Analog Communication

Rowan Ashraf	20011312
Retaj Tarek	20010606
Mohamed Nagy	20012409
Youssef Maged Elnady	20012286
Yousef Mohammed	20012301

Table of Contents

DSB.......... 2

Code Snipps.......... 2

Screenshots.......... 10

FM.......... 21

Code Snipps.......... 21

Screenshots.......... 25

SSB.......... 28

Code Snipps.......... 28

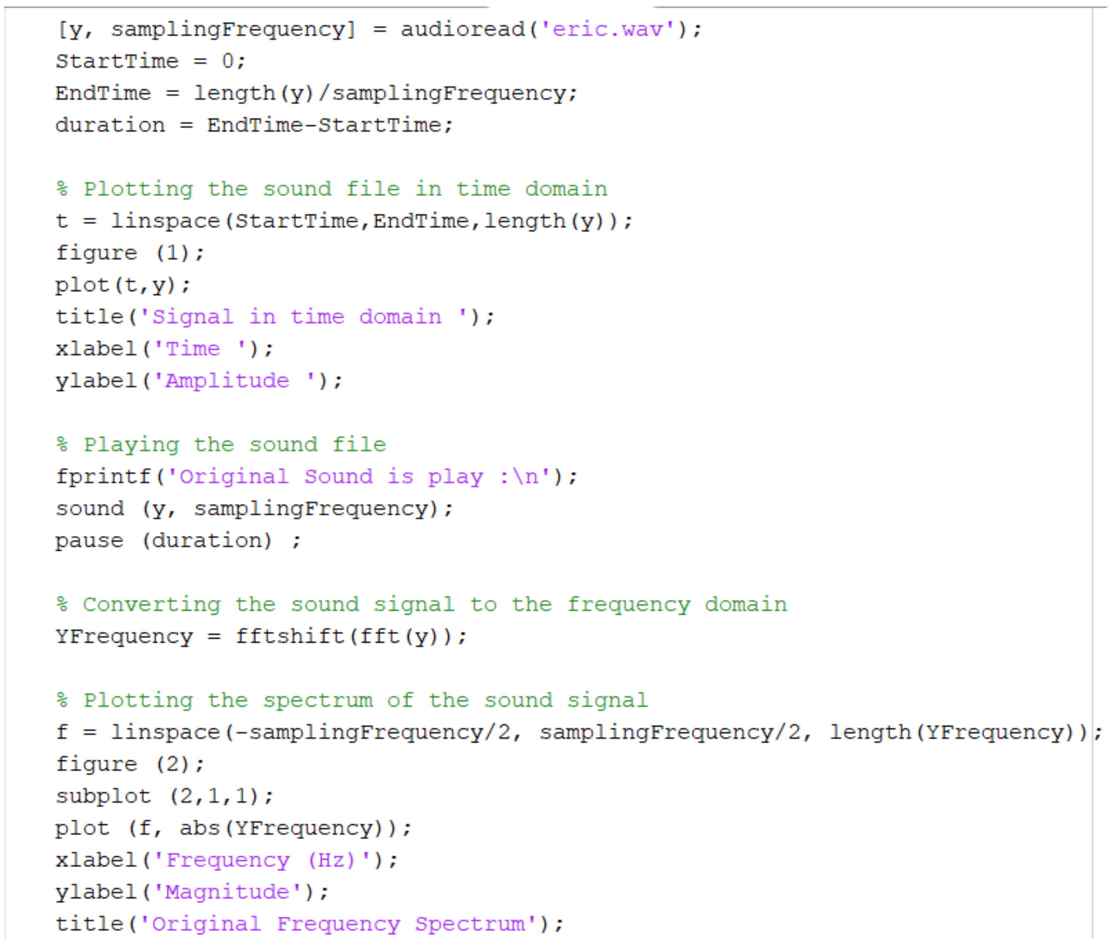
Screenshots.......... 34

DSB

This section explores the fundamentals and the significance of two essential analog modulation techniques: Double Sideband with Time Carrier (DSB-TC) and Double Sideband Suppressed Carrier (DSB-SC). Both techniques play pivotal roles in analog communication, addressing the need for bandwidth and power efficiency, whereas DSB-TC involves modulating a message signal with a carrier, transmitting both upper and lower sidebands, emphasizing its role in analog television broadcasting and certain AM radio transmissions while DSB-SC eliminates the carrier frequency, transmitting only sidebands. Both DSB-TC and DSB-SC contribute to improved bandwidth and power efficiency in analog communication.

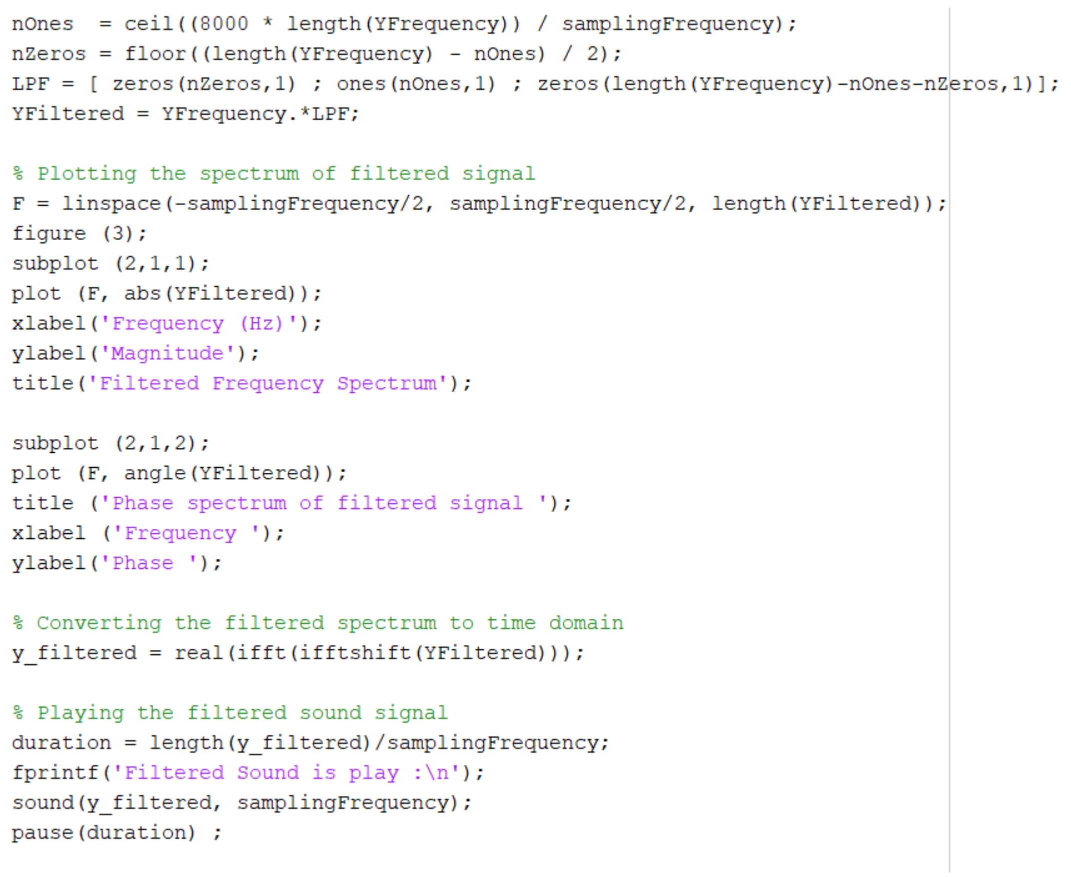
Code Snipps

- The initial part of the code deals with audio signal processing. It reads an audio file named 'eric.wav' using the audioread function and then plot it in time domain, We convert the time domain signal (Y) into the frequency domain using the Fast Fourier Transform (FFT). It then plots the magnitude of the frequency domain signal, fftshift shifts the zero-frequency component to the center of the spectrum, then computes the magnitude of the complex values, and then we create a frequency vector F corresponding to the frequency range of the FFT result.

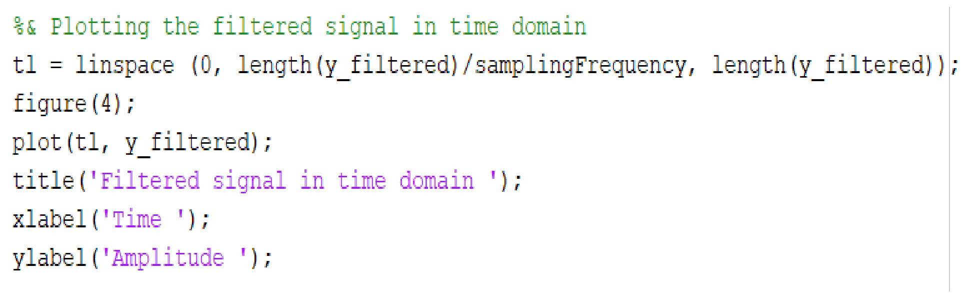


```
[y, samplingFrequency] = audioread('eric.wav');
StartTime = 0;
EndTime = length(y)/samplingFrequency;
duration = EndTime-StartTime;

% Plotting the sound file in time domain
t = linspace(StartTime,EndTime,length(y));
figure (1);
plot(t,y);
title('Signal in time domain ');
xlabel('Time ');
ylabel('Amplitude ');

% Playing the sound file
fprintf('Original Sound is play :\n');
sound (y, samplingFrequency);
pause (duration) ;

% Converting the sound signal to the frequency domain
YFrequency = fftshift(fft(y));

% Plotting the spectrum of the sound signal
f = linspace(-samplingFrequency/2, samplingFrequency/2, length(YFrequency));
figure (2);
subplot (2,1,1);
plot (f, abs(YFrequency));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Original Frequency Spectrum');
```

- The frequency spectrum is then filtered in which a low pass filter is applied to the signal to remove all frequencies above 4000Hz using a cutoff frequency , and the filtered signal is plotted in frequency domain and then converted back to time domain using ifft and ifftshift functions and then the filtered sound is played and the filtered signal is then plotted in time domain.

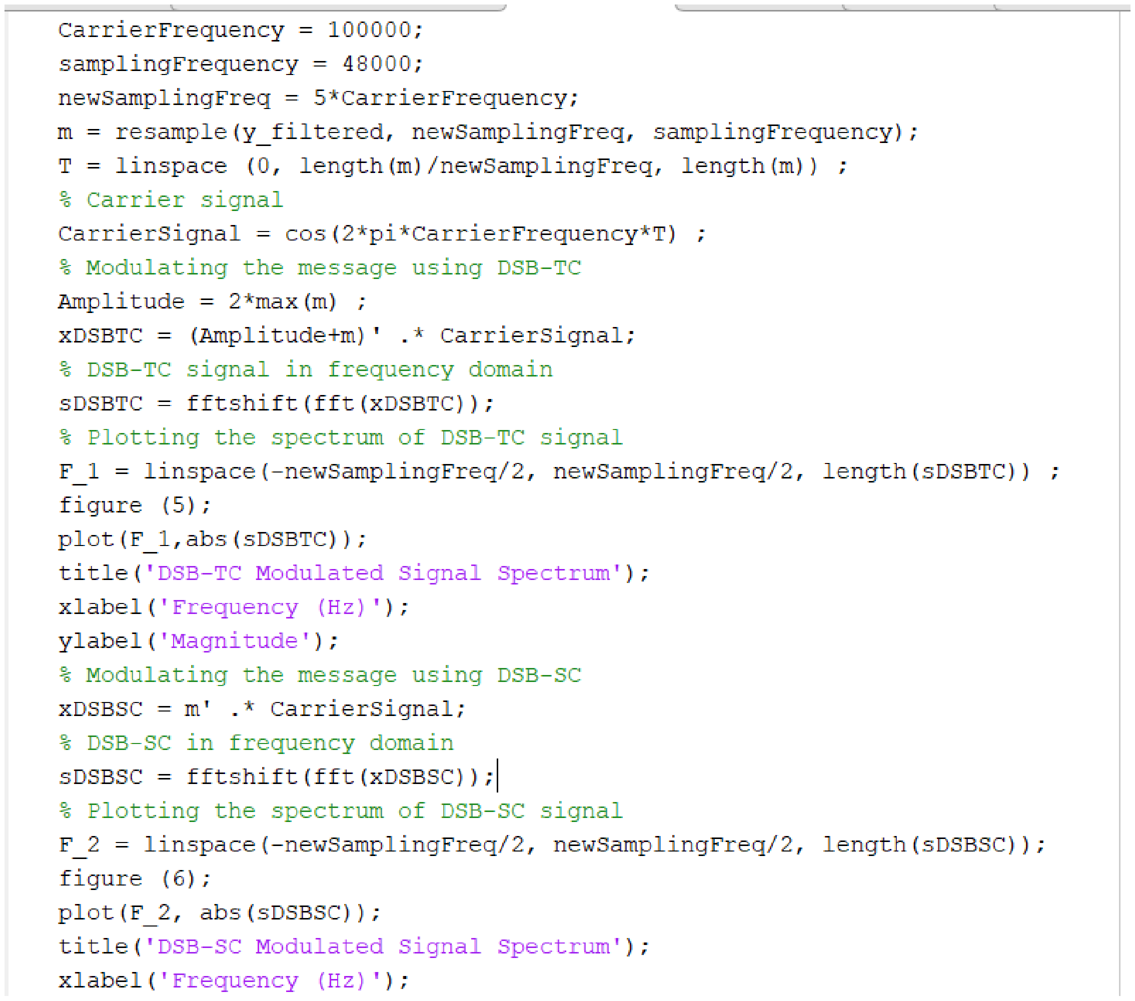


```
nOnes  = ceil((8000 * length(YFrequency)) / samplingFrequency);
nZeros = floor((length(YFrequency) - nOnes) / 2);
LPF = [ zeros(nZeros,1) ; ones(nOnes,1) ; zeros(length(YFrequency)-nOnes-nZeros,1)];
YFiltered = YFrequency.*LPF;

% Plotting the spectrum of filtered signal
F = linspace(-samplingFrequency/2, samplingFrequency/2, length(YFiltered));
figure (3);
subplot (2,1,1);
plot (F, abs(YFiltered));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Filtered Frequency Spectrum');

subplot (2,1,2);
plot (F, angle(YFiltered));
title ('Phase spectrum of filtered signal ');
xlabel ('Frequency ');
ylabel('Phase ');

% Converting the filtered spectrum to time domain
y_filtered = real(ifft(ifftshift(YFiltered)));

% Playing the filtered sound signal
duration = length(y_filtered)/samplingFrequency;
fprintf('Filtered Sound is play :\n');
sound(y_filtered, samplingFrequency);
pause(duration) ;
```

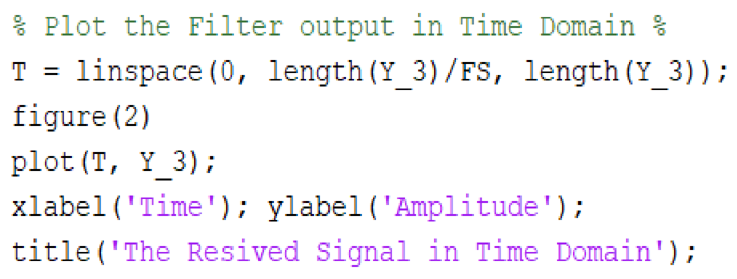


```
%& Plotting the filtered signal in time domain
tl = linspace (0, length(y_filtered)/samplingFrequency, length(y_filtered));
figure(4);
plot(tl, y_filtered);
title('Filtered signal in time domain ');
xlabel('Time ');
ylabel('Amplitude ');
```

- Next, the filtered signal is resampled to a new sampling frequency The resample function is used for this purpose. Two modulation techniques are then applied to the resampled signal: Double-Sideband Suppressed Carrier (DSB-SC) modulation and Double-Sideband with Carrier (DSB-TC) modulation. The modulated signals are obtained by multiplying the resampled signal with cosine functions at a carrier frequency of 100,000 Hz, in which in DSB_TC double the magnitude is added to the carrier The frequency spectra of the modulated signals are plotted using the FFT. The abs function is applied to the FFT result to obtain the magnitude spectrum. Two plots are created to display the spectra of the DSB-SC and DSB-TC modulated signals separately.



```
CarrierFrequency = 100000;
samplingFrequency = 48000;
newSamplingFreq = 5*CarrierFrequency;
m = resample(y_filtered, newSamplingFreq, samplingFrequency);
T = linspace (0, length(m)/newSamplingFreq, length(m)) ;
% Carrier signal
CarrierSignal = cos(2*pi*CarrierFrequency*T) ;
% Modulating the message using DSB-TC
Amplitude = 2*max(m) ;
xDSBTC = (Amplitude+m)' .* CarrierSignal;
% DSB-TC signal in frequency domain
sDSBTC = fftshift(fft(xDSBTC));
% Plotting the spectrum of DSB-TC signal
F_1 = linspace(-newSamplingFreq/2, newSamplingFreq/2, length(sDSBTC)) ;
figure (5);
plot(F_1,abs(sDSBTC));
title('DSB-TC Modulated Signal Spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
% Modulating the message using DSB-SC
xDSBSC = m' .* CarrierSignal;
% DSB-SC in frequency domain
sDSBSC = fftshift(fft(xDSBSC));
% Plotting the spectrum of DSB-SC signal
F_2 = linspace(-newSamplingFreq/2, newSamplingFreq/2, length(sDSBSC));
figure (6);
plot(F_2, abs(sDSBSC));
title('DSB-SC Modulated Signal Spectrum');
xlabel('Frequency (Hz)');
```

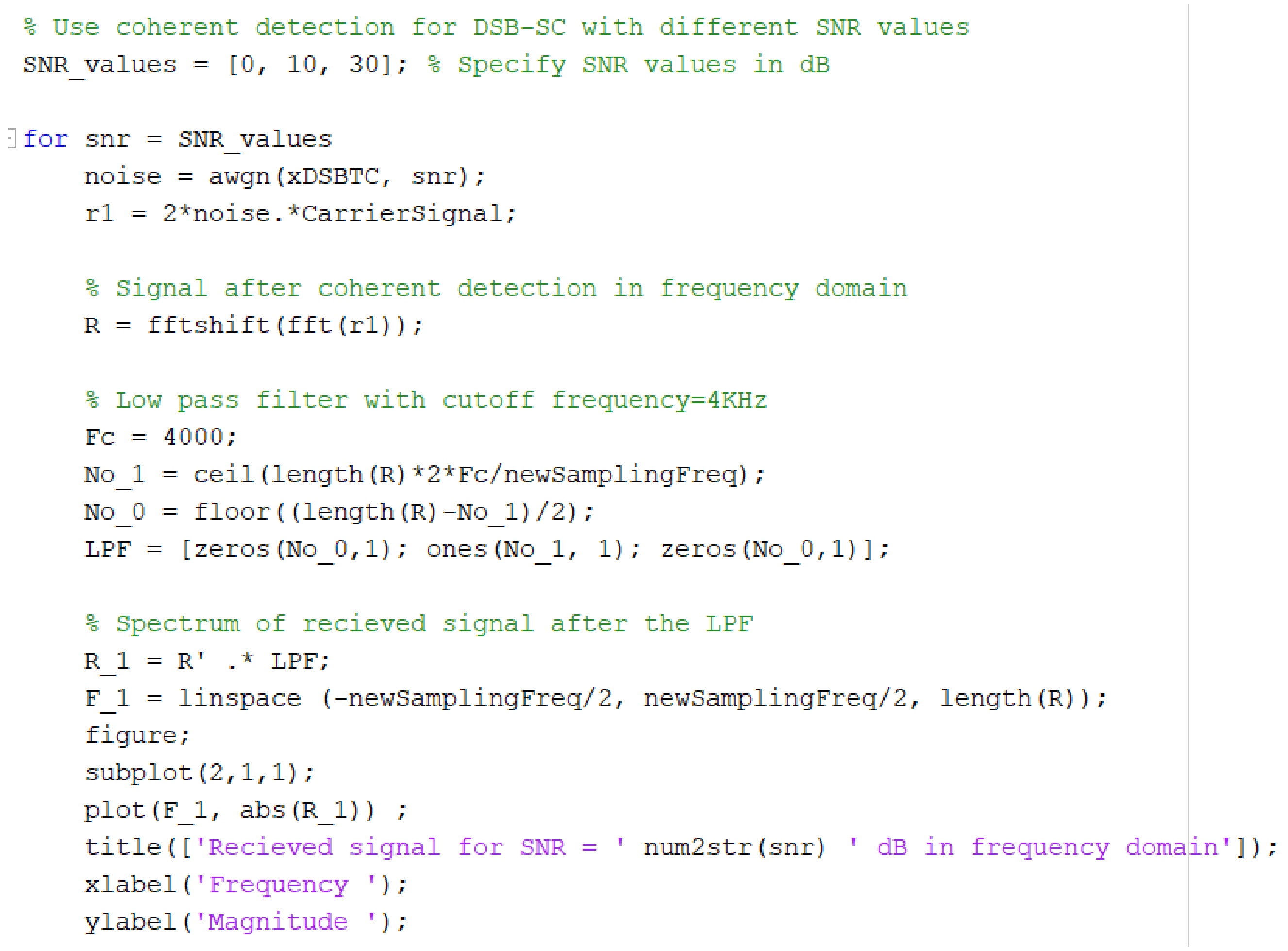


```
% Plot the Filter output in Time Domain %
T = linspace(0, length(Y_3)/FS, length(Y_3));
figure(2)
plot(T, Y_3);
xlabel('Time'); ylabel('Amplitude');
title('The Resived Signal in Time Domain');
```

- This part of the code focuses on envelope detection. The hilbert function is used to compute the analytic signal of the modulated signals. The absolute value (abs) of the resulting analytic signal represents the envelope of the modulated signals and then We plot the Envelope of DSB-TC and DSB-SC for both signals, the sampling frequency is down-sampled back to its initial value and both sounds of envelope of DSB SC and envelope of DSB TC are played.

