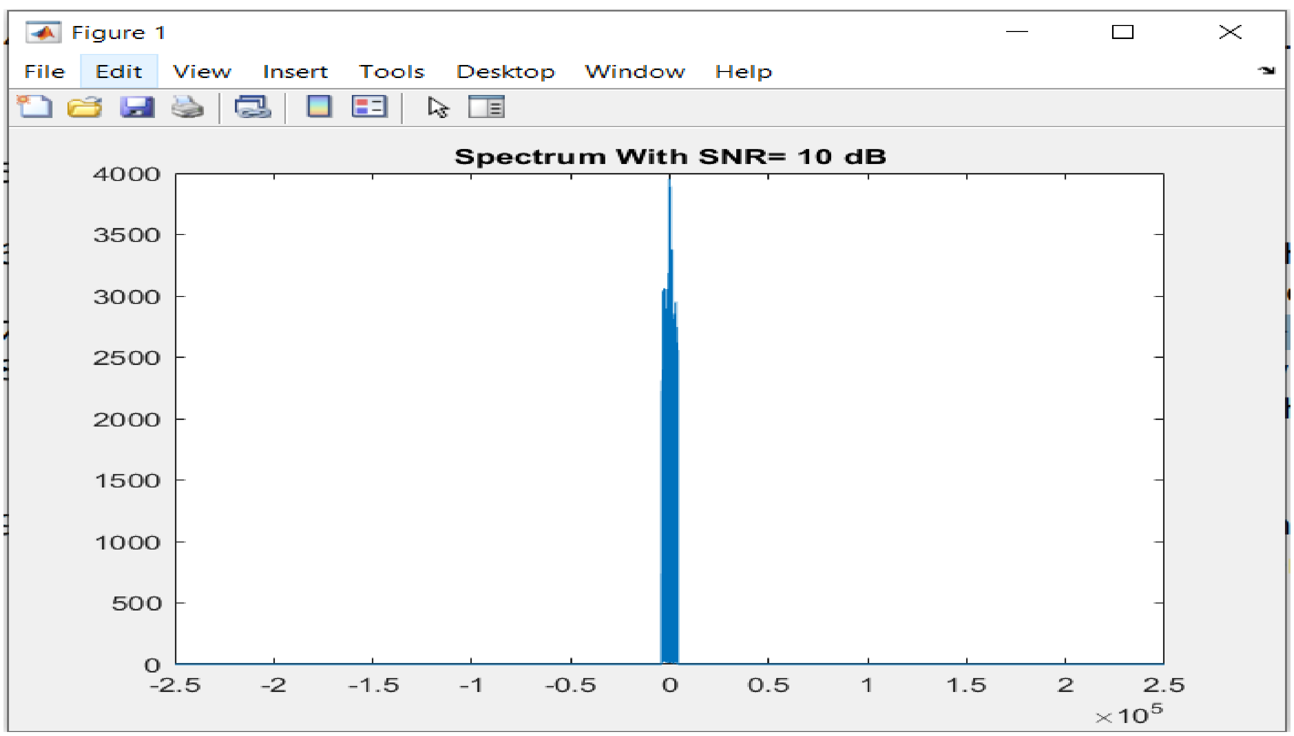
B) SNR = 10:

```
%% Coherent Detection With SNR = 10 dB
dB = 10;
coherentDetection(dB, SSB_LSB, carrierFreq, timeVector, cutoffFreq, f_S, phase);
```

Signal in Time Domain:



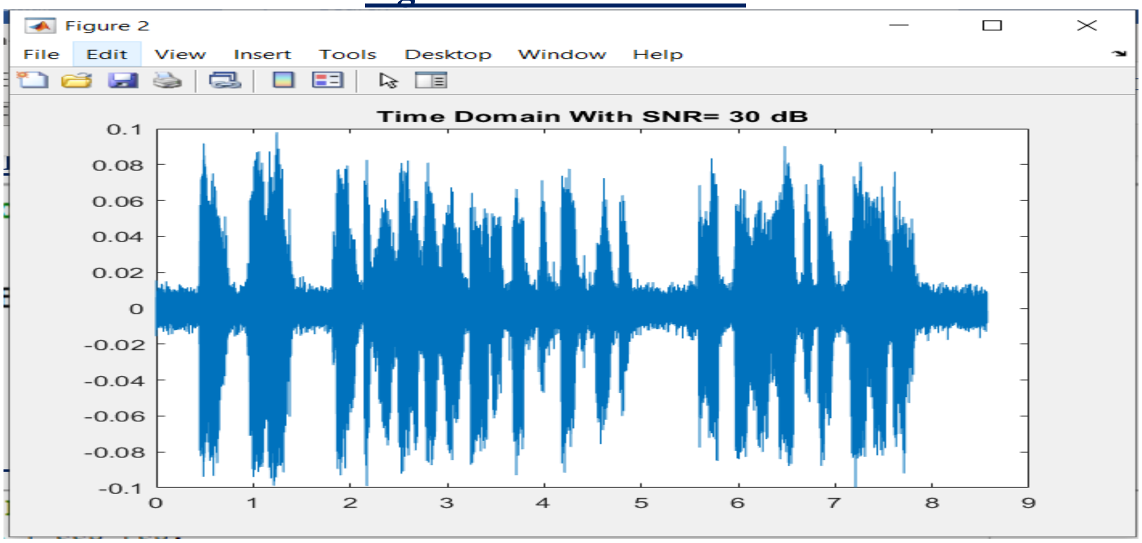
Signal in Frequency Domain:



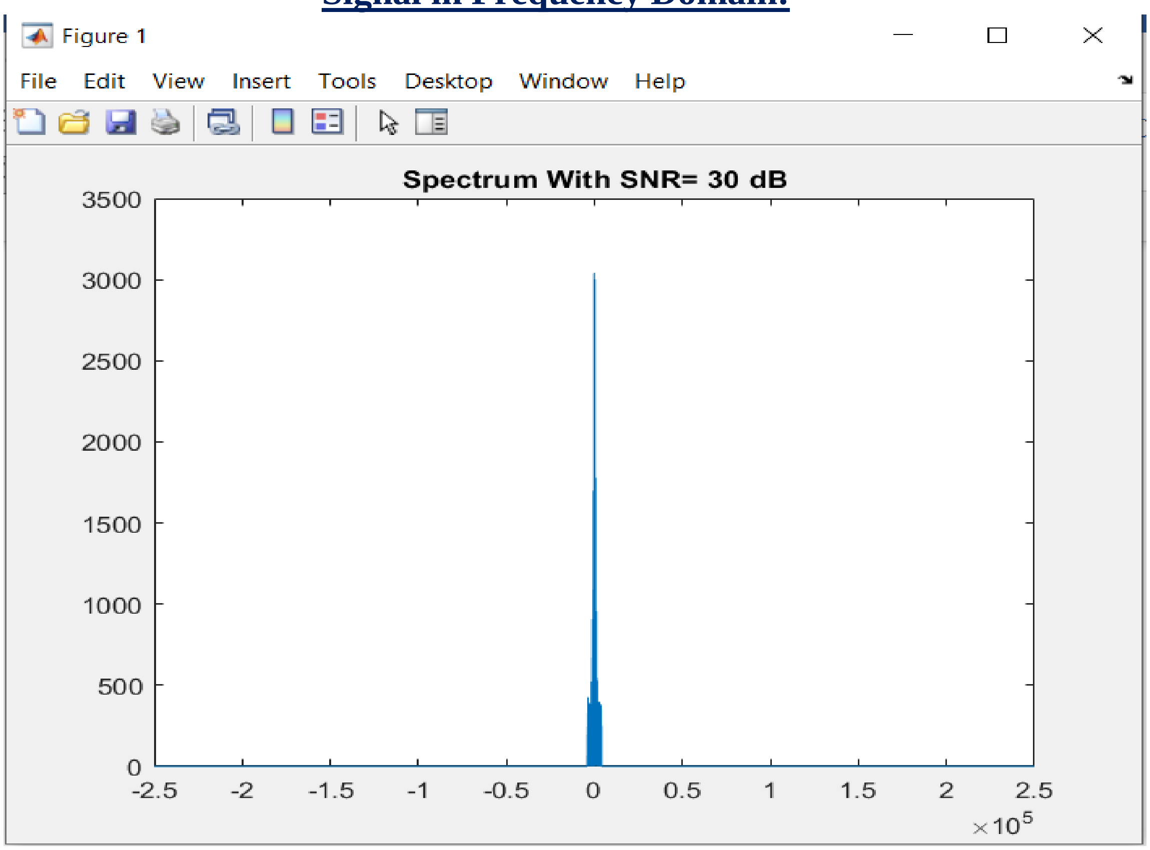
C) SNR=30:

```
%% Coherent Detection With SNR = 30 dB
dB = 30;
coherentDetection(dB, SSB LSB, carrierFreq, timeVector, cutoffFreq, f S, phase);
```

Signal in Time Domain:



Signal in Frequency Domain:

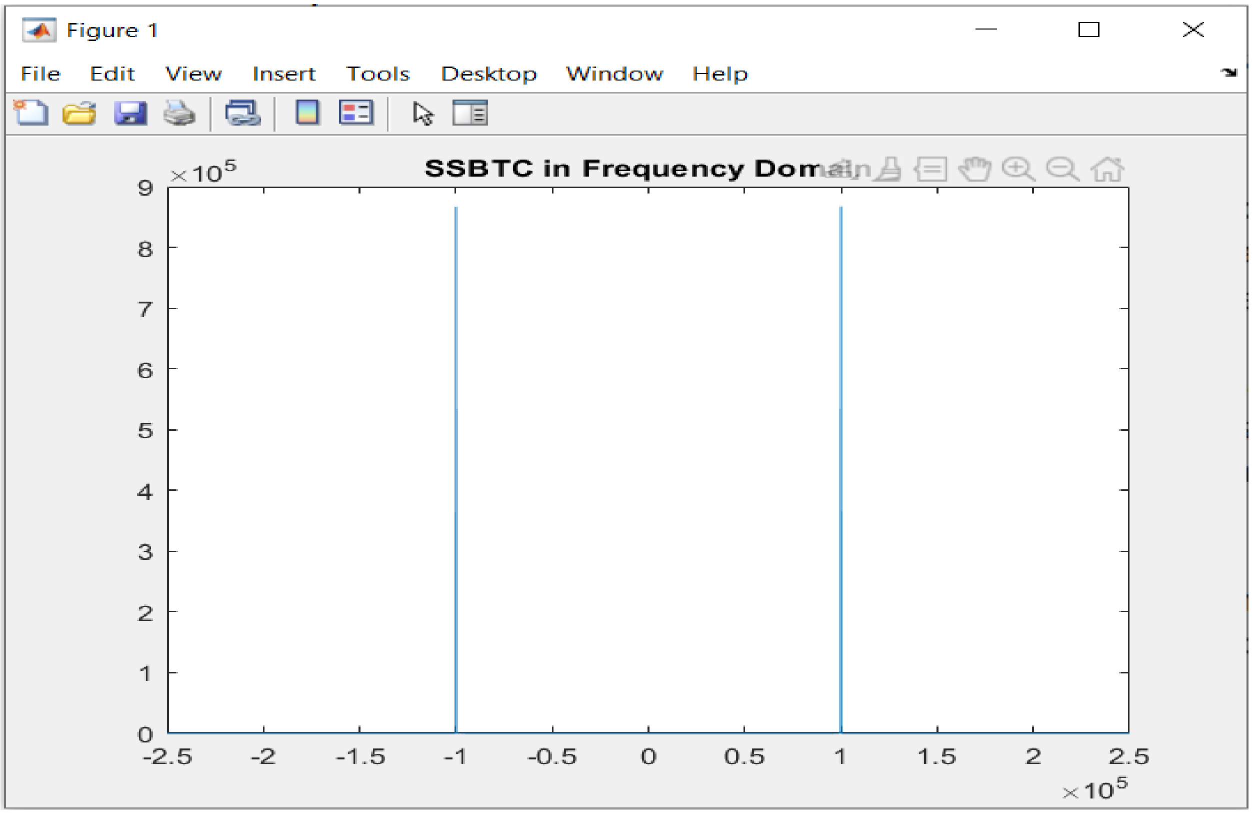


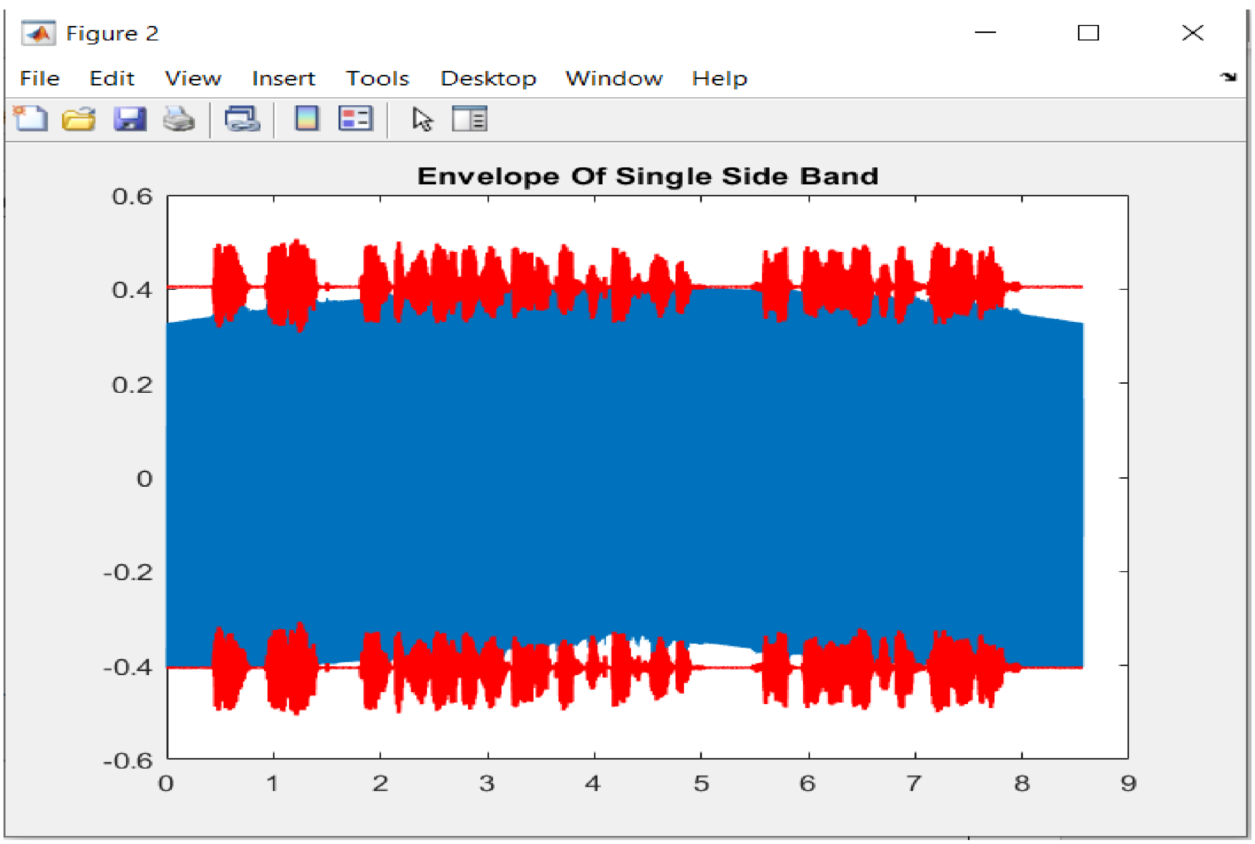
Single Sided Band Transmitted Carrier:

```
%% Single Side Band Transmitted Carrier
SSBTC = carrier + SSB_LSB;
len = length(SSBTC);
S_freq = fftshift(fft(SSBTC));
freq = f_S/2 * linspace(-1, 1, len);

figure;
plot(freq, abs(S_freq)    );
title('SSBTC in Frequency Domain');
envelopeDetection(SSBTC, timeVector, f_S);
```

```
function envelopeDetection(signal, time, f_Sampling)
    % envelopeDetection computes and plots the envelope of a signal
    % Inputs:
    %   signal: Input signal for envelope detection
    %   time: Time vector corresponding to the signal
    %   f_Sampling: Sampling frequency in Hz

    % Compute the envelope of the signal using Hilbert transform
    signalEnvelope = abs(hilbert(signal));