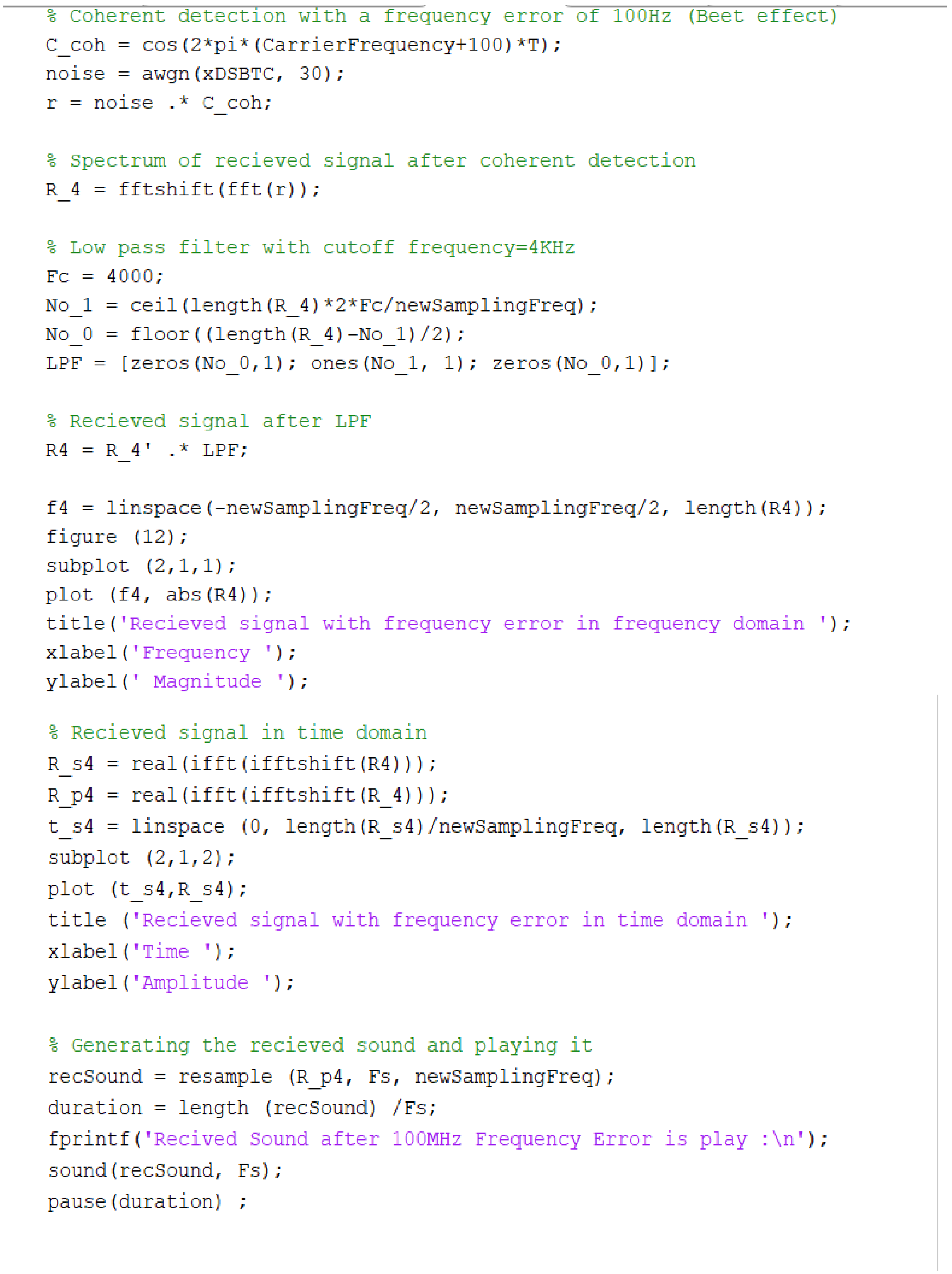
```
% Detection of DSB-TC signal using envelope detector
envelopDSBTC = abs(hilbert(xDSBTC));
% Plotting the detected envelope in time domain
figure (7);
plot(T,envelopDSBTC) ;
title('Recieved signal after DSB-TC modulation ');
xlabel('Time' );
ylabel('Amplitude ');
% Decreasing the sampling frequency
Fs=0.2*newSamplingFreq;
% Generating the recieved sound and playing it
recSound = resample(envelopDSBTC, Fs, newSamplingFreq) ;
duration = length(recSound)/Fs;
fprintf('Recived Sound after DSB-TC Envelope is play :\n');
sound(recSound, Fs);
pause(duration) ;
% Detection of DSB-SC signal using envelope detector
envelopDSBSC = abs(hilbert(xDSBSC));
% Plotting the recieved envelope in time domain
figure(8);
plot(T,envelopDSBSC) ;
title('Recieved signal after DSB-SC modulation ');
xlabel('Time ');
ylabel('Amplitude ');
% Generating the recieved sound and playing it
recSound_2 = resample (envelopDSBSC,Fs,newSamplingFreq) ;
duration = length(recSound_2) /Fs;
fprintf('Recived Sound after DSB-SC Envelope is play :\n');
sound(recSound_2, Fs);
```

- This block of code is part of the simulation for coherent detection of a DSB-SC (Double Sideband Suppressed Carrier) modulated signal with different Signal-to-Nois e Ratio (SNR) values. This loop iterates through a set of SNR values specified in the array SNR_values. For each SNR value, the following steps are performed. AWGN is added to the envelope signal (envelope_ DSB_SC) to simulate the effect of noise in a communication system. The snr parameter controls the Signal-to-Noise Ratio. and then the signal is converted to the frequency domain and a low pass filter of frequency 4KHz is applied and the received signal after the LPF is plotted in frequency domain and the signal is converted back to time domain using ifft and ifftshift functions and then plotted in time domain and the sound is played.



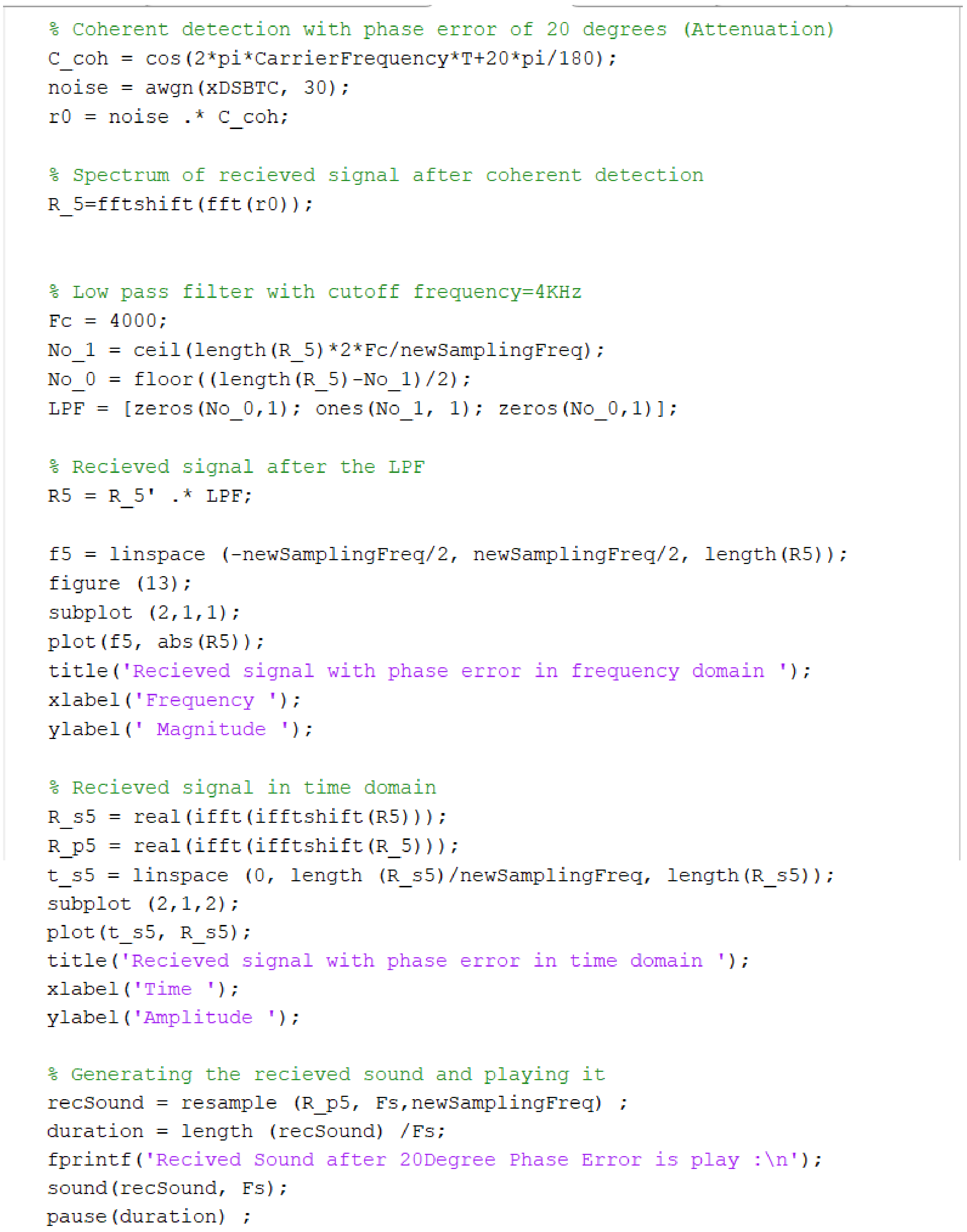
```
% Use coherent detection for DSB-SC with different SNR values
SNR_values = [0, 10, 30]; % Specify SNR values in dB

for snr = SNR_values
    noise = awgn(xDSBTC, snr);
    r1 = 2*noise.*CarrierSignal;

    % Signal after coherent detection in frequency domain
    R = fftshift(fft(r1));

    % Low pass filter with cutoff frequency=4KHz
    Fc = 4000;
    No_1 = ceil(length(R)*2*Fc/newSamplingFreq);
    No_0 = floor((length(R)-No_1)/2);
    LPF = [zeros(No_0,1); ones(No_1, 1); zeros(No_0,1)];

    % Spectrum of recieved signal after the LPF
    R_1 = R' .* LPF;
    F_1 = linspace (-newSamplingFreq/2, newSamplingFreq/2, length(R));
    figure;
    subplot(2,1,1);
    plot(F_1, abs(R_1)) ;
    title(['Recieved signal for SNR = ' num2str(snr) ' dB in frequency domain']);
    xlabel('Frequency ');
    ylabel('Magnitude ');
    % Recieved signal in time domain
    R_s1 = real(ifft(ifftshift(R_1)));
    R_p1 = real(ifft(ifftshift(R)));
    t_s1 = linspace (0, length(R_s1)/newSamplingFreq,length(R_s1));
    subplot(2,1,2);
    plot(t_s1, R_s1);
    title(['Recieved signal for SNR = ' num2str(snr) ' dB in time domain']);
    xlabel('Time ');
    ylabel('Amplitude ');

    % Generating the recieved sound and playing it
    recSound = resample(R_p1, Fs, newSamplingFreq);
    duration = length(recSound)/Fs;
    fprintf('Recived Sound after Noise Addition is play :\n');
    sound(recSound, Fs);
    pause(duration) ;
end
```

- Here We added a frequency error to the carrier and noise of SNR value=30 and a low pass filter with cutoff frequency=4KHz is then applied to the received signal and We plot the Signal in both frequency and time domain and then the sound is played. A frequency error causes a shift in the spectrum of the modulated sign.



```
% Coherent detection with a frequency error of 100Hz (Beet effect)
C_coh = cos(2*pi*(CarrierFrequency+100)*T);
noise = awgn(xDSBTC, 30);
r = noise .* C_coh;

% Spectrum of recieved signal after coherent detection
R_4 = fftshift(fft(r));

% Low pass filter with cutoff frequency=4KHz
Fc = 4000;
No_1 = ceil(length(R_4)*2*Fc/newSamplingFreq);
No_0 = floor((length(R_4)-No_1)/2);
LPF = [zeros(No_0,1); ones(No_1, 1); zeros(No_0,1)];

% Recieved signal after LPF
R4 = R_4' .* LPF;

f4 = linspace(-newSamplingFreq/2, newSamplingFreq/2, length(R4));
figure (12);
subplot (2,1,1);
plot (f4, abs(R4));
title('Recieved signal with frequency error in frequency domain ');
xlabel('Frequency ');
ylabel(' Magnitude ');

% Recieved signal in time domain
R_s4 = real(ifft(ifftshift(R4)));
R_p4 = real(ifft(ifftshift(R_4)));
t_s4 = linspace (0, length(R_s4)/newSamplingFreq, length(R_s4));
subplot (2,1,2);
plot (t_s4,R_s4);
title ('Recieved signal with frequency error in time domain ');
xlabel('Time ');
ylabel('Amplitude ');

% Generating the recieved sound and playing it
recSound = resample (R_p4, Fs, newSamplingFreq);
duration = length (recSound) /Fs;
fprintf('Recived Sound after 100MHz Frequency Error is play :\n');
sound(recSound, Fs);
pause(duration) ;
```

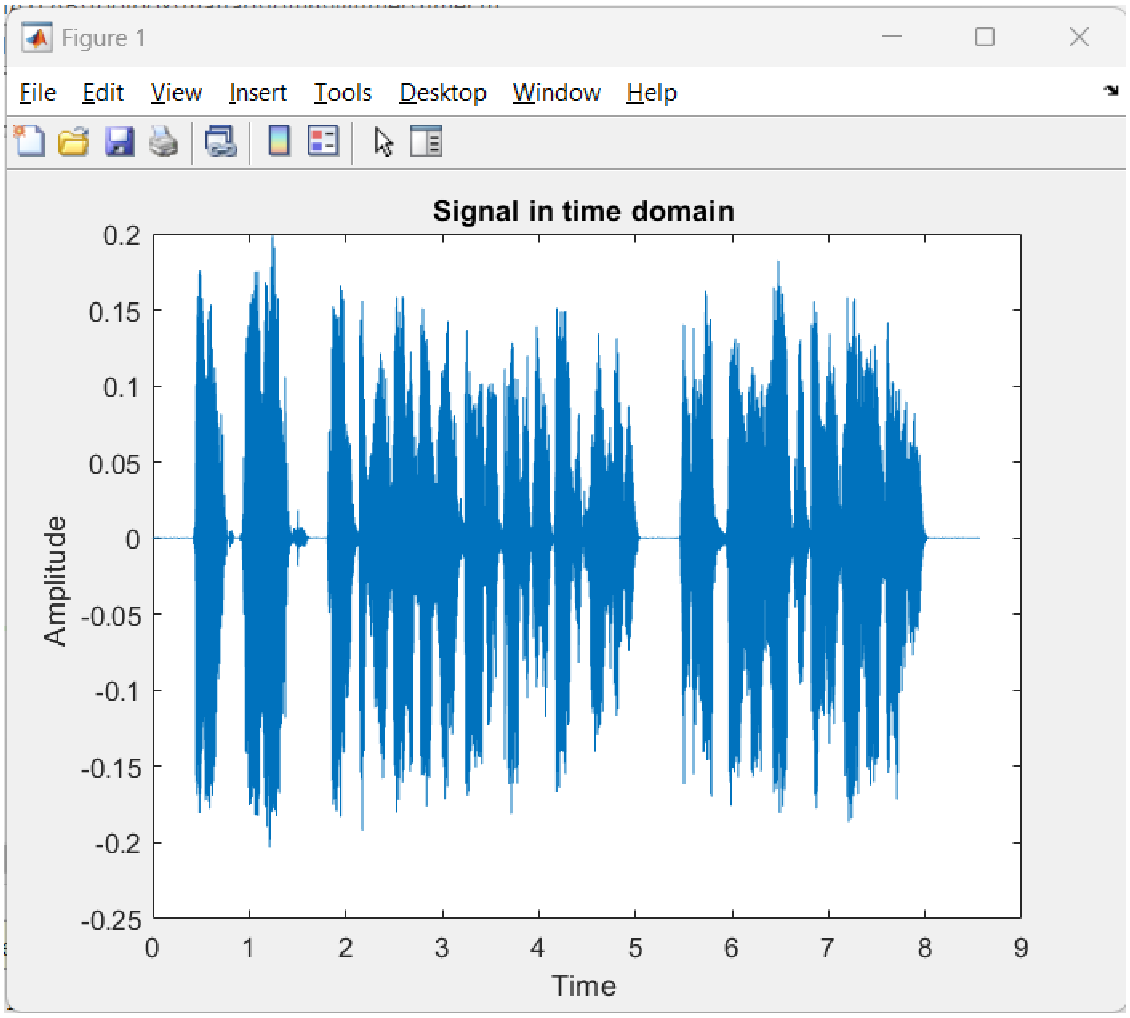
- In this part of the code We added a phase error to the signal and a noise with SNR value of 30 and then a low pass filter with cutoff frequency=4KHz is applied and plotted in both frequency and time domains Phase error can lead to a distortion of the modulated signal and then the sound is played.



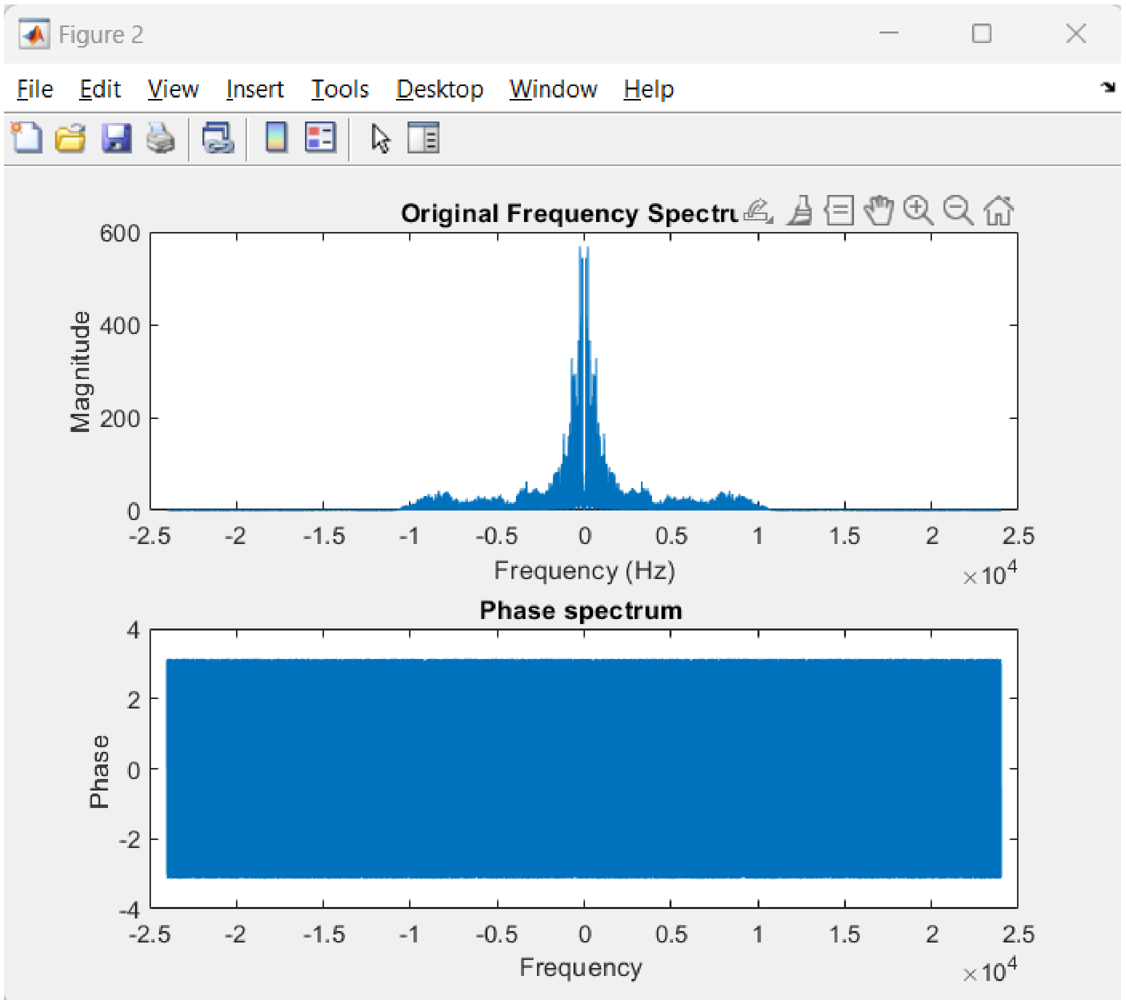
```
% Coherent detection with phase error of 20 degrees (Attenuation)
C_coh = cos(2*pi*CarrierFrequency*T+20*pi/180);
noise = awgn(xDSBTC, 30);
r0 = noise .* C_coh;

% Spectrum of recieved signal after coherent detection
R_5=fftshift(fft(r0));


% Low pass filter with cutoff frequency=4KHz
Fc = 4000;
No_1 = ceil(length(R_5)*2*Fc/newSamplingFreq);
No_0 = floor((length(R_5)-No_1)/2);
LPF = [zeros(No_0,1); ones(No_1, 1); zeros(No_0,1)];

% Recieved signal after the LPF
R5 = R_5' .* LPF;

f5 = linspace (-newSamplingFreq/2, newSamplingFreq/2, length(R5));
figure (13);
subplot (2,1,1);
plot(f5, abs(R5));
title('Recieved signal with phase error in frequency domain ');
xlabel('Frequency ');
ylabel(' Magnitude ');

% Recieved signal in time domain
R_s5 = real(ifft(ifftshift(R5)));
R_p5 = real(ifft(ifftshift(R_5)));
t_s5 = linspace (0, length (R_s5)/newSamplingFreq, length(R_s5));
subplot (2,1,2);
plot(t_s5, R_s5);
title('Recieved signal with phase error in time domain ');
xlabel('Time ');
ylabel('Amplitude ');

% Generating the recieved sound and playing it
recSound = resample (R_p5, Fs,newSamplingFreq) ;
duration = length (recSound) /Fs;
fprintf('Recived Sound after 20Degree Phase Error is play :\n');
sound(recSound, Fs);
pause(duration) ;
```

Screenshots

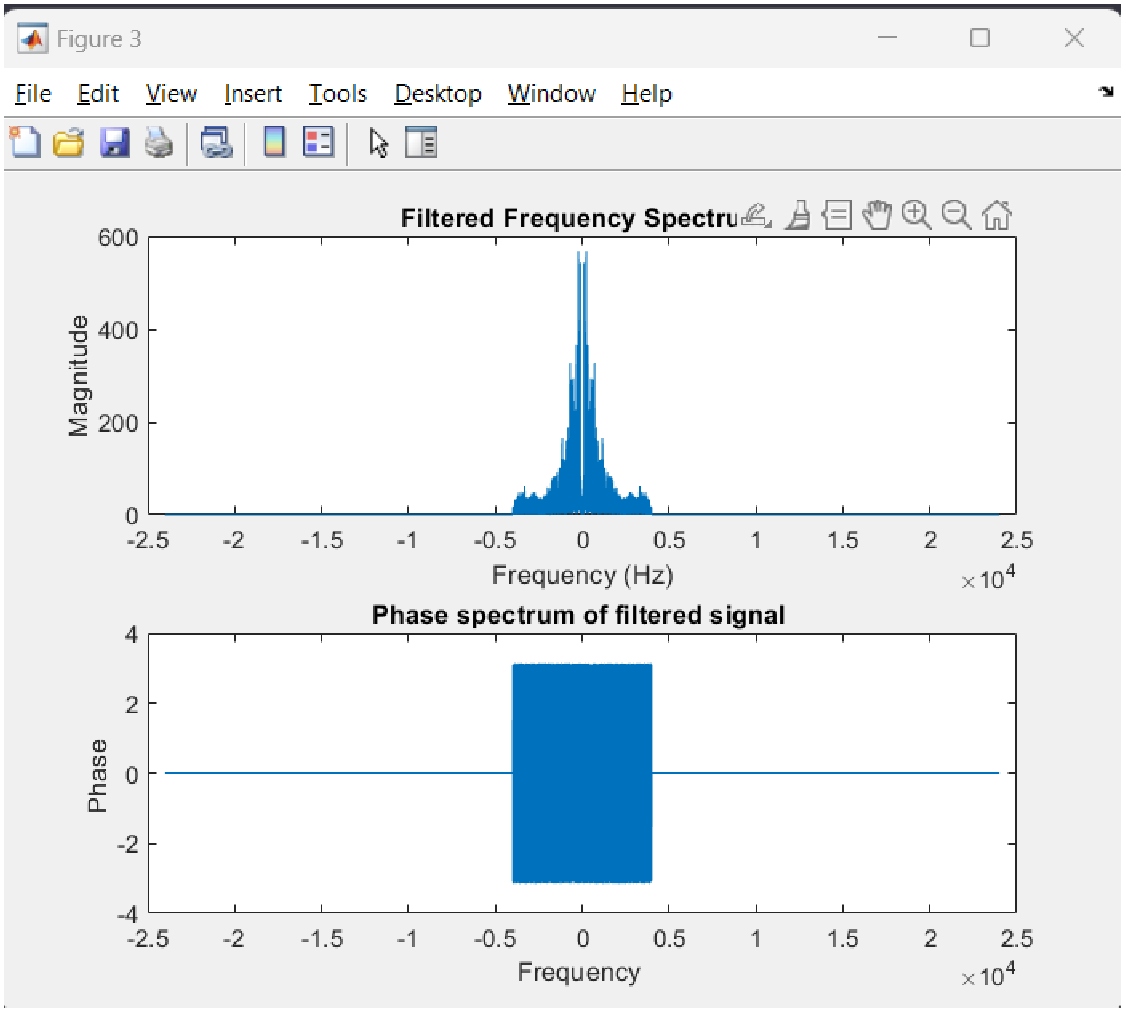
The original signal in time domain:



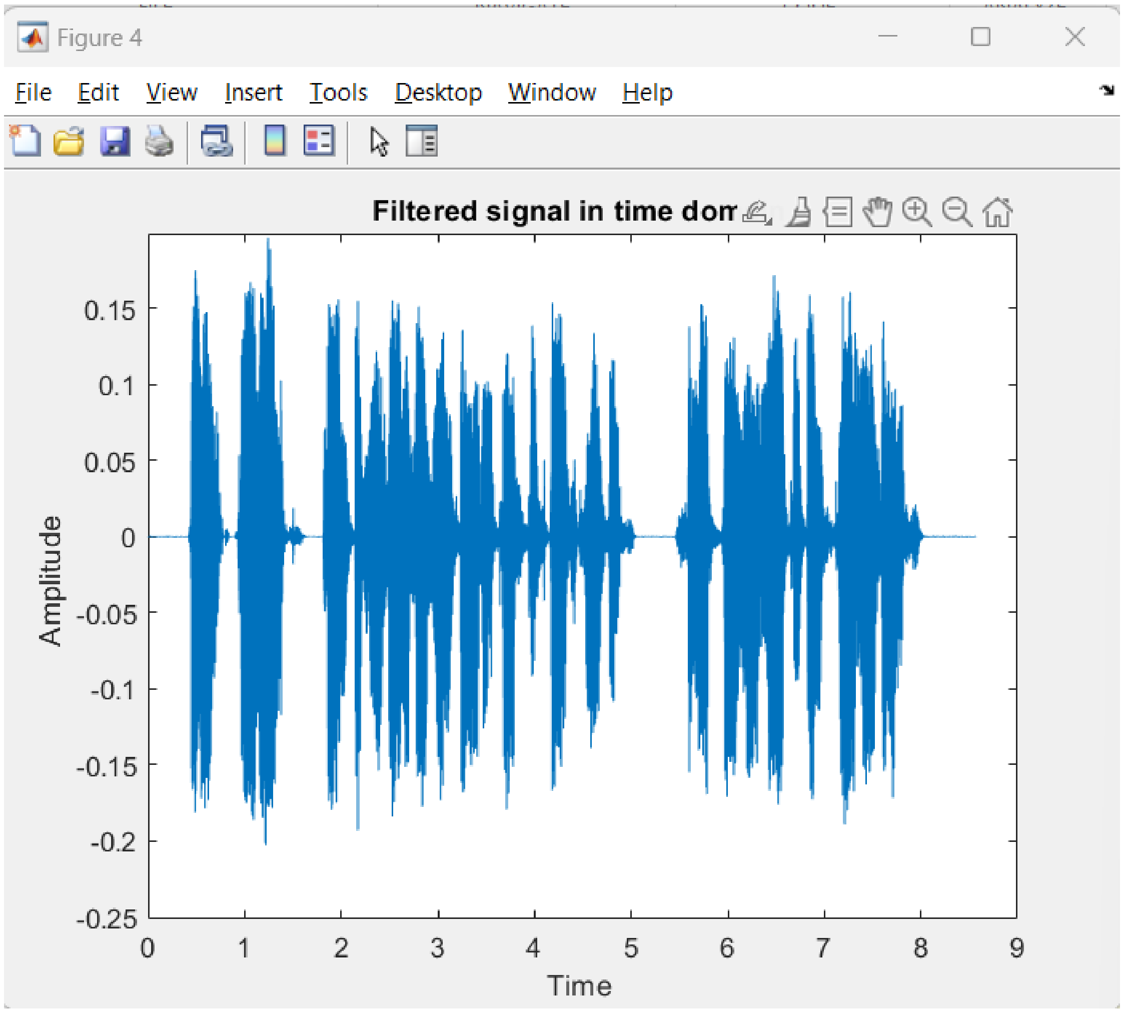
The original signal's presentation in frequency and phase spectrums:



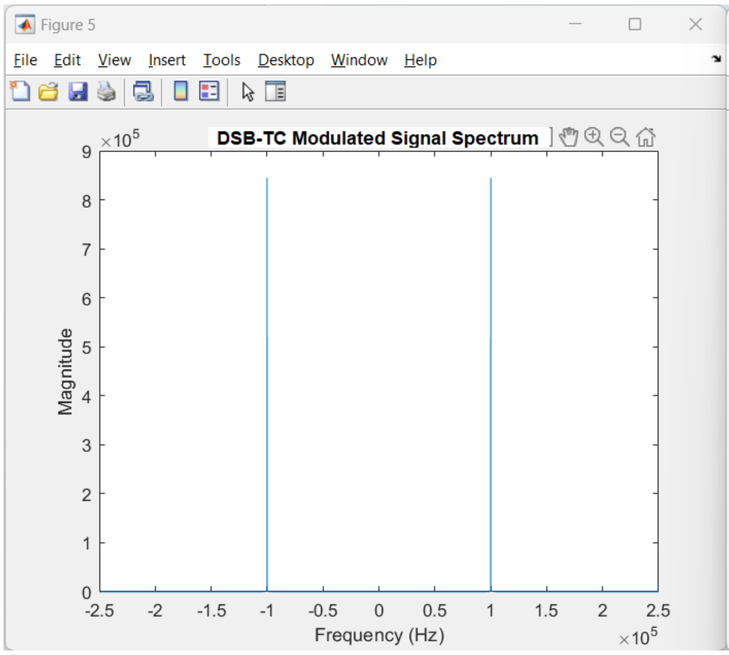
The filtered signal's presentation in frequency and phase spectrums:

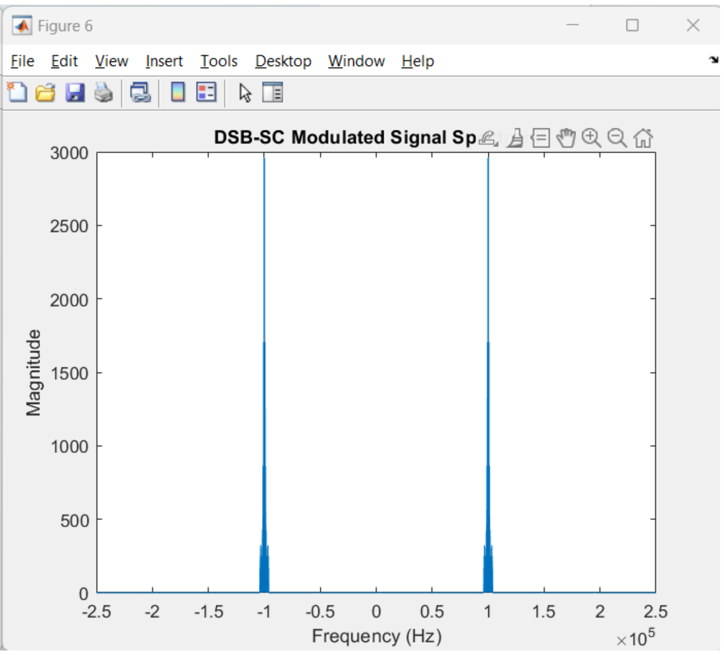


The filtered signal in time domain:

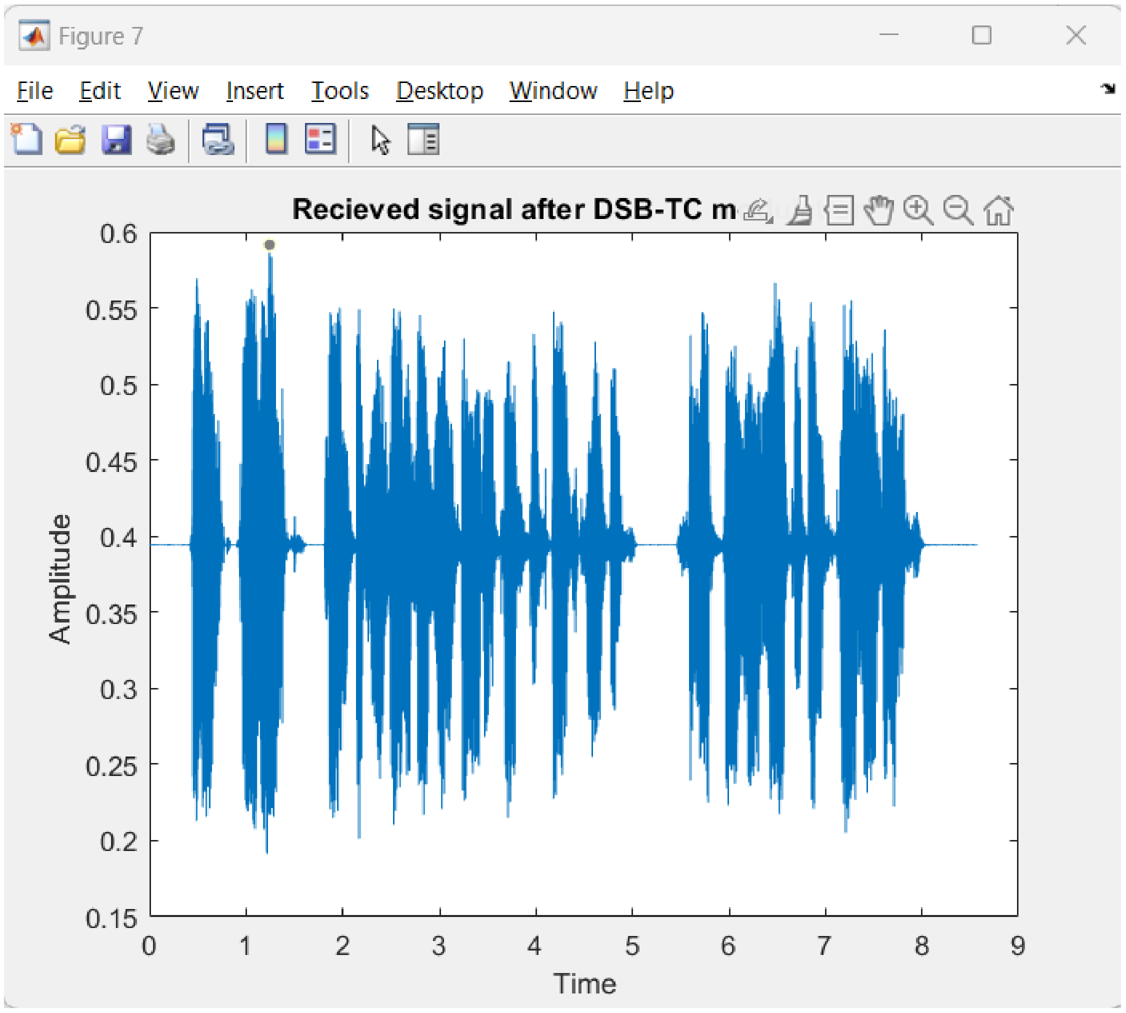


The Modulated Signal in Frequency Domain in Both DSB-SC and DSB-TC

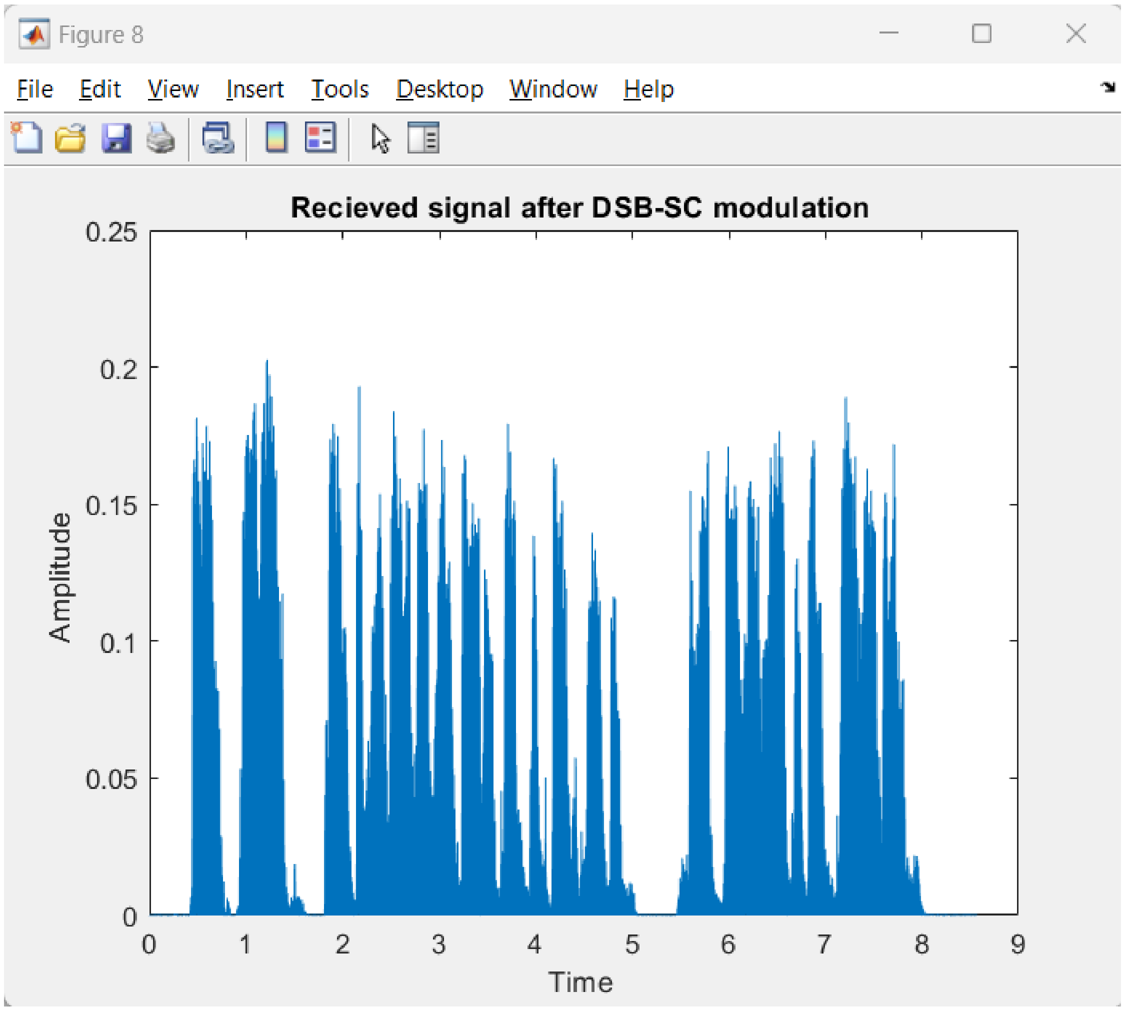




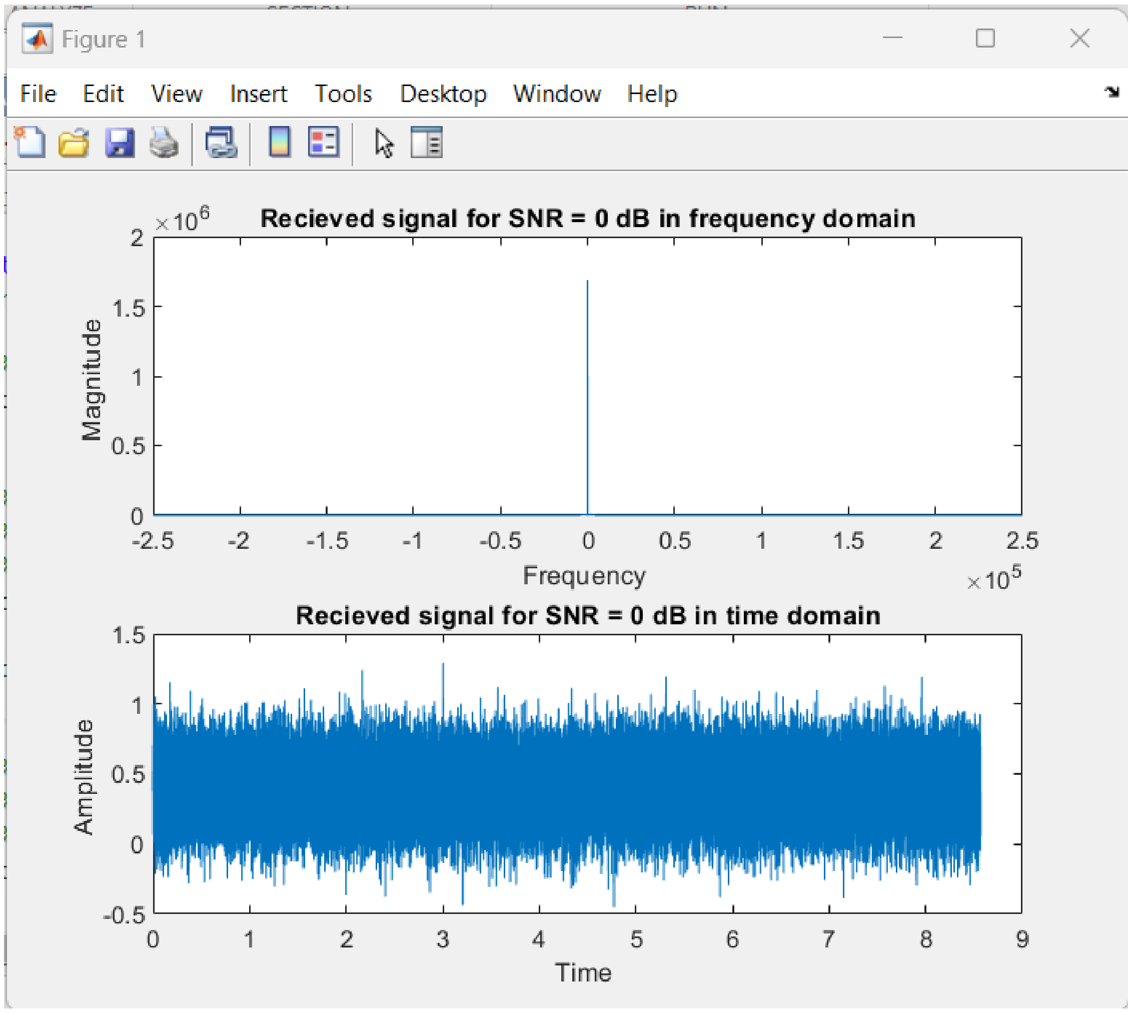
Signal received following DSB-TC modulation:



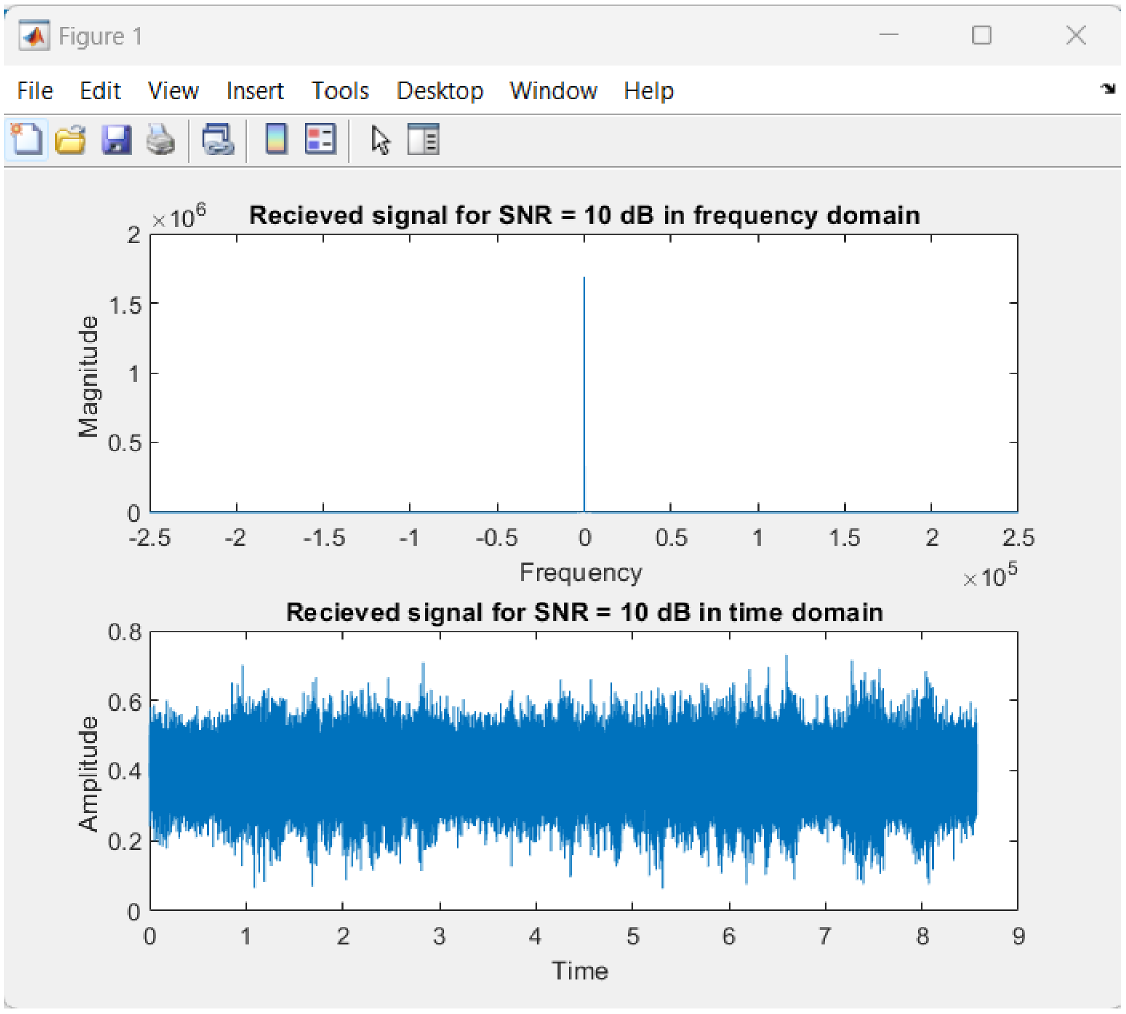
Signal received following DSB-SC modulation:



The signal received after noise addition SNR = 0:



The signal received after noise addition SNR = 10:



The signal received after noise addition SNR = 30:

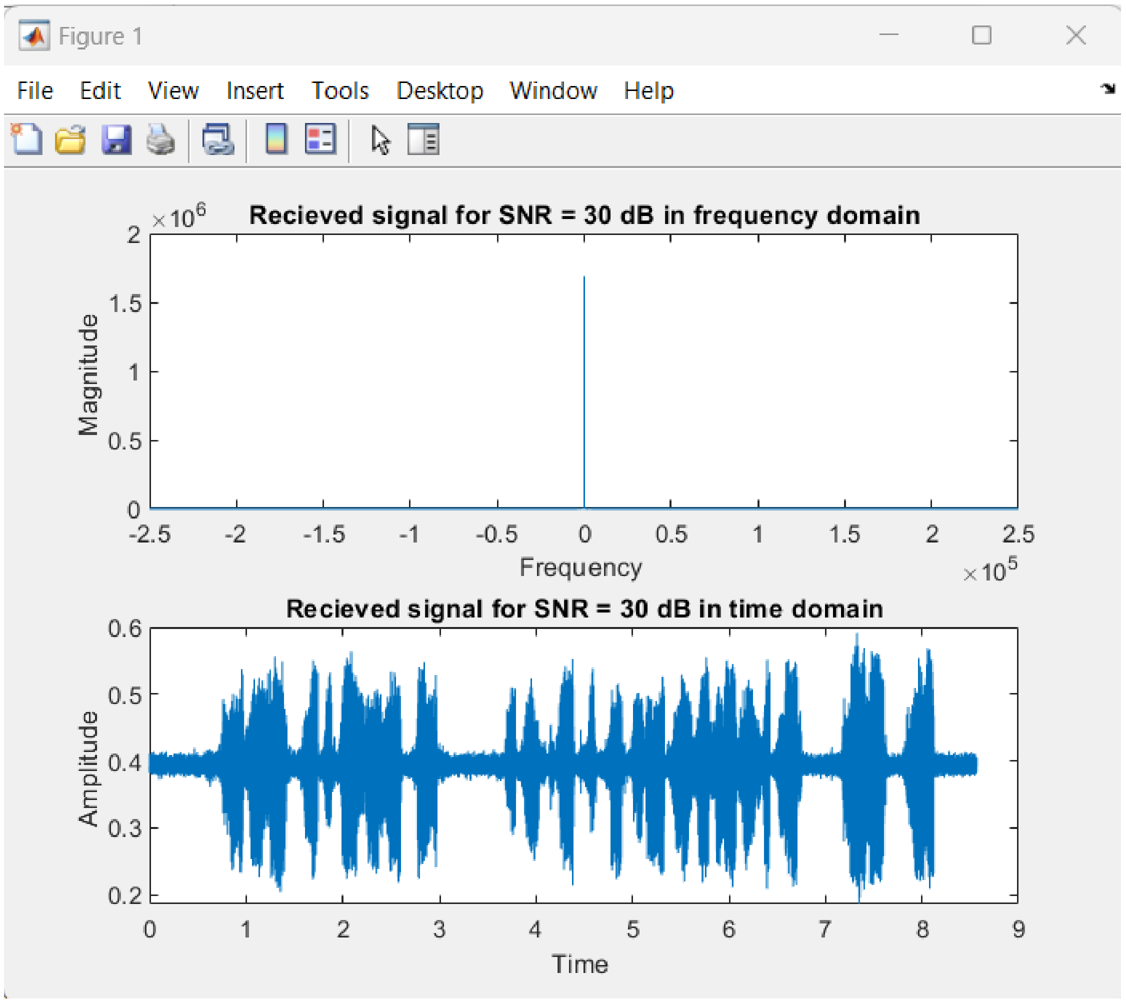
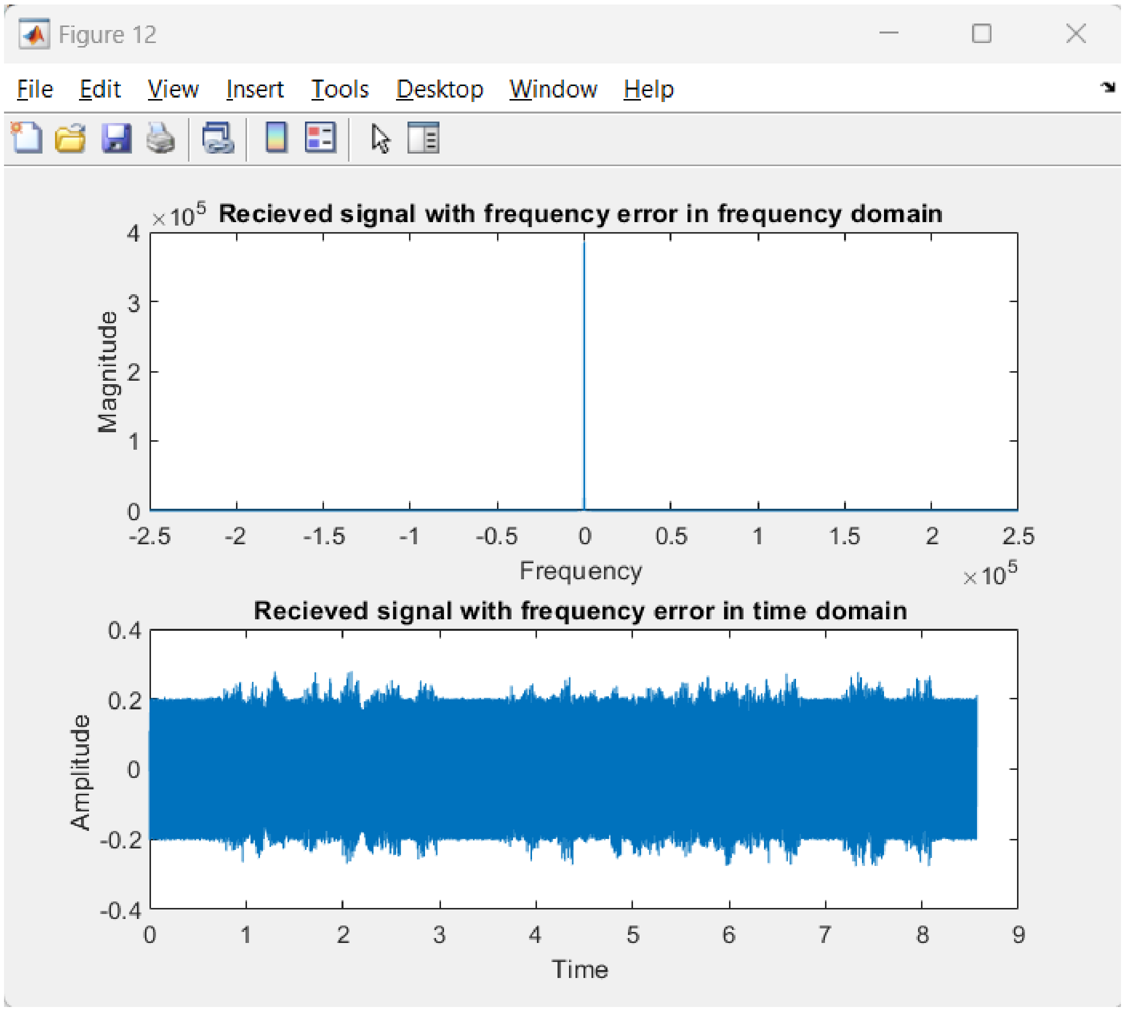
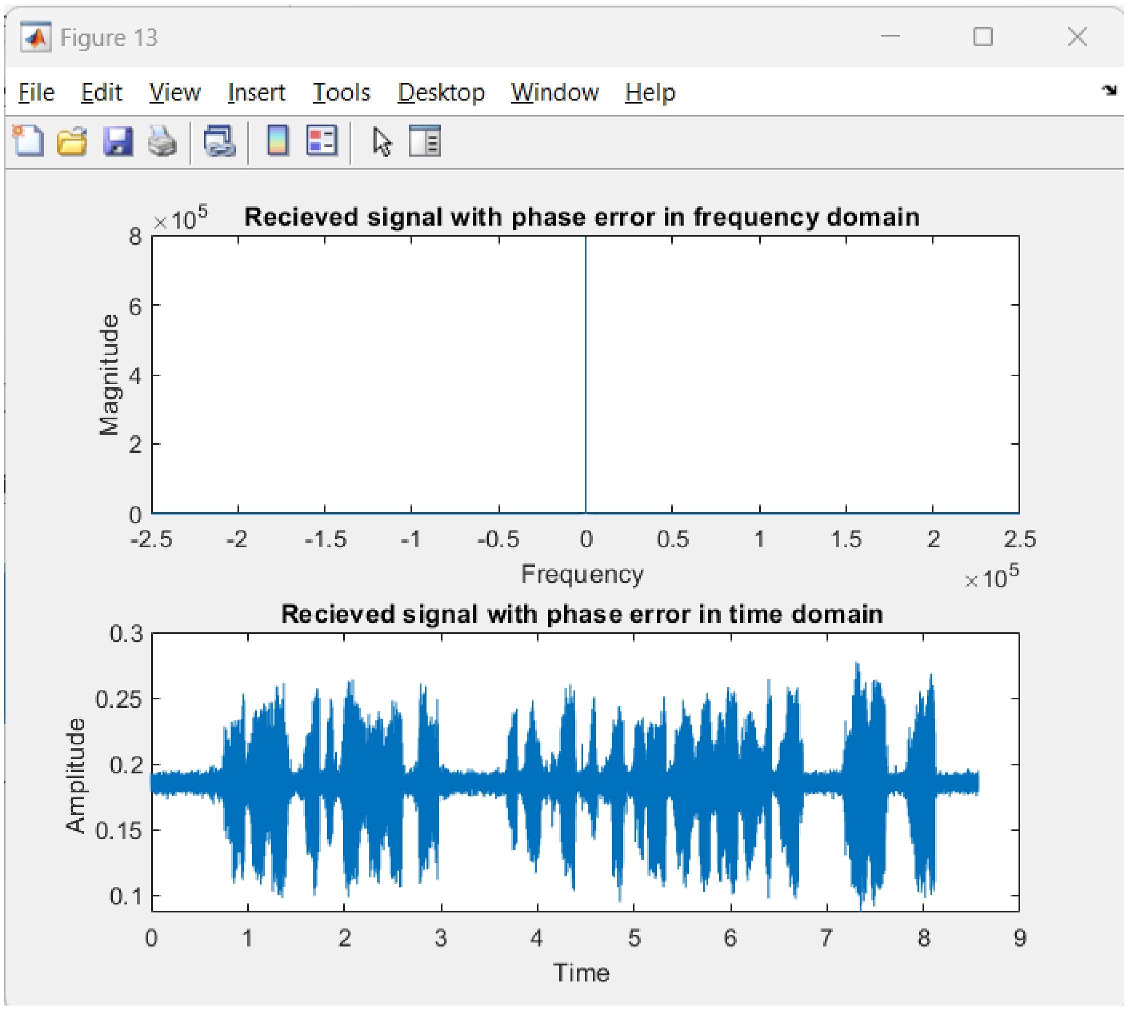


Figure 1
File Edit View Insert Tools Desktop Window Help
Recieved signal for SNR = 30 dB in frequency domain
Magnitude
Frequency
$\times 10^6$
$\times 10^5$
Recieved signal for SNR = 30 dB in time domain
Amplitude
Time

Received Sound after a 100MHz Frequency Error:



Sound received following a 20-degree phase error:

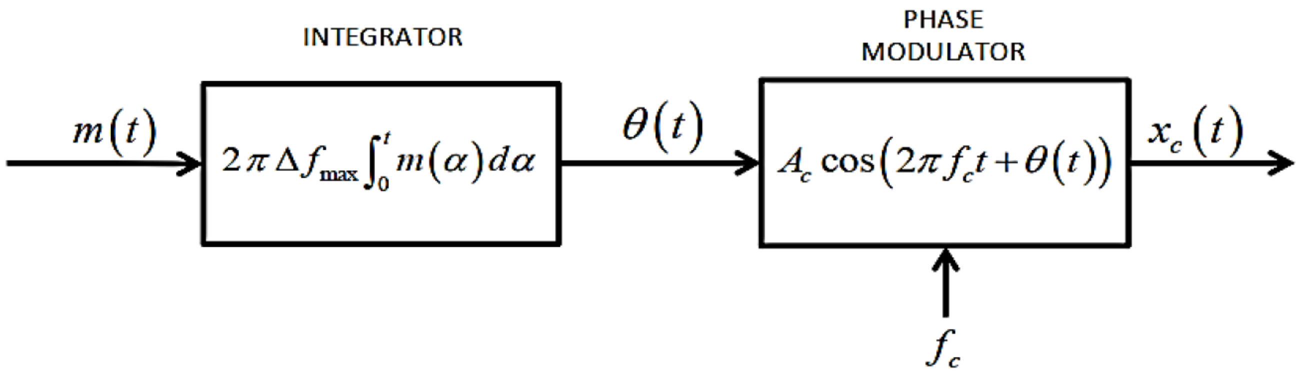


FM

Frequency modulation (FM) is a modulation type in which the instantaneous frequency of the carrier is changed according to the message amplitude. The motive behind the frequency modulation was to develop a scheme with inherent ability to combat noise. The noise, being usually modeled as additive, has a negative effect on the amplitude by introducing unavoidable random variations which are superimposed on the desired signal. Unlike the amplitude, frequency has a latent immunity against noise. Since it resides “away” from the amplitude, any changes in the amplitude would be completely irrelevant to the frequency. In other words, there is no direct correlation between the variation in amplitude and frequency, thus making FM a better candidate over AM with respect to noise immunity. However, what FM gains in noise immunity lacks in bandwidth efficiency. Since FM usually occupies larger bandwidth, AM is considered more bandwidth wise.