    % Plot the original signal and its envelope
    figure;
    plot(time, signal); % Plot the original signal
    hold on;
    plot(time, -signalEnvelope, '-r', time, signalEnvelope, '-r', 'Linewidth', 1.5); % Plot the envelope
    hold off;
    title('Envelope Of Single Side Band'); % Set the title for the plot

    % Resample the signal envelope and play the audio
    signalEnvelope = resample(abs(signalEnvelope), f_Sampling / 5, f_Sampling);
    sound(abs(signalEnvelope), f_Sampling / 5); % Play the resampled signal envelope
end
```

<u>Demodulated signal using Envelop detection:</u>

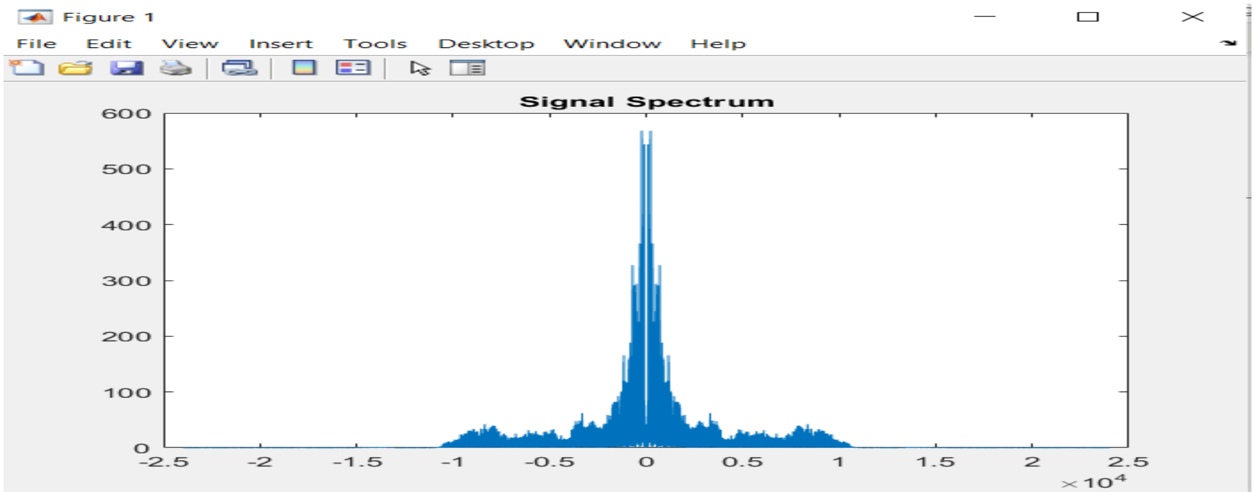




Experiment Three: Frequency Modulation: -

```
%% Reading Audio File
[signal, f S, S freq, freq] = start('eric.wav');
```