The NBFM condition is defined by the modulation index β it should be less than 1, therefore $\beta = \frac{\Delta f_{max}}{f_m} \ll 1$ which Δf_{max} is the maximum frequency and f_m is the message frequency.

$$x_c(t) = A_c \cos(2\pi f_c t + \theta(t))$$


INTEGRATOR
PHASE MODULATOR
$m(t)$
$2\pi\Delta f_{\max}\int_0^t m(\alpha)d\alpha$
$\theta(t)$
$A_c\cos(2\pi f_c t+\theta(t))$
$x_c(t)$
f_c

Code Snipps

- First, we import the input sound wave named as eric.wav it’s just for test, then we play it with sampling frequency 48000 then we wait for 8 seconds until it finishes.

```
% Importing sound wave %
FS = 48000 ;
[Y, FS] = audioread('eric.wav');

% Play sound wave %
fprintf('Original Sound is play :\n');
sound(Y, FS);

% Wait for the wave %
pause(8)
```

- Here we are plotting the time domain signal of the audio data, this done by creates a time vector T where each element represents the corresponding time for each sample in the audio signal. The division by FS converts the sample indices to time values.

```
% Plot time domain signal %
T = [(1 : length(Y)) / FS];
figure(1);
subplot(1, 2, 1);
plot(T, Y);
grid on ;
title('Time Domain Signal');
```

- Here we convert the time domain signal (Y) into the frequency domain using the Fast Fourier Transform (FFT). It then plots the magnitude of the frequency domain signal, fft(Y) computes the discrete Fourier transform, and fftshift shifts the zero-frequency component to the center of the spectrum, then compute the magnitude of the complex values, then we create a frequency vector F corresponding to the frequency range of the FFT result.

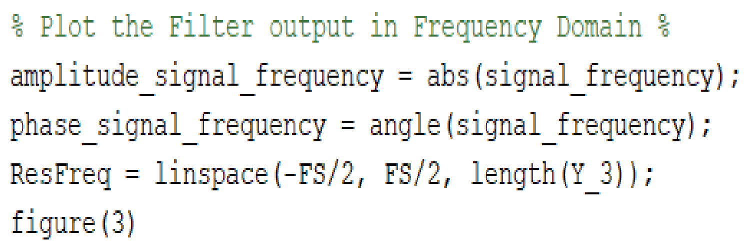
```
% Convert the Signal into Frequency Domain %
YF = fftshift(fft(Y));
YF_mag = abs(YF);

% Plot frequency domain signal %
F = linspace(-FS/2 , FS/2 , length(Y));
subplot(1, 2, 2);
plot(F, YF_mag);
grid on ;
title('Frequency Domain Signal');
```

- Here we are applying a low-pass filter (Butterworth filter) to the frequency domain representation of the audio signal then converting it back to the time domain using inverse FFT, this is done by calculating the number of ones in the filter, corresponding to the cutoff frequency of 8000 Hz and number of zeros to fill before and after the ones in the filter. Then This creates a Butterworth low-pass filter with zeros before and after the ones then applies the filter to the frequency domain representation then converts the

```
% Low Pass Filter (Butterworth) %
nOnes  = ceil((8000 * length(Y)) / FS);
nZeros = floor((length(YF) - nOnes) / 2);
LPF = [ zeros(nZeros,1) ; ones(nOnes,1) ;
        zeros(length(YF)-nOnes-nZeros,1)];
signal_frequency = YF.*LPF;
Y_3 = real(ifft(ifftshift(signal_frequency)));

% Play the Sound after the Filter %
fprintf('Sound after Filter is play :\n');
sound(Y_3, FS);

% Wait for the wave %

% Plot the Filter output in Time Domain %
T = linspace(0, length(Y_3)/FS, length(Y_3));
figure(2)
plot(T, Y_3);
xlabel('Time'); ylabel('Amplitude');
title('The Resived Signal in Time Domain');
```

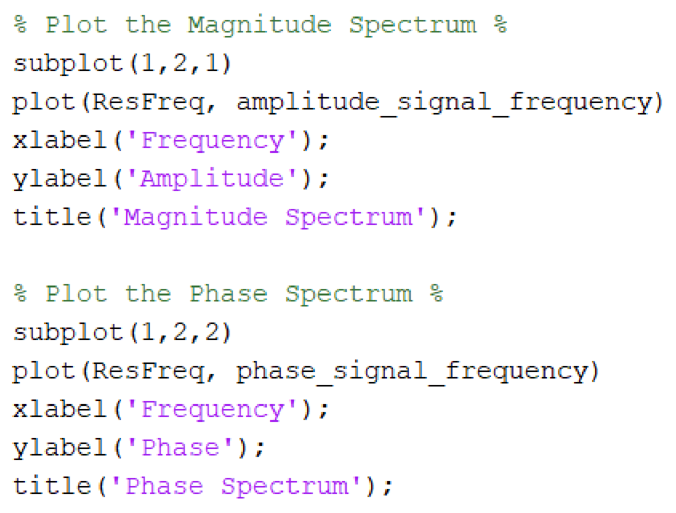
filtered frequency domain signal back to the time domain using the inverse FFT then the sound is paly after the filter then plots the time domain representation of the signal after applying the low-pass filter

- Here we are calculating the amplitude and phase spectrum of the filtered signal in the frequency domain first computes the magnitude of the complex values in signal_frequency then calculates the phase of the complex values in signal_frequency.



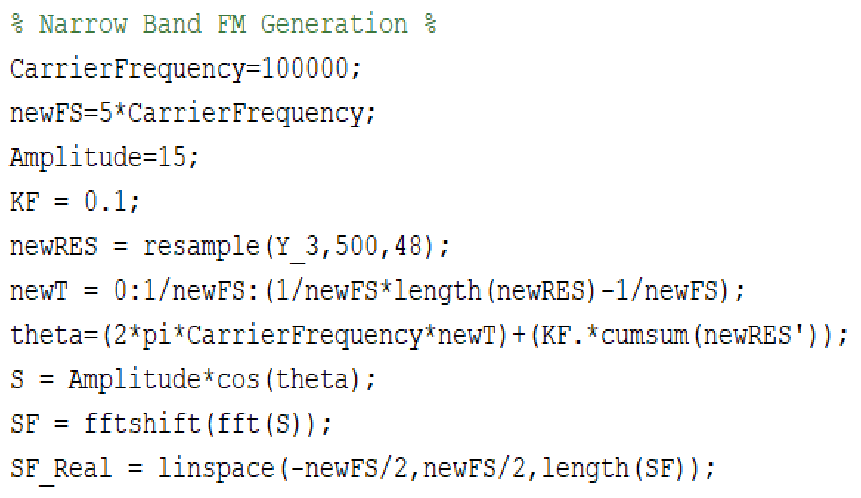
```
% Plot the Filter output in Frequency Domain %
amplitude_signal_frequency = abs(signal_frequency);
phase_signal_frequency = angle(signal_frequency);
ResFreq = linspace(-FS/2, FS/2, length(Y_3));
figure(3)
```

- plotting amplitude and phase spectrum of the filtered signal in the frequency domain.



```
% Plot the Magnitude Spectrum %
subplot(1,2,1)
plot(ResFreq, amplitude_signal_frequency)
xlabel('Frequency');
ylabel('Amplitude');
title('Magnitude Spectrum');

% Plot the Phase Spectrum %
subplot(1,2,2)
plot(ResFreq, phase_signal_frequency)
xlabel('Frequency');
ylabel('Phase');
title('Phase Spectrum');
```

- Here we are generating a narrow-band Frequency Modulation (FM) signal by specifying the carrier frequency, new sampling frequency, the amplitude, and the frequency deviation constant then we compute the frequency spectrum (SF) of the generated FM signal using FFT and shifts it to have the zero frequency in the center (fftshift). The frequency vector is stored in SF_Real and finally plot it.



```
% Narrow Band FM Generation %
CarrierFrequency=100000;
newFS=5*CarrierFrequency;
Amplitude=15;
KF = 0.1;
newRES = resample(Y_3,500,48);
newT = 0:1/newFS:(1/newFS*length(newRES)-1/newFS);
theta=(2*pi*CarrierFrequency*newT)+(KF.*cumsum(newRES'));
S = Amplitude*cos(theta);
SF = fftshift(fft(S));
SF_Real = linspace(-newFS/2,newFS/2,length(SF));
```

- Here we are performing signal demodulation on the narrow-band FM signal S. this is done by first Taking the derivative of the FM signal then Computing the envelope of the derivative using the Hilbert transform then Blocking the DC Component and finally Applying a low-pass filter in the frequency domain to remove high-frequency components then Performing inverse FFT to obtain the demodulated signal. Finally plot the output signal and play the wave.

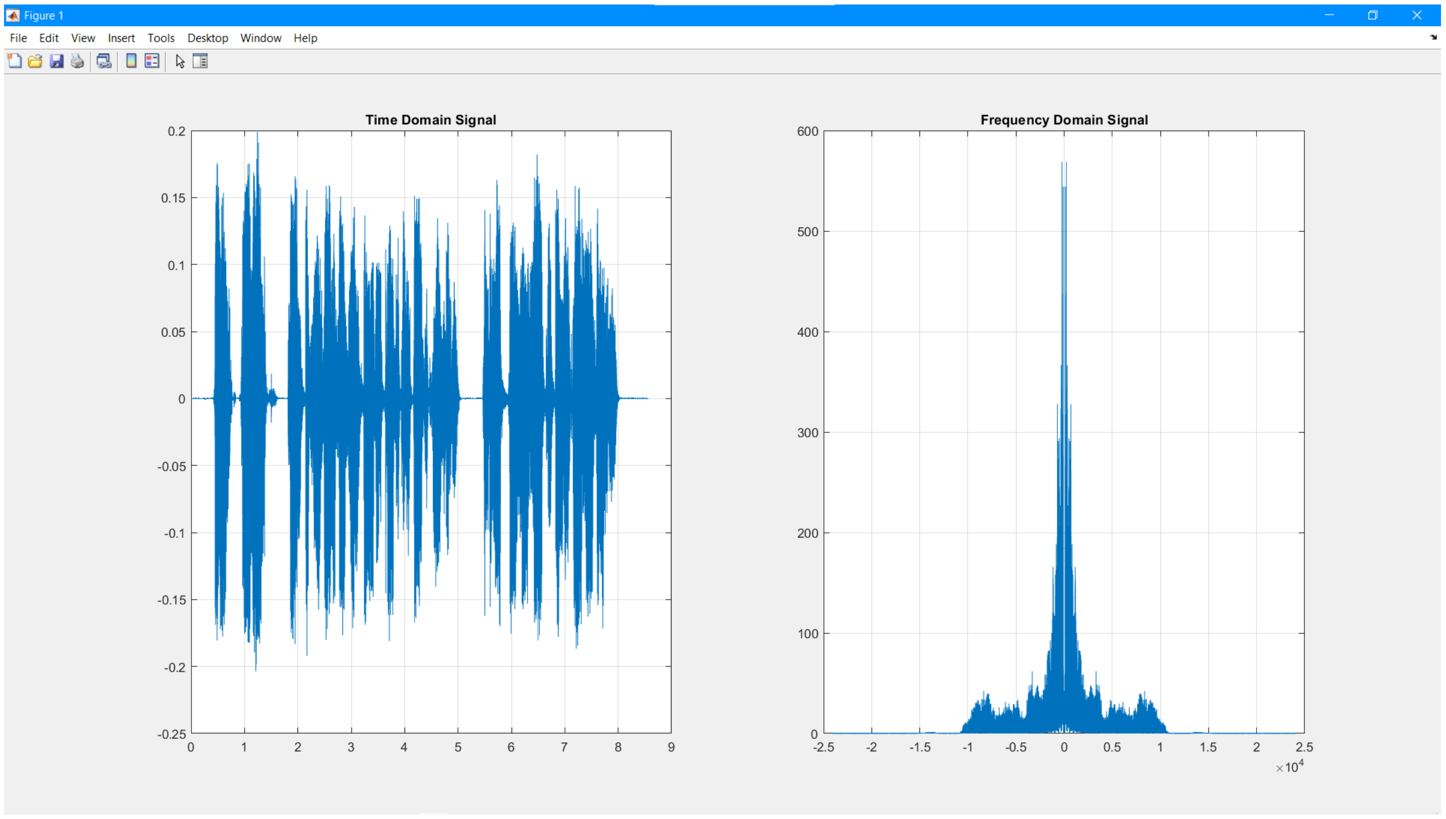
```
% Signal Demodulation %
S_diff= diff(S);
Envelop = abs(hilbert(S_diff));
Envelop = Envelop - mean(Envelop);
DeMod_Time = linspace(0,(length(Envelop)/newFS),length(Envelop));
R_env = fftshift(fft(Envelop))./length(Envelop);
n = length(R_env)/newFS;
DC_F = (newFS/2)- 1;
R_env( floor(DC_F*n) : end-floor(DC_F*n) ) = 0;
f = linspace(-newFS/2,newFS/2,length(R_env));
r = real(ifft(ifftshift(R_env))).*length(R_env);
```

```
% Plotting Narrow Band FM %
figure(4);
subplot(2,1,1);
plot(newT,S);
title('NBFM Signal - Time Domain');
subplot(2,1,2);
plot(SF_Real,abs(SF));
title('NBFM Signal - Frequency Domain');
```

```
% Plotting Demodulated Signal %
figure(5)
subplot(2,1,1)
plot(DeMod_Time,r)
title('Received Signal - Time Domain')
subplot(2,1,2)
plot(f,R_magnitude)
title('Received signal - Frequency Domain')
out = resample(r,48,500);
fprintf('Demodulated Sound is play :\n');
sound(out,FS);

% Wait for the wave %
pause(8);
```

Screenshots

The Input signal in time domain



The Filtered Signal in Time Domain

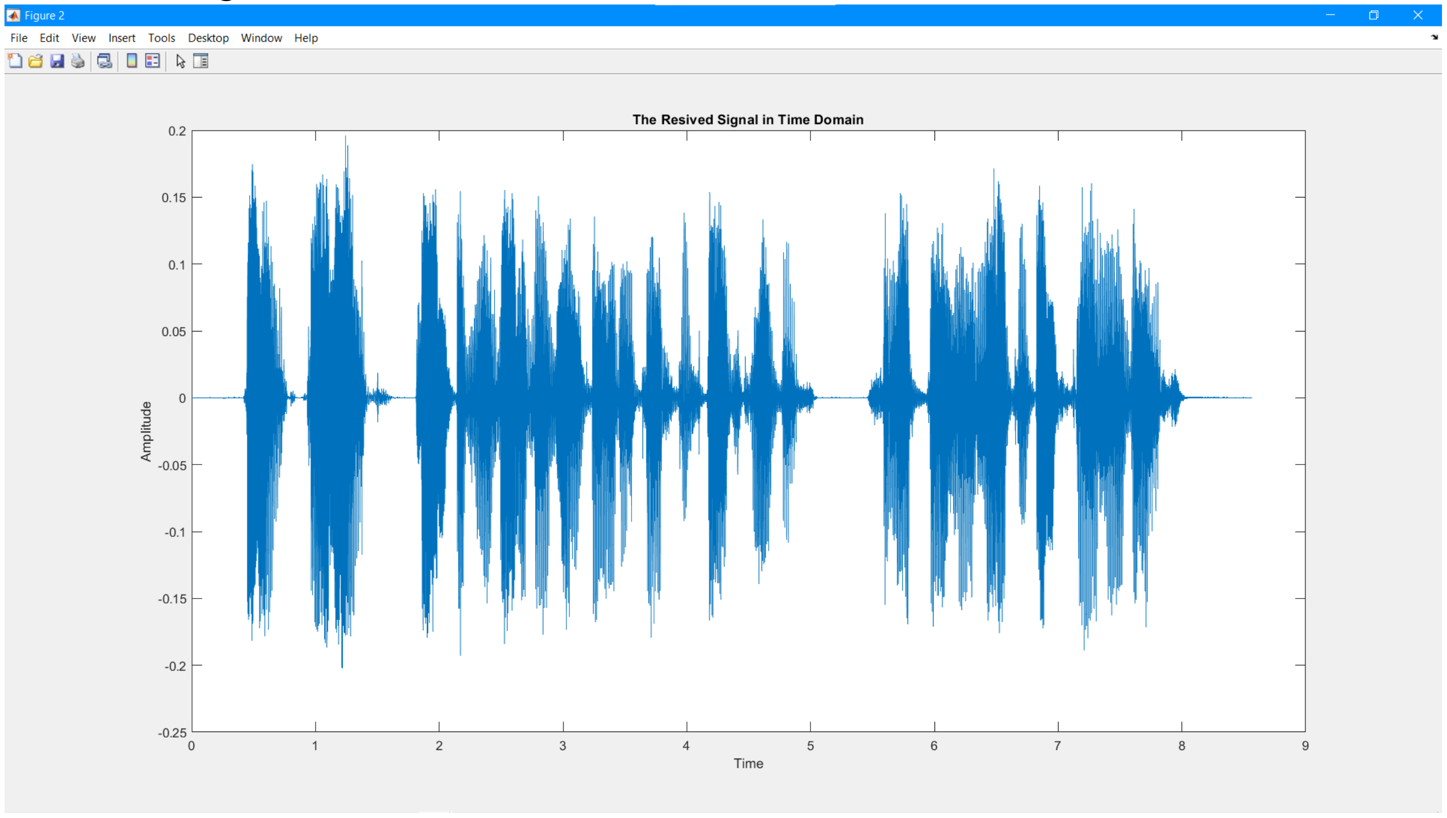
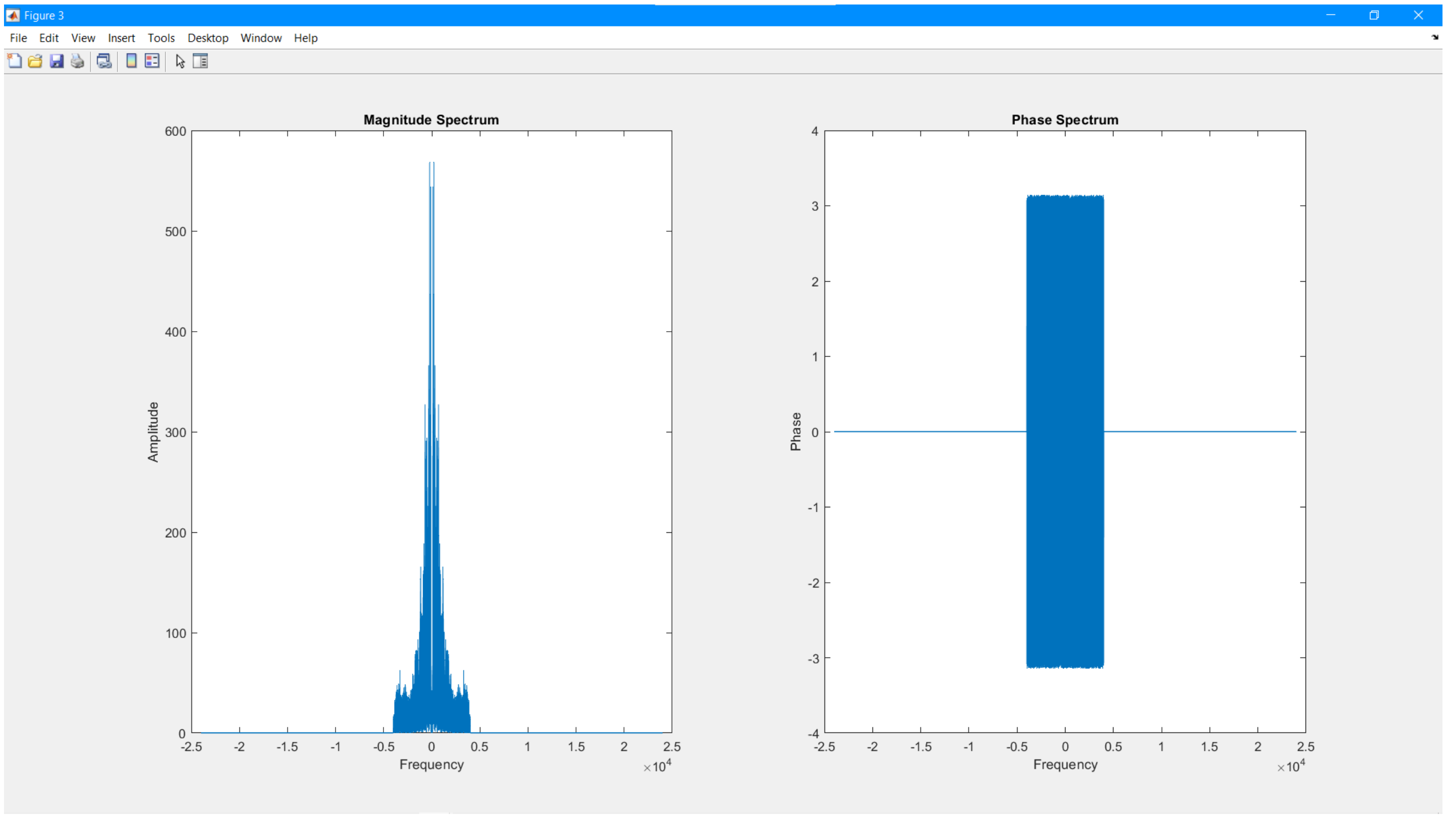
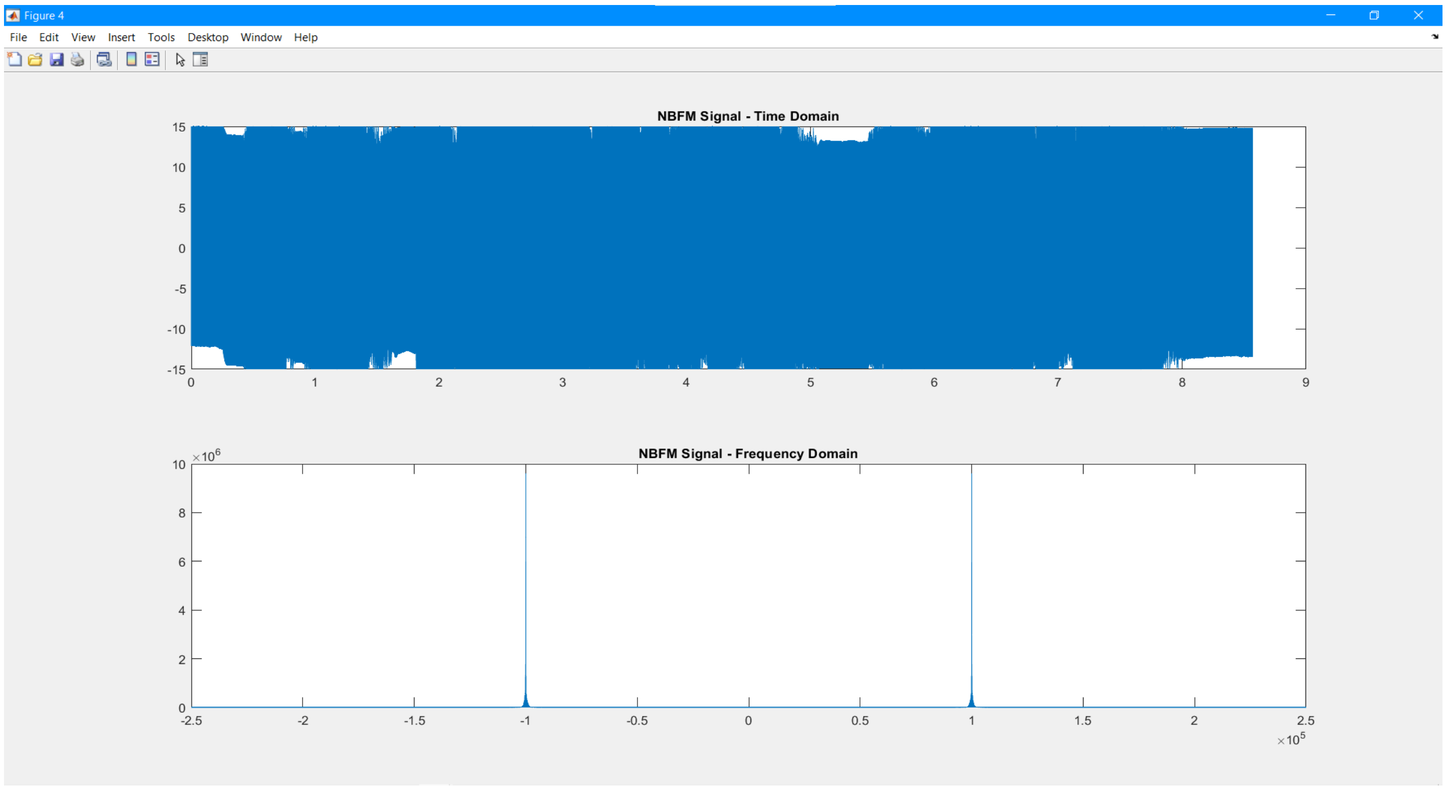


Figure 2
File Edit View Insert Tools Desktop Window Help
The Resived Signal in Time Domain
Amplitude
Time
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0 1 2 3 4 5 6 7 8 9

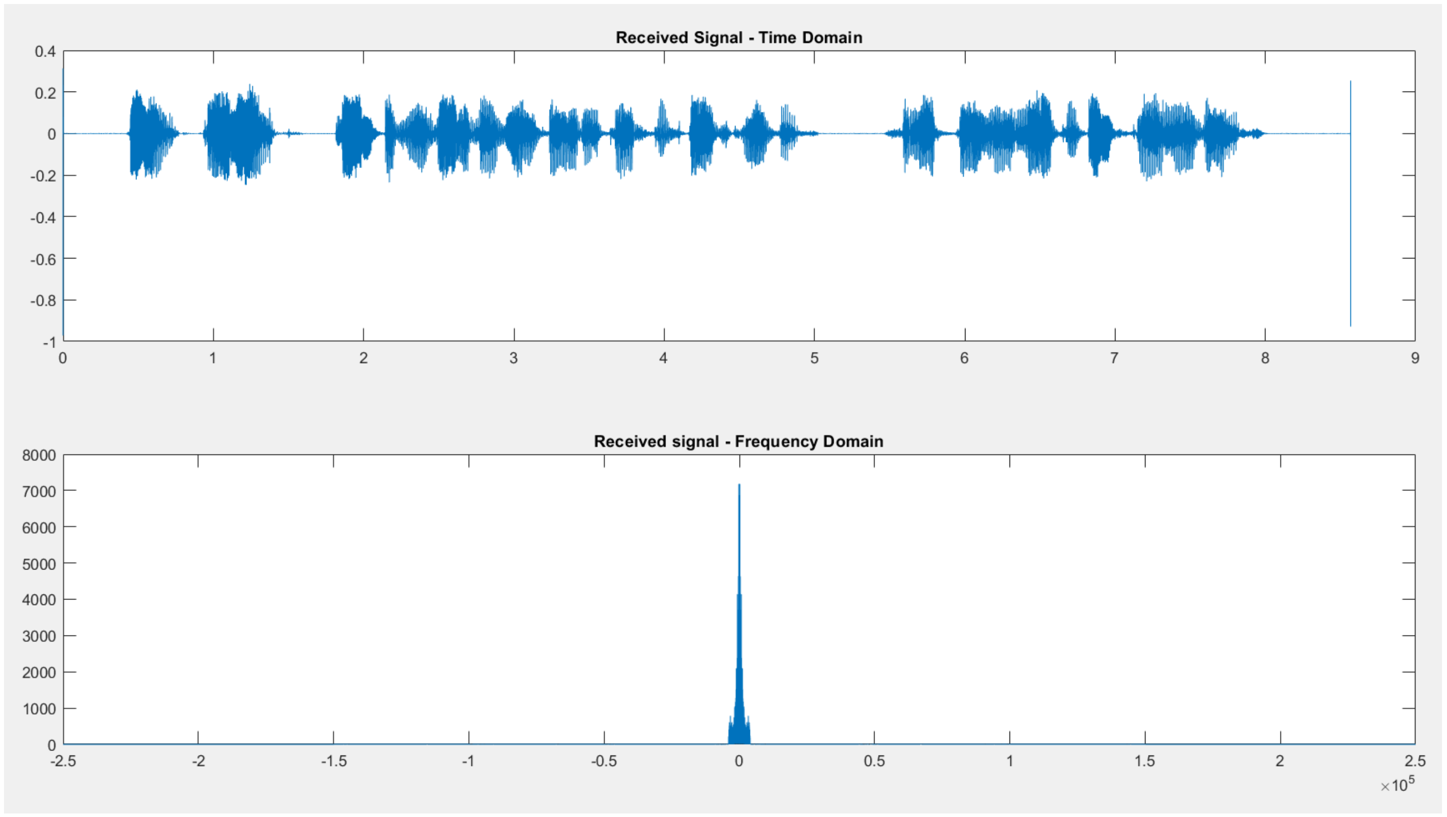
The Filtered Signal in Frequency Domain



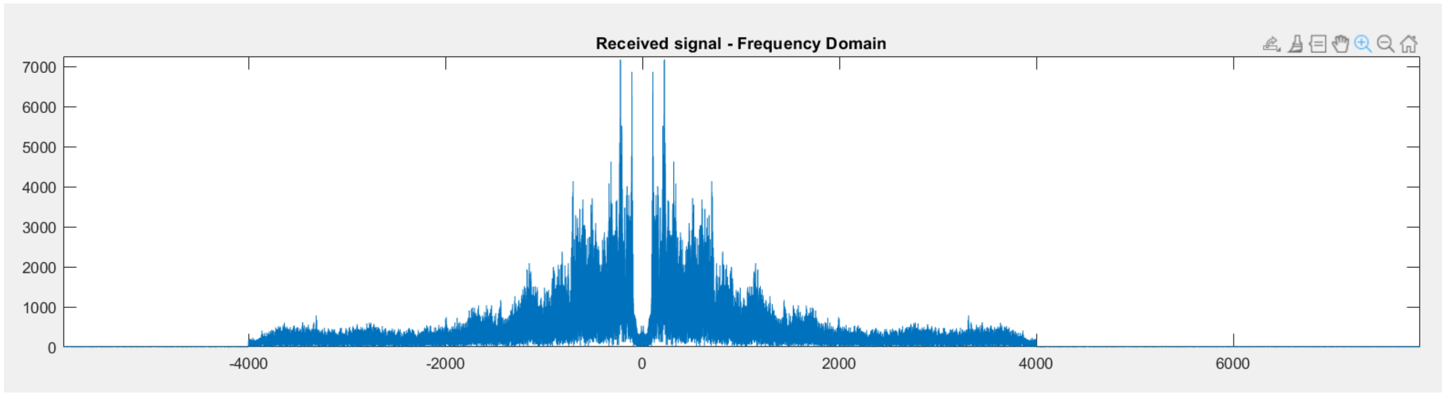
Narrow Band Frequency Modulated Signal in Time and Frequency Domain



The Demodulated Signal in Time and Frequency Domain



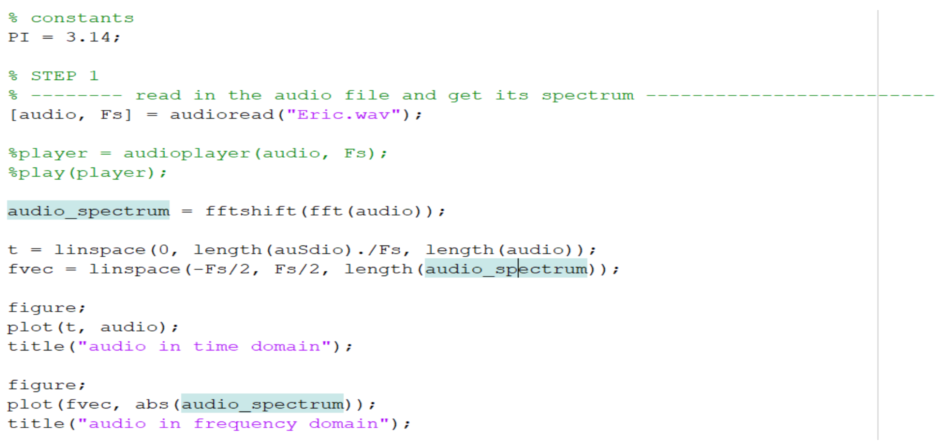
This is the same but zoomed in to check the filter validity.



SSB

Single Sideband (SSB) modulation is a technique where only one sideband of the carrier signal is transmitted, suppressing the other sideband and the carrier itself. Unlike traditional Amplitude Modulation (AM), which redundantly duplicates information in both sidebands and the carrier, SSB modulation conserves bandwidth by transmitting only essential information in a more compact form. This approach significantly improves bandwidth efficiency, making SSB modulation a more streamlined choice compared to AM. The suppression of one sideband optimizes spectrum utilization, addressing the inefficiency associated with AM and providing a more effective solution for scenarios where bandwidth conservation is crucial. However, it's essential to weigh the benefits of bandwidth efficiency against the challenges introduced, such as increased demodulation complexity and potential distortion.

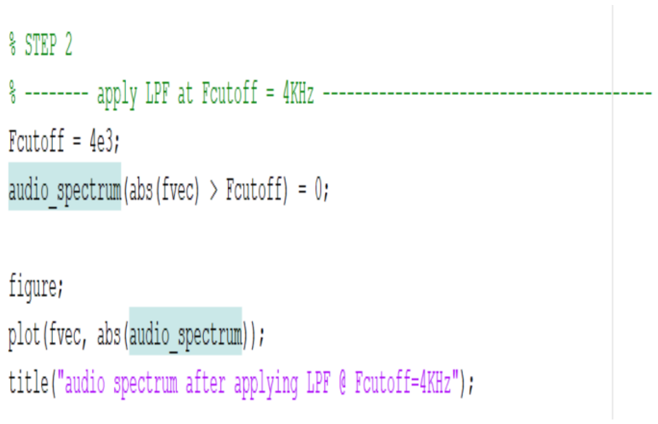
Code Snipps



```
% constants
PI = 3.14;

% STEP 1
% -------- read in the audio file and get its spectrum -------------------------
[audio, Fs] = audioread("Eric.wav");

%player = audioplayer(audio, Fs);
%play(player);

audio_spectrum = fftshift(fft(audio));

t = linspace(0, length(auSdio)./Fs, length(audio));
fvec = linspace(-Fs/2, Fs/2, length(audio_spectrum));

figure;
plot(t, audio);
title("audio in time domain");

figure;
plot(fvec, abs(audio_spectrum));
title("audio in frequency domain");
```

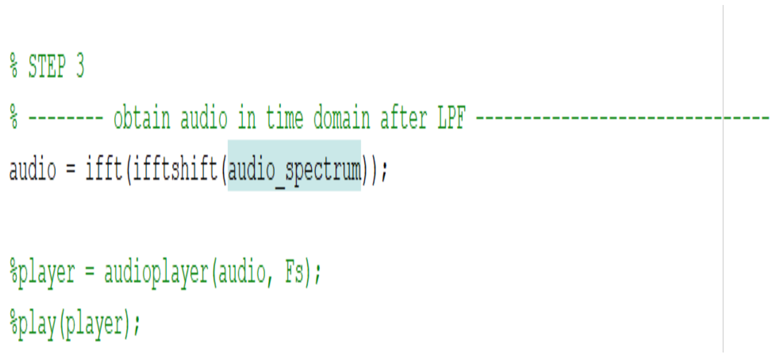
The code defines a constant, PI, with a value of approximately 3.14. It then reads an audio file named "Eric.wav" using the audioread function, storing the audio data in the variable 'audio' and the sampling frequency in 'Fs'. The code computes the spectrum of the audio signal using the Fast Fourier Transform (FFT) and plots the audio in both the time and frequency domains. The time and frequency vectors are created to represent time values and cover the frequency range, respectively. The optional audioplayer section is provided but commented out, allowing for the possibility of playing the audio using an audioplayer object.

In Step 2, a Low Pass Filter (LPF) is applied to the audio spectrum with a cutoff frequency of 4 kHz (Fcutoff = 4e3). The variable `audio_spectrum` represents the audio's frequency content obtained through the Fast Fourier Transform. By setting the magnitude of frequency components exceeding the cutoff frequency to zero, the high-frequency content above 4 kHz is filtered out. The resulting plot visualizes the modified audio spectrum, highlighting the impact of the LPF by attenuating or eliminating frequencies beyond the 4 kHz cutoff. This step is a common practice to control and limit the audio signal's bandwidth.



```
% STEP 2
% -------- apply LPF at Fcutoff = 4KHz ----------------------------------------
Fcutoff = 4e3;
audio_spectrum(abs(fvec) > Fcutoff) = 0;

figure;
plot(fvec, abs(audio_spectrum));
title("audio spectrum after applying LPF @ Fcutoff=4KHz");
```

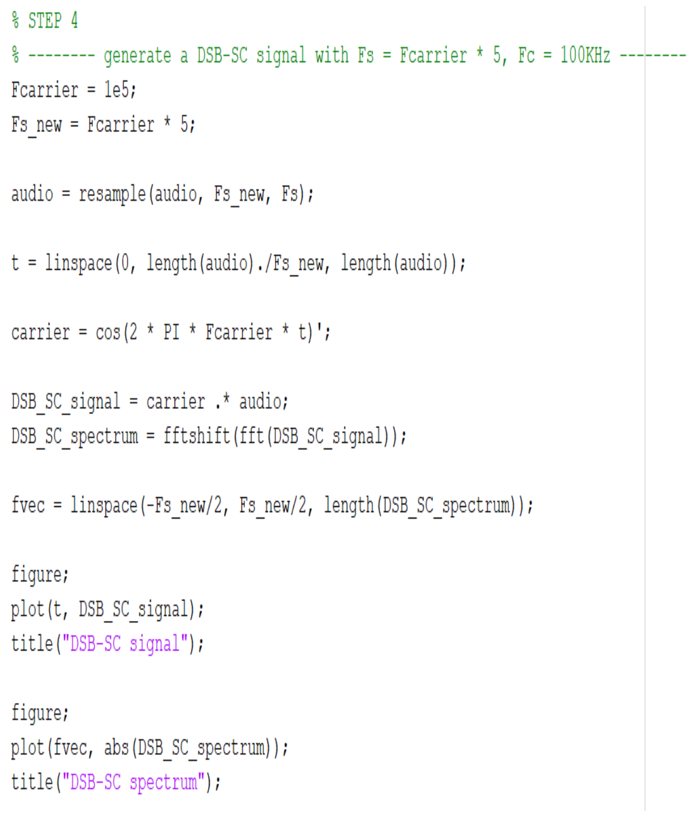
In Step 3, the inverse Fast Fourier Transform (ifft) is applied to the modified audio spectrum (audio_spectrum) to obtain the audio signal in the time domain. The ifftshift function is used to reverse the shift applied earlier in the frequency domain. The resulting audio variable represents the modified audio signal after the application of the Low Pass Filter (LPF).



```
% STEP 3
% -------- obtain audio in time domain after LPF ------------------------------
audio = ifft(ifftshift(audio_spectrum));

%player = audioplayer(audio, Fs);
%play(player);
```

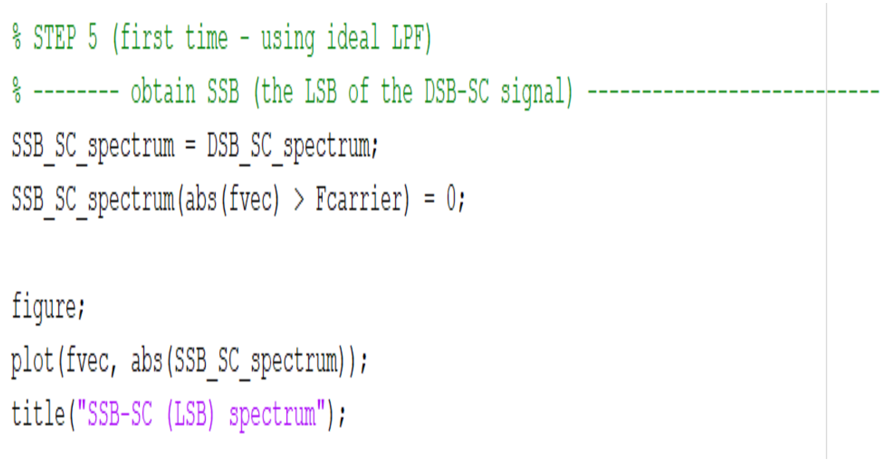
An audioplayer object (player) is created with the modified audio signal and the original sampling frequency (Fs). Finally, the play function is used to play the modified audio signal using the audioplayer object.