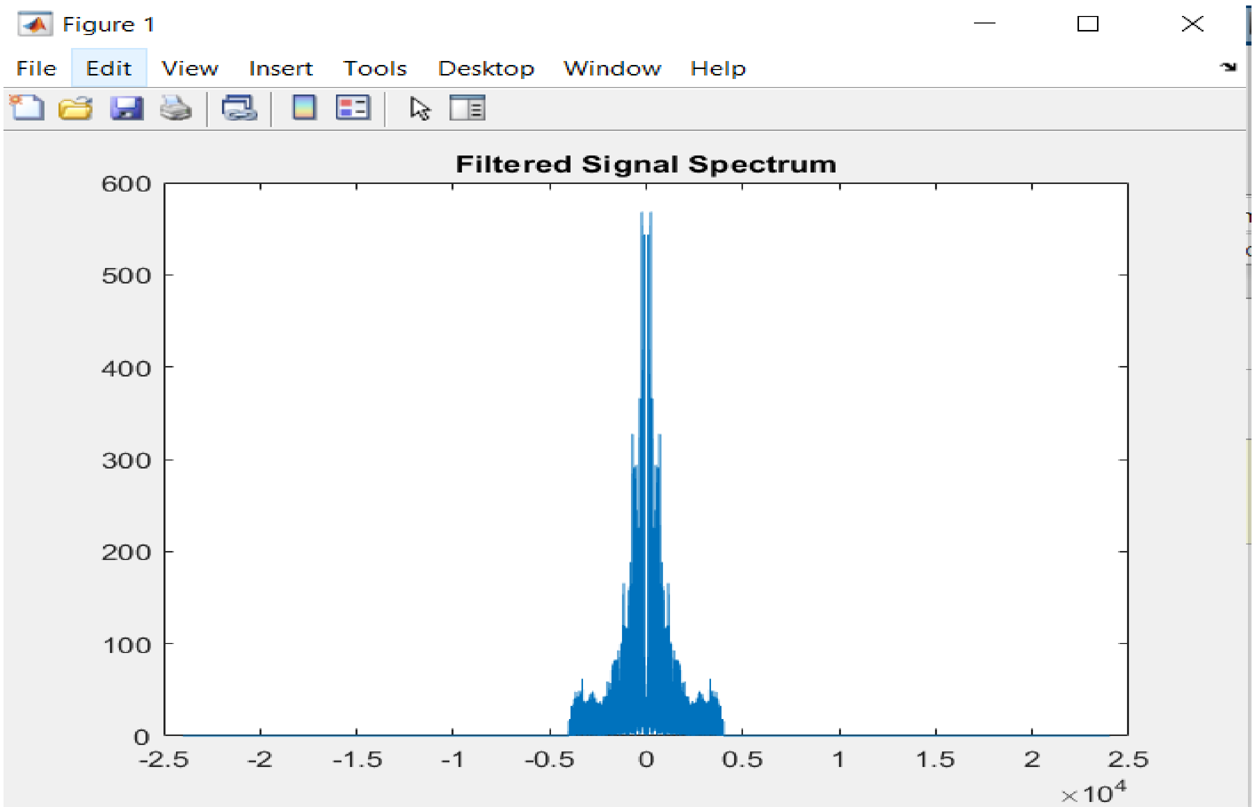
Signal in frequency domain:



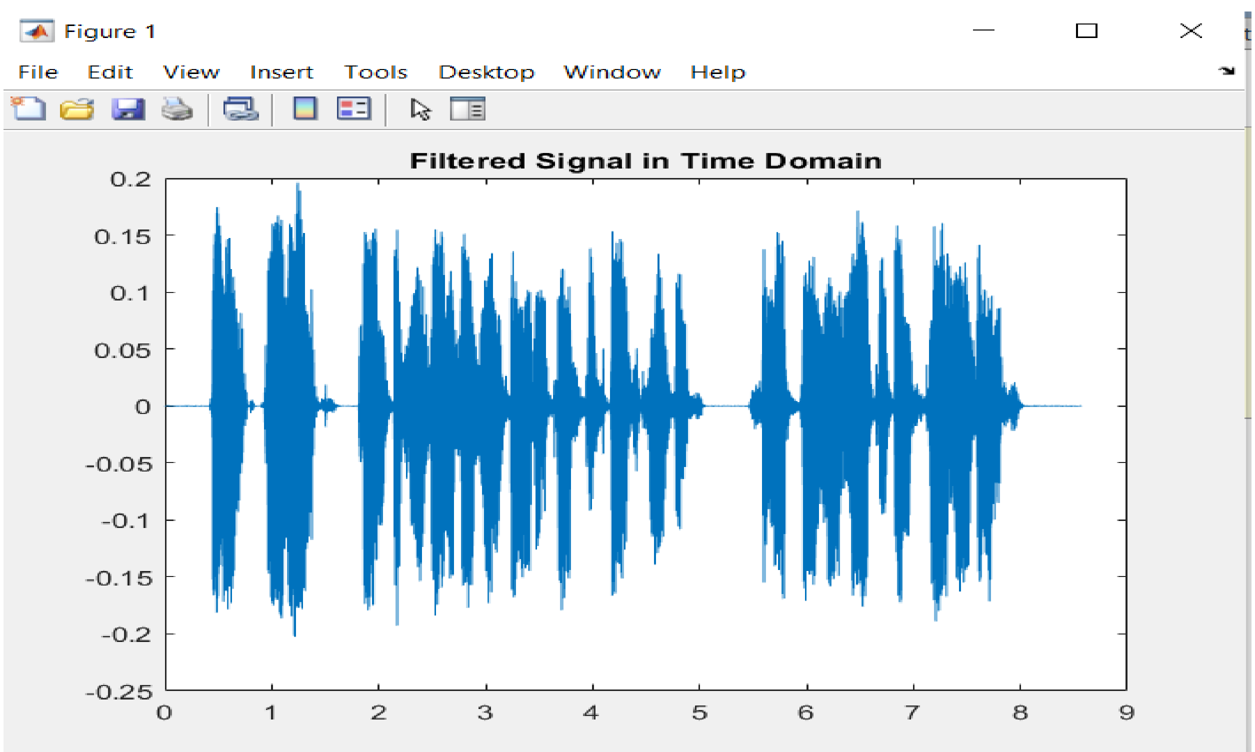
Filtered Signal:

```
%% Filtering the Input Signal
cutoffFreq = 4000;
filteredSignal = filtering(cutoffFreq, S_freq, freq, f_S);

startTime = 0;
endTime = startTime + length(filteredSignal) / f_S;
timeVector = linspace(startTime, endTime, length(filteredSignal));
timeVector = timeVector';

figure;
plot(timeVector, filteredSignal);
title('Filtered Signal Time Domain');

sound(abs(filteredSignal), f S);
```

```
function filteredSignal = filtering(cutoffFreq, S_freq, freq, f_Sampling)
    % filtering filters the input frequency-domain signal S_freq based on the cutoff frequency
    % Inputs:
    %   cutoffFreq: Frequency cutoff for filtering in Hz
    %   S_freq: Frequency domain signal
    %   freq: Frequency vector corresponding to the signal
    %   f_Sampling: Sampling frequency in Hz

    % Apply frequency domain filtering by zeroing out frequencies outside cutoff
    S_freq(freq >= cutoffFreq | freq <= -cutoffFreq) = 0;

    % Retrieve the filtered signal in time domain using inverse FFT
    filteredSignal = ifft(ifftshift(S_freq));

    % Compute the length of the filtered signal
    len = length(filteredSignal);

    % Compute the frequency spectrum of the filtered signal
    S_freq = fftshift(fft(filteredSignal));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the frequency spectrum of the filtered signal
    figure;
    plot(freq, abs(S_freq)      );
    title('Filtered Signal Spectrum'); % Set the title for the plot
end
```



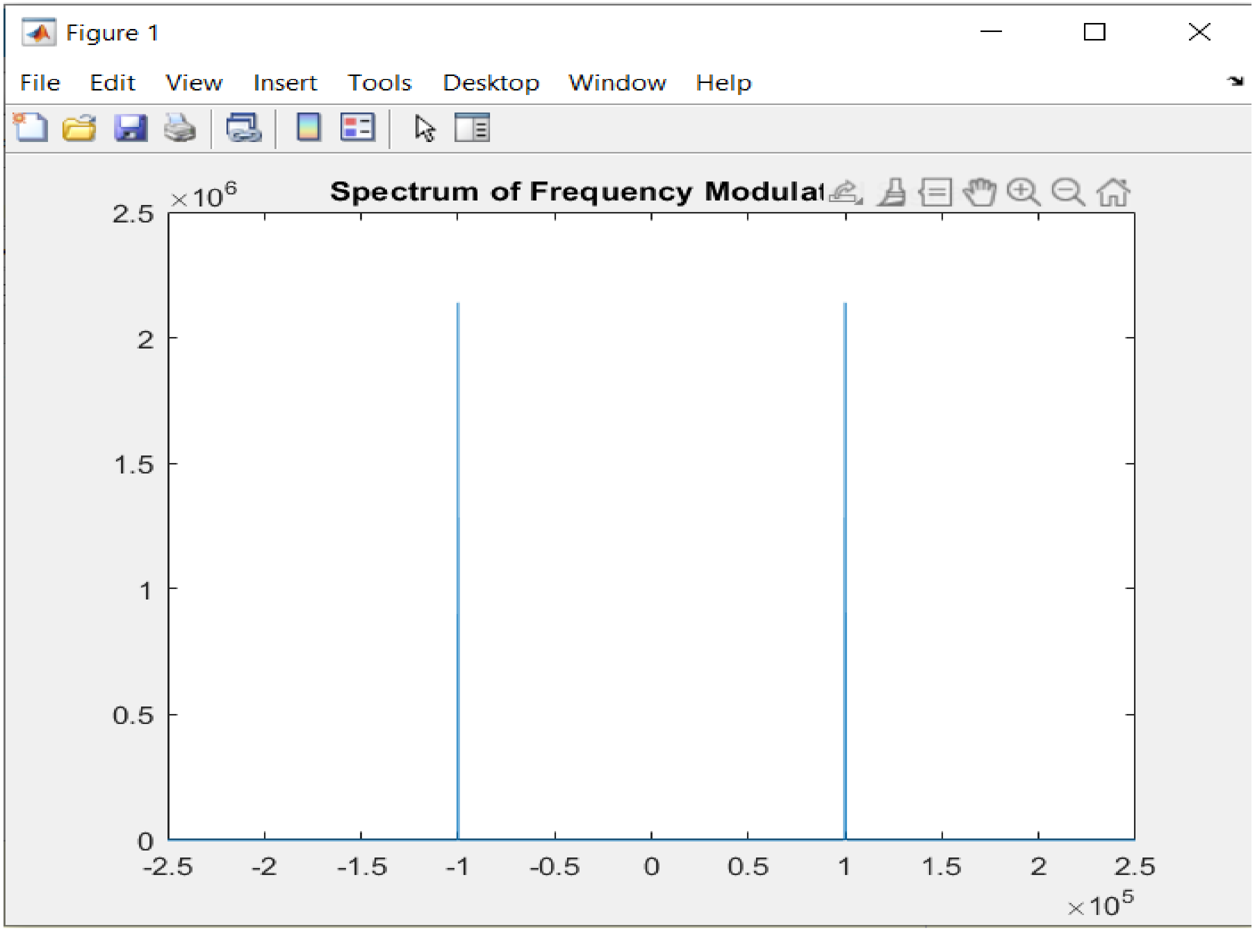
Filtered Signal in Time Domain:



Modulated NBFM signal in Frequency Domain:

```
%% Modulating the Signal
kf = 0.2/(2*pi*max(abs(cumsum(filteredSignal)))./ f_S);
carrierFreq = 100000;
carrierAmp = 1;
filteredSignal = resample(filteredSignal, 5 * carrierFreq, f_S);
f_S = 5 * carrierFreq;
[modSignal, timeVector] = frequencyModulation(kf, carrierFreq, carrierAmp, filteredSignal, f_S);
```

```
function [modSignal, timeVector] = frequencyModulation(kf, carrierFreq, carrierAmp, signal, f_Sampling)
    % frequencyModulation performs frequency modulation on an input signal
    % Inputs:
    %   kf: Frequency deviation constant (modulation index)
    %   carrierFreq: Carrier frequency of the modulation
    %   carrierAmp: Amplitude of the carrier signal
    %   signal: Input signal to be modulated
    %   f_Sampling: Sampling frequency of the signal

    % Generate the time vector corresponding to the signal
    startTime = 0;
    endTime = startTime + length(signal) / f_Sampling;
    timeVector = linspace(startTime, endTime, length(signal));
    timeVector = timeVector';

    % Perform Frequency Modulation (FM) on the input signal to generate the modulated signal
    modSignal = carrierAmp * cos(2 * pi * carrierFreq * timeVector + 2 * pi * kf * cumsum(signal) ./ f_Sampling);

    % Compute the Fourier transform of the input signal
    len = length(modSignal);
    S_Freq = fftshift(fft(modSignal));
    freq = f_Sampling / 2 * linspace(-1, 1, len);

    % Plot the spectrum of the frequency-modulated signal
    figure;
    plot(freq, S_Freq);
    title('Spectrum of Frequency Modulated Signal');
 d
```



-we notice that the resulting spectrum shape is almost like DSB-TC with bandwidth equal $2F_m$.

-According to Carson's rule:

$$B.W = 2F_m(\beta + 1)$$

So, the condition to achieve narrow band frequency modulation is to have a very small frequency deviation ratio (<1) so that its value can be ignored compared to 1 which makes the bandwidth equal double the bandwidth of the message which is like DSB and specially DBS-TC as the carrier is also transmitted.