In Step 4, a Double Sideband-Suppressed Carrier (DSB-SC) signal is generated. The audio signal is resampled to a new sampling frequency (Fs_new) that is five times the carrier frequency (Fcarrier). A cosine carrier signal is created and multiplied by the resampled audio signal to obtain the DSB-SC signal. The time-domain and frequency-domain representations of the DSB-SC signal are then plotted for visualization.



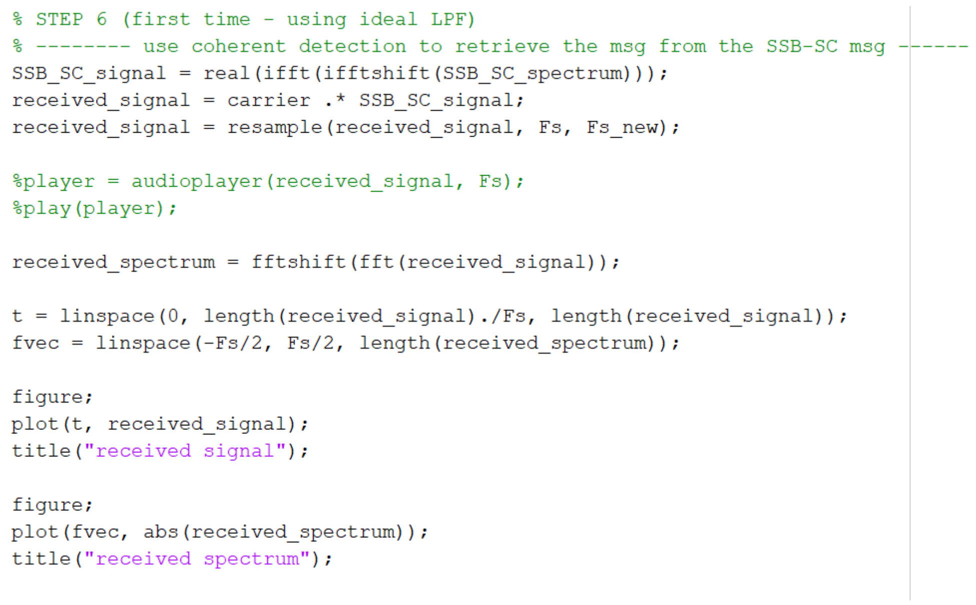
```
% STEP 4
% -------- generate a DSB-SC signal with Fs = Fcarrier * 5, Fc = 100KHz --------
Fcarrier = 1e5;
Fs_new = Fcarrier * 5;

audio = resample(audio, Fs_new, Fs);

t = linspace(0, length(audio)./Fs_new, length(audio));

carrier = cos(2 * PI * Fcarrier * t)';

DSB_SC_signal = carrier .* audio;
DSB_SC_spectrum = fftshift(fft(DSB_SC_signal));

fvec = linspace(-Fs_new/2, Fs_new/2, length(DSB_SC_spectrum));

figure;
plot(t, DSB_SC_signal);
title("DSB-SC signal");

figure;
plot(fvec, abs(DSB_SC_spectrum));
title("DSB-SC spectrum");
```

In Step 5, the Single Sideband (SSB) signal is obtained by suppressing the unwanted sideband in the Double Sideband-Suppressed Carrier (DSB-SC) signal. The spectrum of the SSB signal (SSB_SC_spectrum) is set to be identical to the DSB-SC spectrum initially. An ideal Low Pass Filter (LPF) is simulated by setting the magnitude of high-frequency components (above the carrier frequency) to zero. The resulting spectrum represents the Lower Sideband (LSB) of the SSB signal. The code then plots the spectrum of the SSB-SC (LSB) signal for visualization.



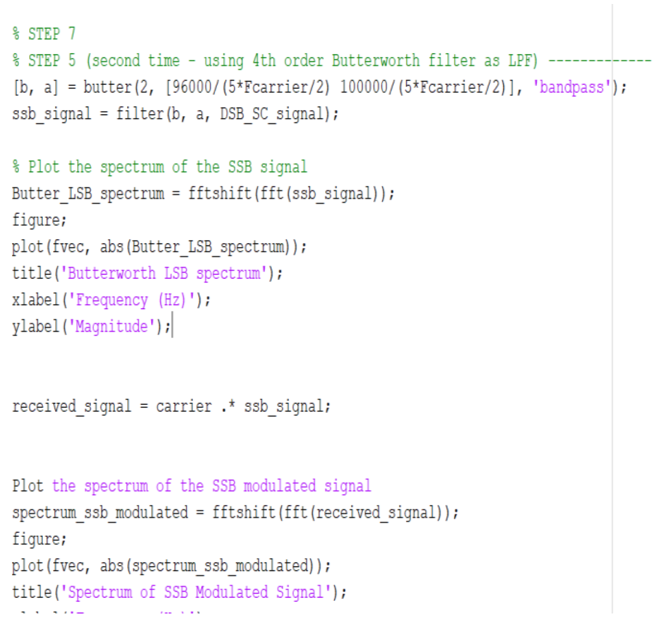
```
% STEP 5 (first time - using ideal LPF)
% -------- obtain SSB (the LSB of the DSB-SC signal) ---------------------------
SSB_SC_spectrum = DSB_SC_spectrum;
SSB_SC_spectrum(abs(fvec) > Fcarrier) = 0;

figure;
plot(fvec, abs(SSB_SC_spectrum));
title("SSB-SC (LSB) spectrum");
```

In Step 6, after using coherent detection to retrieve the message from the Single Sideband (SSB) signal, the received signal is plotted in both the time domain and frequency domain. The received signal is obtained by resampling the product of the carrier and the SSB signal to the original sampling frequency. The time-domain plot (received_signal) illustrates the recovered message signal, while the frequency-domain plot (received_spectrum) showcases the spectrum of the received signal after coherent detection.

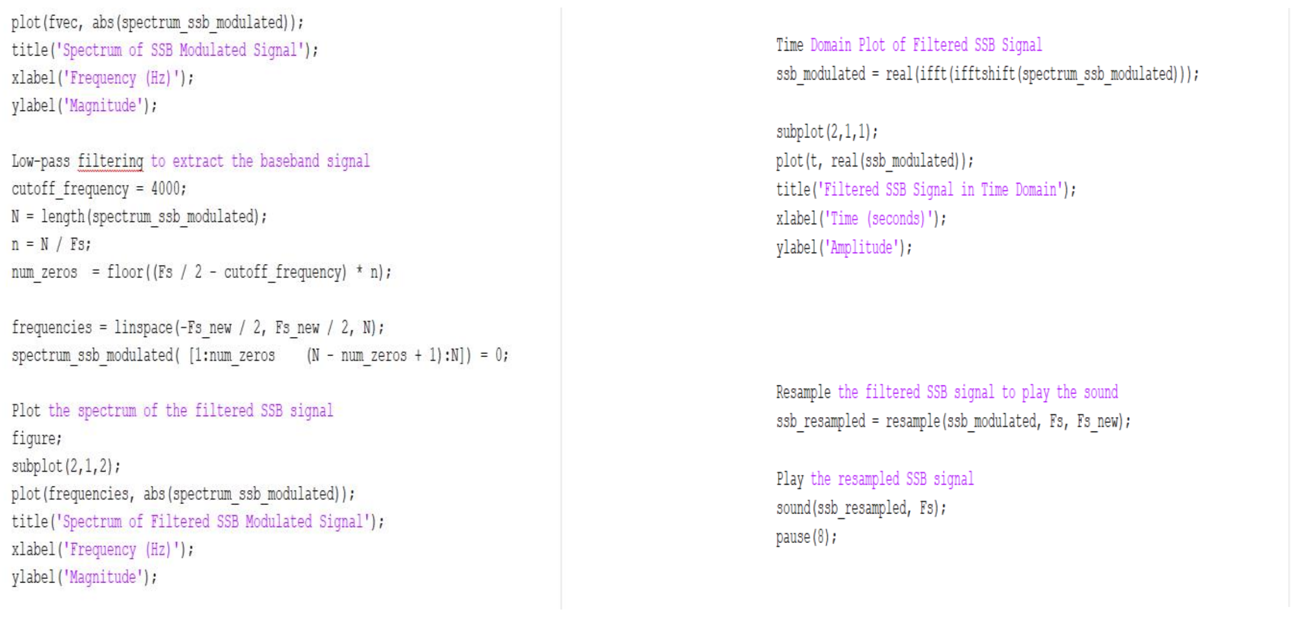


```
% STEP 6 (first time - using ideal LPF)
% -------- use coherent detection to retrieve the msg from the SSB-SC msg ------
SSB_SC_signal = real(ifft(ifftshift(SSB_SC_spectrum)));
received_signal = carrier .* SSB_SC_signal;
received_signal = resample(received_signal, Fs, Fs_new);

%player = audioplayer(received_signal, Fs);
%play(player);

received_spectrum = fftshift(fft(received_signal));

t = linspace(0, length(received_signal)./Fs, length(received_signal));
fvec = linspace(-Fs/2, Fs/2, length(received_spectrum));

figure;
plot(t, received_signal);
title("received signal");

figure;
plot(fvec, abs(received_spectrum));
title("received spectrum");
```

In Step 7, the MATLAB code refines the Single Sideband (SSB) modulation process introduced in Step 5 by employing a 4th order Butterworth filter as a bandpass filter. This Butterworth filter is designed with cutoff frequencies determined by the carrier frequency, aiming to enhance the modulation of the Double Sideband Suppressed Carrier (DSB-SC) signal. The filtered SSB signal is obtained through the application of the Butterworth filter, and its frequency spectrum is visualized. Following modulation with a carrier signal, the code showcases the resulting SSB-modulated signal in both the time and frequency domains. Additionally, a low-pass filter is applied to extract the baseband signal, effectively focusing on the essential information. The time and frequency domain representations of the filtered SSB signal are then presented, illustrating the impact of the Butterworth filter on the modulation process. Finally, the resampled and filtered SSB signal is played, allowing for an auditory understanding of the modified SSB modulation.

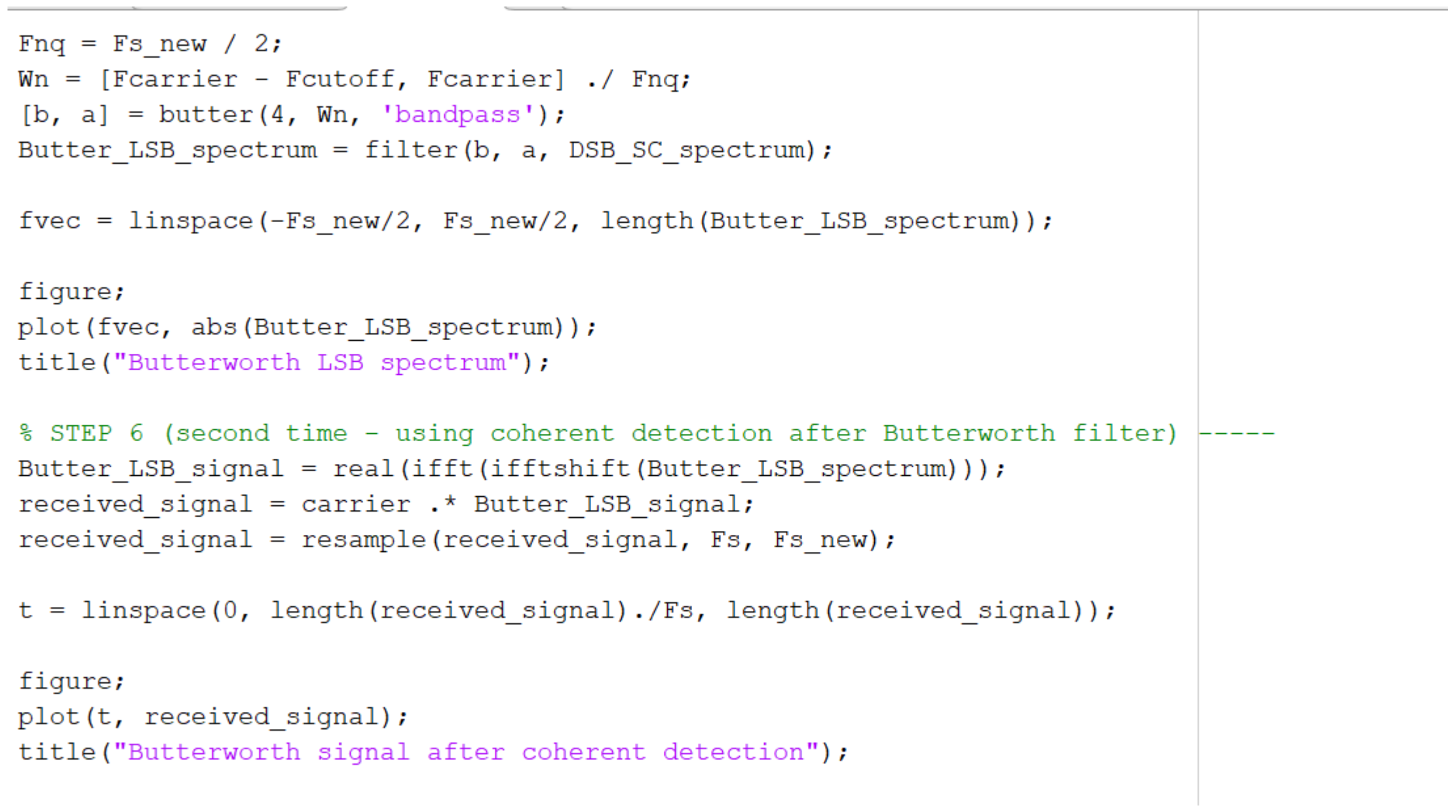


```
% STEP 7
% STEP 5 (second time - using 4th order Butterworth filter as LPF) ------------
[b, a] = butter(2, [96000/(5*Fcarrier/2) 100000/(5*Fcarrier/2)], 'bandpass');
ssb_signal = filter(b, a, DSB_SC_signal);

% Plot the spectrum of the SSB signal
Butter_LSB_spectrum = fftshift(fft(ssb_signal));
figure;
plot(fvec, abs(Butter_LSB_spectrum));
title('Butterworth LSB spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


received_signal = carrier .* ssb_signal;


Plot the spectrum of the SSB modulated signal
spectrum_ssb_modulated = fftshift(fft(received_signal));
figure;
plot(fvec, abs(spectrum_ssb_modulated));
title('Spectrum of SSB Modulated Signal');
```



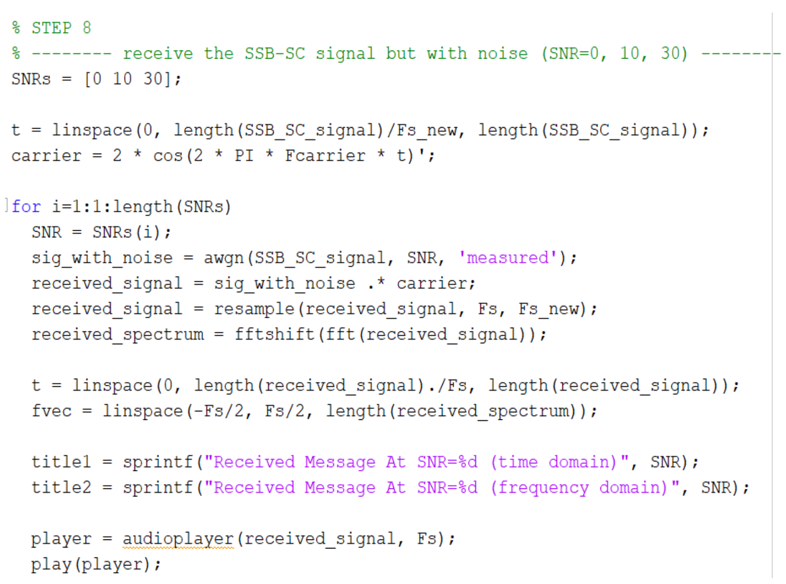
```
plot(fvec, abs(spectrum_ssb_modulated));
title('Spectrum of SSB Modulated Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

Low-pass filtering to extract the baseband signal
cutoff_frequency = 4000;
N = length(spectrum_ssb_modulated);
n = N / Fs;
num_zeros  = floor((Fs / 2 - cutoff_frequency) * n);

frequencies = linspace(-Fs_new / 2, Fs_new / 2, N);
spectrum_ssb_modulated( [1:num_zeros    (N - num_zeros + 1):N]) = 0;

Plot the spectrum of the filtered SSB signal
figure;
subplot(2,1,2);
plot(frequencies, abs(spectrum_ssb_modulated));
title('Spectrum of Filtered SSB Modulated Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

```
Time Domain Plot of Filtered SSB Signal
ssb_modulated = real(ifft(ifftshift(spectrum_ssb_modulated)));

subplot(2,1,1);
plot(t, real(ssb_modulated));
title('Filtered SSB Signal in Time Domain');
xlabel('Time (seconds)');
ylabel('Amplitude');



Resample the filtered SSB signal to play the sound
ssb_resampled = resample(ssb_modulated, Fs, Fs_new);

Play the resampled SSB signal
sound(ssb_resampled, Fs);
pause(8);
```

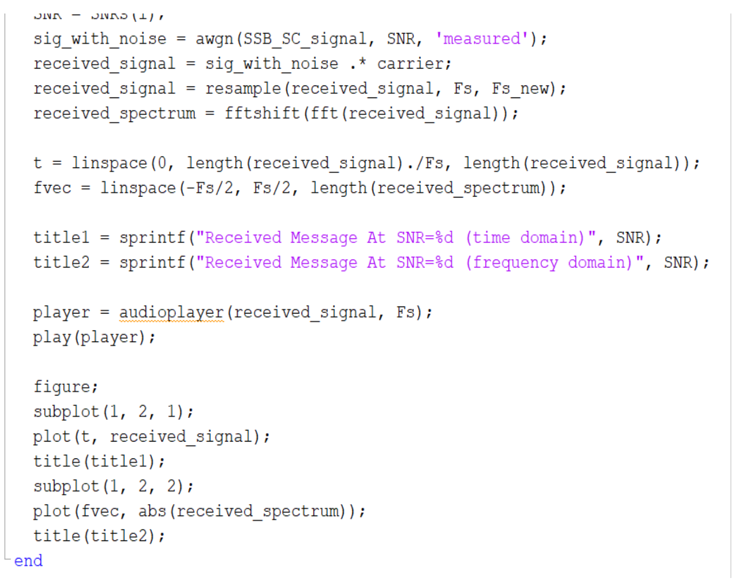


```
Fnq = Fs_new / 2;
Wn = [Fcarrier - Fcutoff, Fcarrier] ./ Fnq;
[b, a] = butter(4, Wn, 'bandpass');
Butter_LSB_spectrum = filter(b, a, DSB_SC_spectrum);

fvec = linspace(-Fs_new/2, Fs_new/2, length(Butter_LSB_spectrum));

figure;
plot(fvec, abs(Butter_LSB_spectrum));
title("Butterworth LSB spectrum");