$$S(t)_{NBFM} = A\cos(\omega_c t) - a\,K_f \int m(t)\,dt\ \sin(\omega_c t)$$

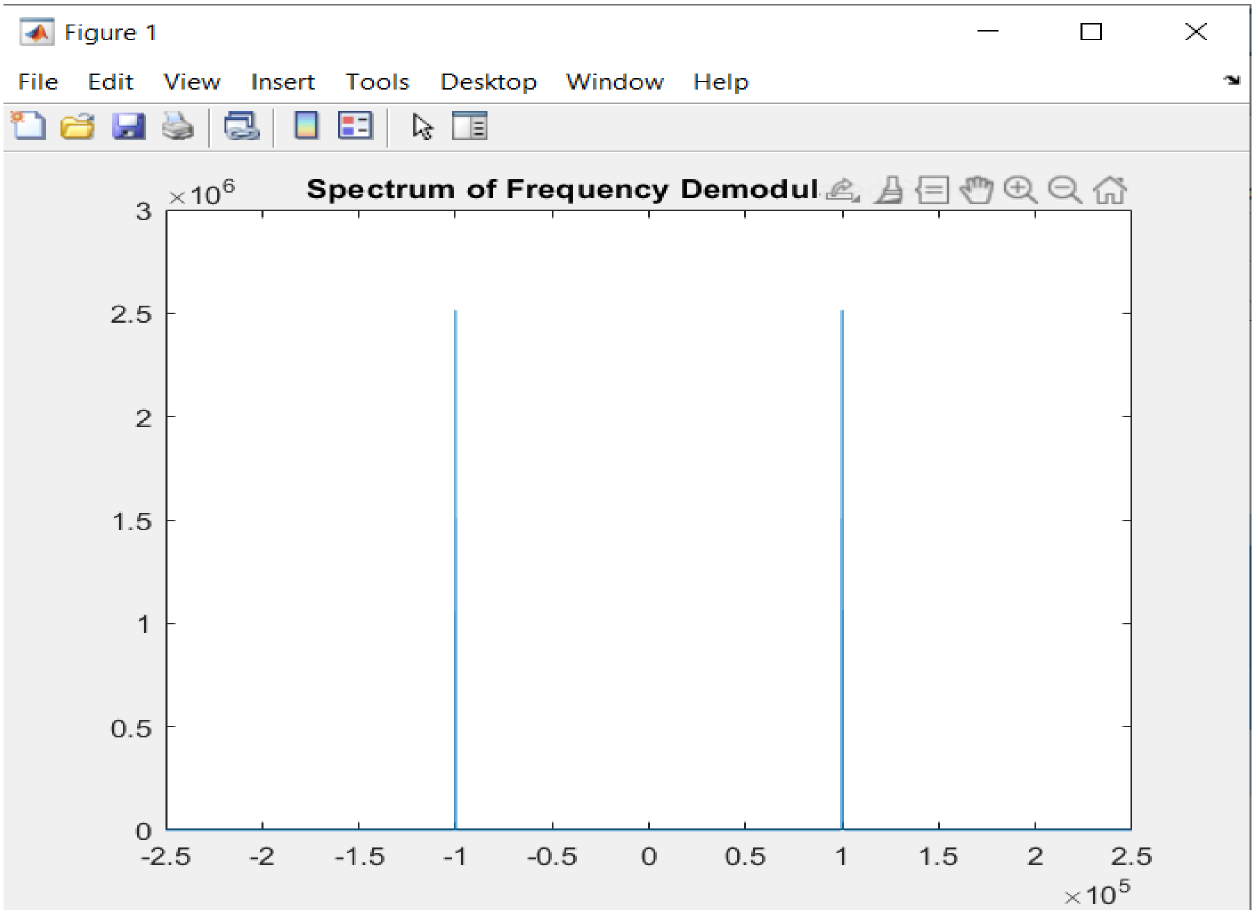
Demodulated NBFM signal in Frequency Domain:

```
%% Demodulating the Signal
frequencyDemodulation(modSignal, f_S, timeVector);
```

```
function frequencyDemodulation(modSignal, f_Sampling, timeVector)
    % frequencyDemodulation performs frequency demodulation on a modulated signal
    % Inputs:
    %   modSignal: Modulated signal to be demodulated
    %   f_Sampling: Sampling frequency of the signal
    %   timeVector: Time vector corresponding to the signal

    % Compute the derivative of the modulated signal to demodulate
    demodSignal = diff(modSignal);
    demodSignal = [0; demodSignal]; % Compensate for length difference due to diff function

    % Obtain the envelope of the demodulated signal using the Hilbert transform
    envelope = abs(hilbert(demodSignal)) - mean(abs(hilbert(demodSignal))); % Extract envelope

    % Plot the spectrum of the frequency-demodulated signal
    S_Freq = fftshift(fft(demodSignal));
    len = length(demodSignal);
    freq = f_Sampling / 2 * linspace(-1, 1, len);
    figure;
    plot(freq, abs(S_Freq));
    title('Spectrum of Frequency Demodulated Signal');

    % Plot the demodulated signal envelope over time
    figure;
    plot(timeVector, envelope);
    title("Demodulated Signal");
    ylim([-2*10^-4 2*10^-4]);

    % Resample the envelope and play the audio
    envelope = resample(envelope, f_Sampling / 5, f_Sampling);
    % Scale and play the envelope (500 is an arbitrary scaling factor for audio perception)
    sound(500 .* abs(envelope), f_Sampling/5);
end
```

Activate Wi

Demodulated NBFM signal in Frequency Domain:



Demodulated NBFM signal in Time Domain