% STEP 6 (second time - using coherent detection after Butterworth filter) -----
Butter_LSB_signal = real(ifft(ifftshift(Butter_LSB_spectrum)));
received_signal = carrier .* Butter_LSB_signal;
received_signal = resample(received_signal, Fs, Fs_new);

t = linspace(0, length(received_signal)./Fs, length(received_signal));

figure;
plot(t, received_signal);
title("Butterworth signal after coherent detection");
```

In Step 8, the SSB-SC signal is received with different signal-to-noise ratios (SNRs). The loop iterates through SNRs of 0, 10, and 30. For each SNR, AWGN (Additive White Gaussian Noise) is added to the SSB-SC signal, and the received signal is obtained by multiplying it with the carrier. The received signal is then resampled to the original sampling frequency, and its time and frequency domain representations are plotted. Audioplayer is used to play the received signal, and the plots show the impact of noise at different SNRs on the received signal.

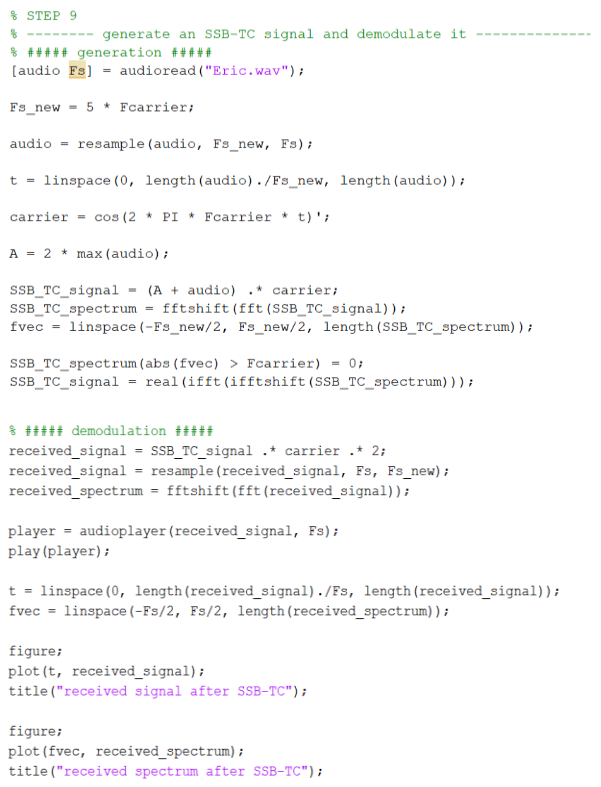


```
% STEP 8
% -------- receive the SSB-SC signal but with noise (SNR=0, 10, 30) --------
SNRs = [0 10 30];

t = linspace(0, length(SSB_SC_signal)/Fs_new, length(SSB_SC_signal));
carrier = 2 * cos(2 * PI * Fcarrier * t)';

for i=1:1:length(SNRs)
  SNR = SNRs(i);
  sig_with_noise = awgn(SSB_SC_signal, SNR, 'measured');
  received_signal = sig_with_noise .* carrier;
  received_signal = resample(received_signal, Fs, Fs_new);
  received_spectrum = fftshift(fft(received_signal));

  t = linspace(0, length(received_signal)./Fs, length(received_signal));
  fvec = linspace(-Fs/2, Fs/2, length(received_spectrum));

  title1 = sprintf("Received Message At SNR=%d (time domain)", SNR);
  title2 = sprintf("Received Message At SNR=%d (frequency domain)", SNR);

  player = audioplayer(received_signal, Fs);
  play(player);
```



```
  SNR = SNRs(i);
  sig_with_noise = awgn(SSB_SC_signal, SNR, 'measured');
  received_signal = sig_with_noise .* carrier;
  received_signal = resample(received_signal, Fs, Fs_new);
  received_spectrum = fftshift(fft(received_signal));

  t = linspace(0, length(received_signal)./Fs, length(received_signal));
  fvec = linspace(-Fs/2, Fs/2, length(received_spectrum));

  title1 = sprintf("Received Message At SNR=%d (time domain)", SNR);
  title2 = sprintf("Received Message At SNR=%d (frequency domain)", SNR);

  player = audioplayer(received_signal, Fs);
  play(player);

  figure;
  subplot(1, 2, 1);
  plot(t, received_signal);
  title(title1);
  subplot(1, 2, 2);
  plot(fvec, abs(received_spectrum));
  title(title2);
end
```

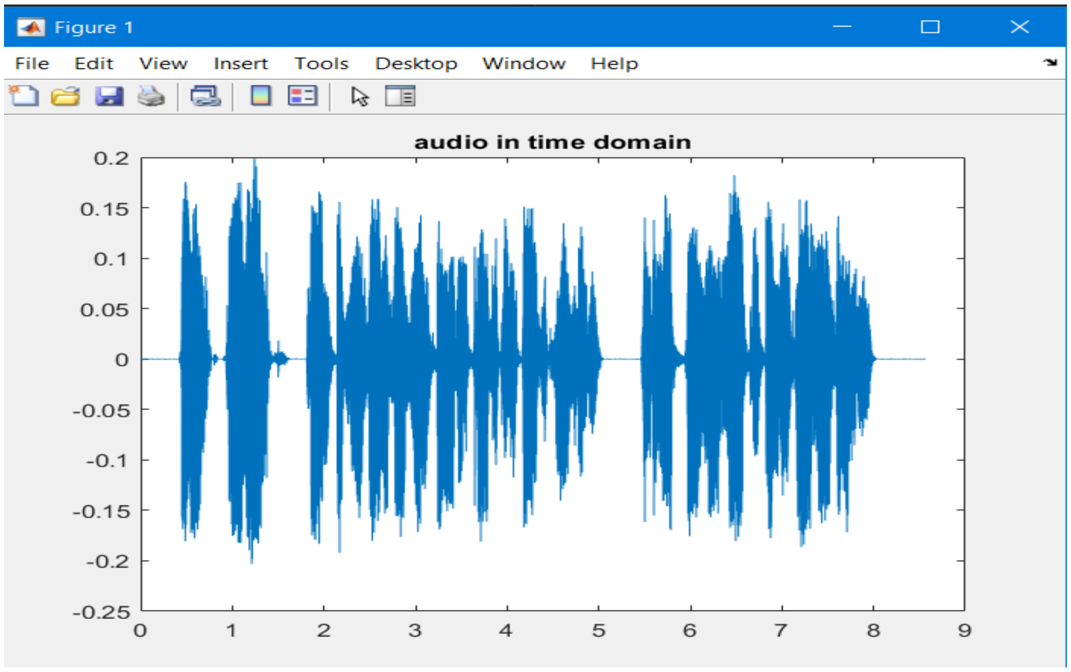
In Step 9, an Single Sideband with Tone Carrier (SSB-TC) signal is generated and demodulated. The SSB-TC signal is created by modulating the audio signal onto a suppressed carrier with an additional amplitude term. The signal is then filtered using a Low Pass Filter (LPF) in the frequency domain, and the demodulation involves multiplying the filtered SSB-TC signal by the carrier and resampling to obtain the received signal. The time-domain and frequency-domain representations of the received signal after SSB-TC demodulation are then plotted, and the received signal is played using audioplayer.



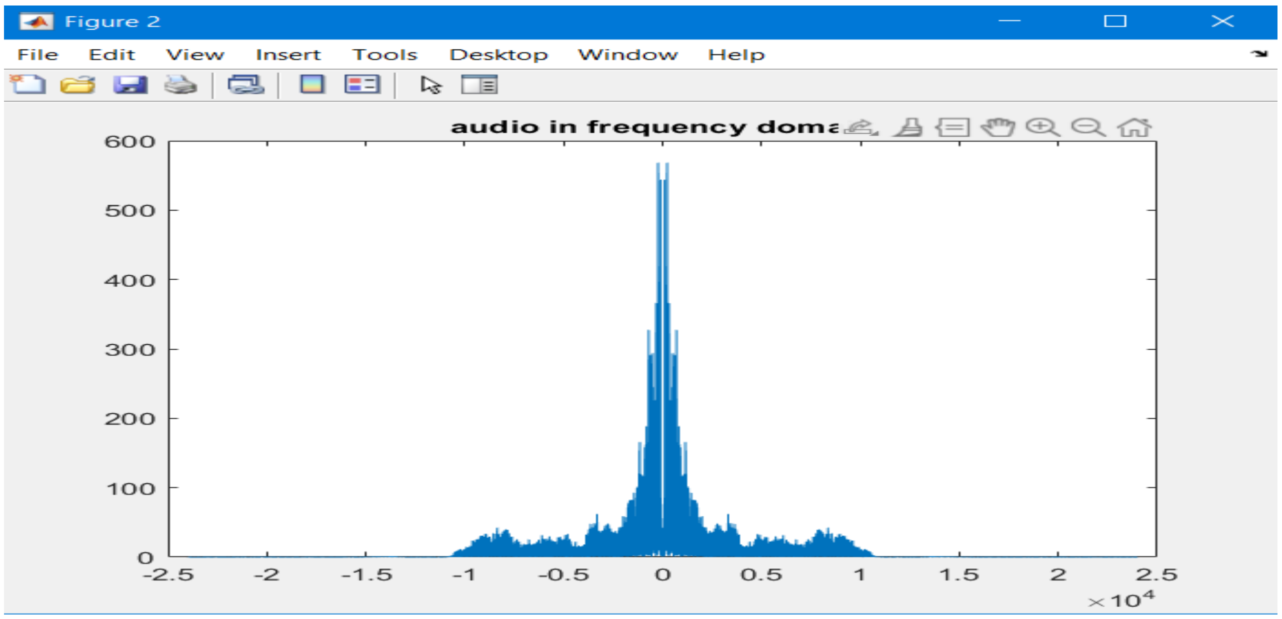
```
% STEP 9
% -------- generate an SSB-TC signal and demodulate it --------------
% ##### generation #####
[audio Fs] = audioread("Eric.wav");

Fs_new = 5 * Fcarrier;

audio = resample(audio, Fs_new, Fs);

t = linspace(0, length(audio)./Fs_new, length(audio));

carrier = cos(2 * PI * Fcarrier * t)';

A = 2 * max(audio);

SSB_TC_signal = (A + audio) .* carrier;
SSB_TC_spectrum = fftshift(fft(SSB_TC_signal));
fvec = linspace(-Fs_new/2, Fs_new/2, length(SSB_TC_spectrum));

SSB_TC_spectrum(abs(fvec) > Fcarrier) = 0;
SSB_TC_signal = real(ifft(ifftshift(SSB_TC_spectrum)));


% ##### demodulation #####
received_signal = SSB_TC_signal .* carrier .* 2;
received_signal = resample(received_signal, Fs, Fs_new);
received_spectrum = fftshift(fft(received_signal));

player = audioplayer(received_signal, Fs);
play(player);

t = linspace(0, length(received_signal)./Fs, length(received_signal));
fvec = linspace(-Fs/2, Fs/2, length(received_spectrum));

figure;
plot(t, received_signal);
title("received signal after SSB-TC");

figure;
plot(fvec, received_spectrum);
title("received spectrum after SSB-TC");
```

Screenshots

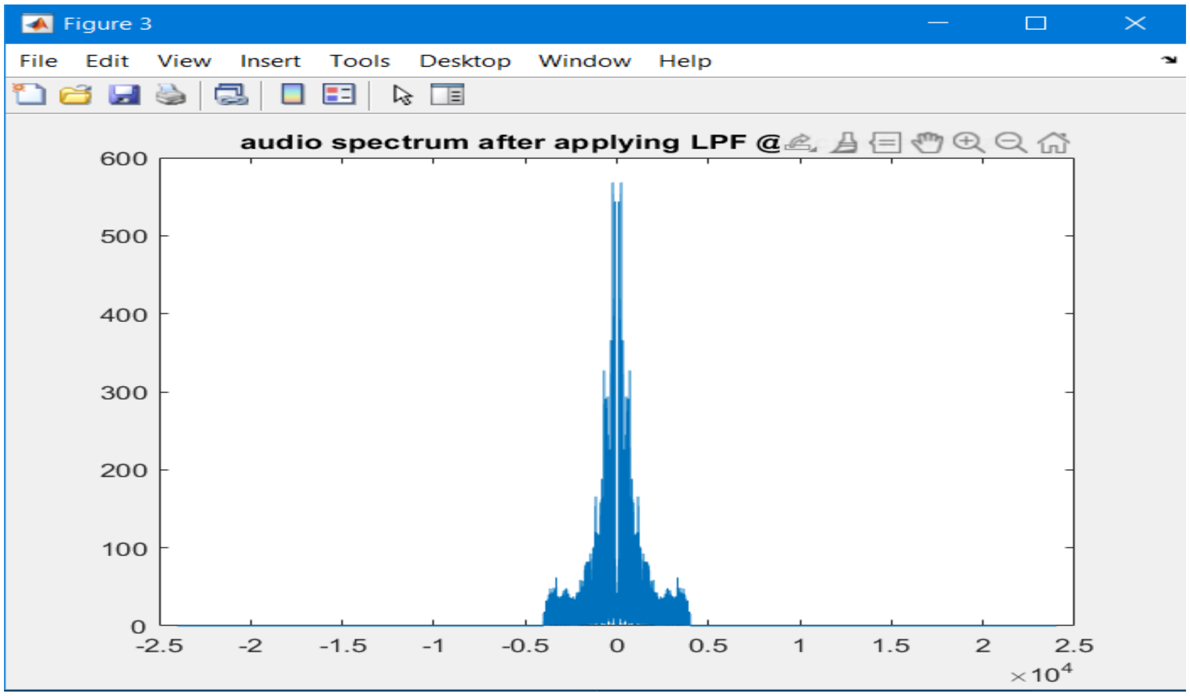
Audio in time domain



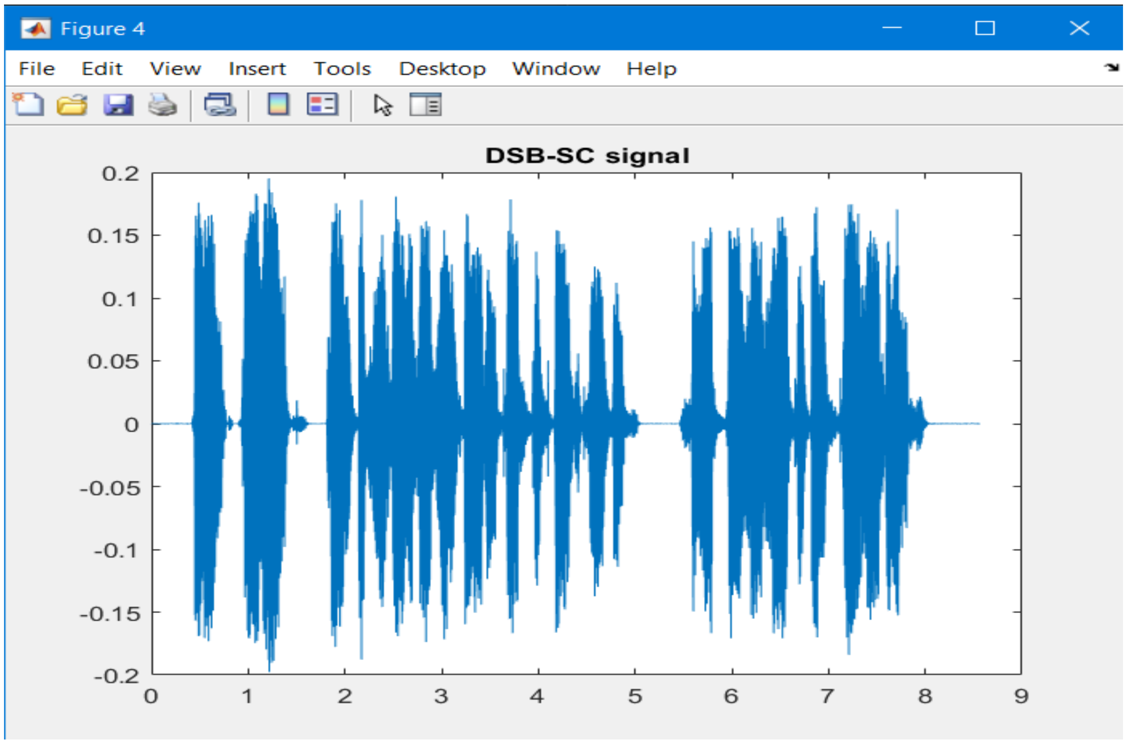
Audio in frequency domain



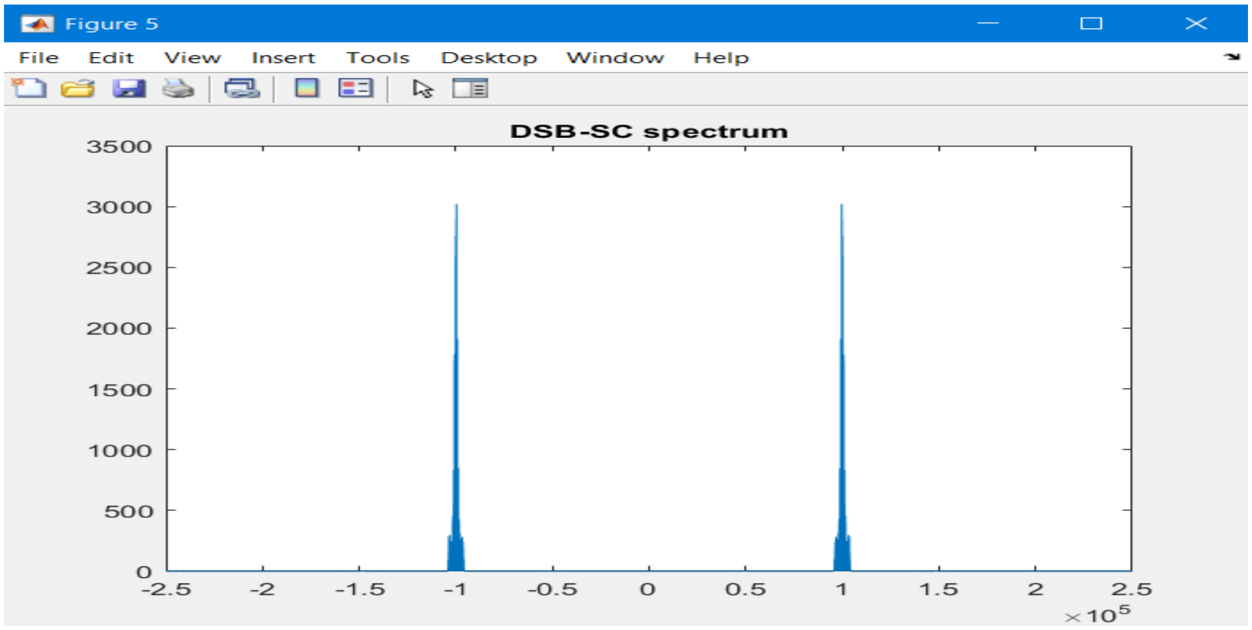
audio spectrum after applying LPF at Fcutoff=4KHz



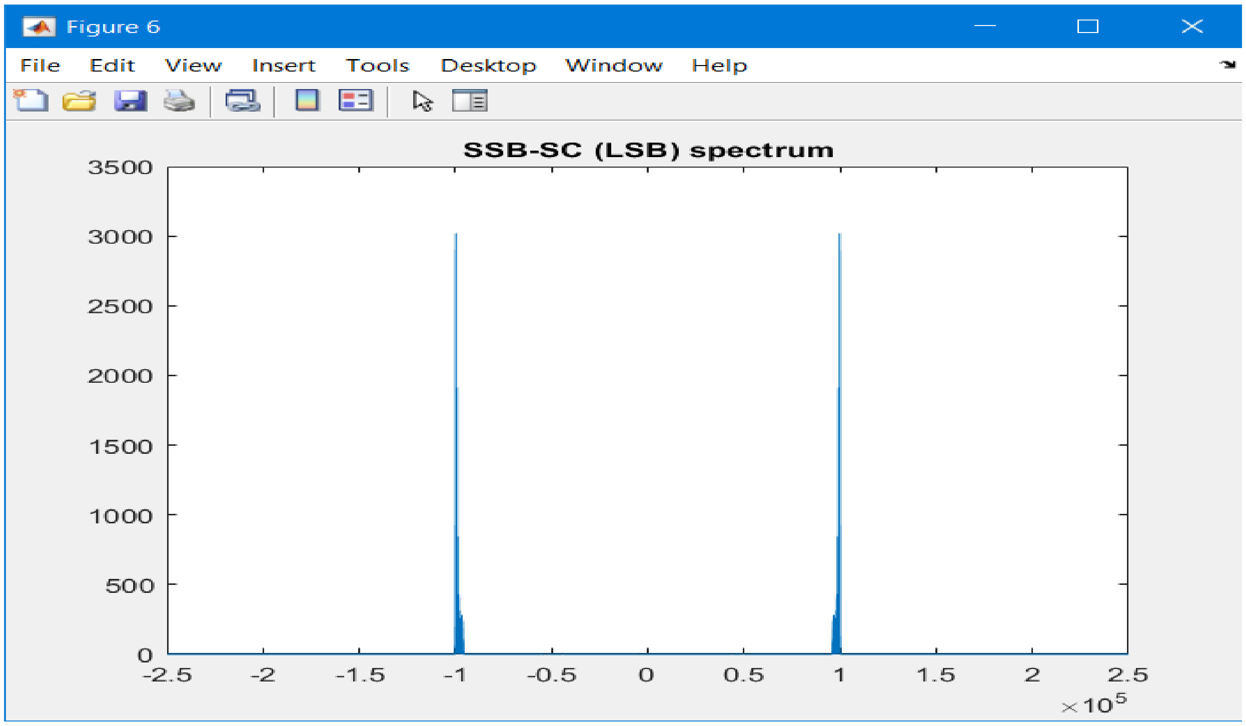
DSB-SC signal



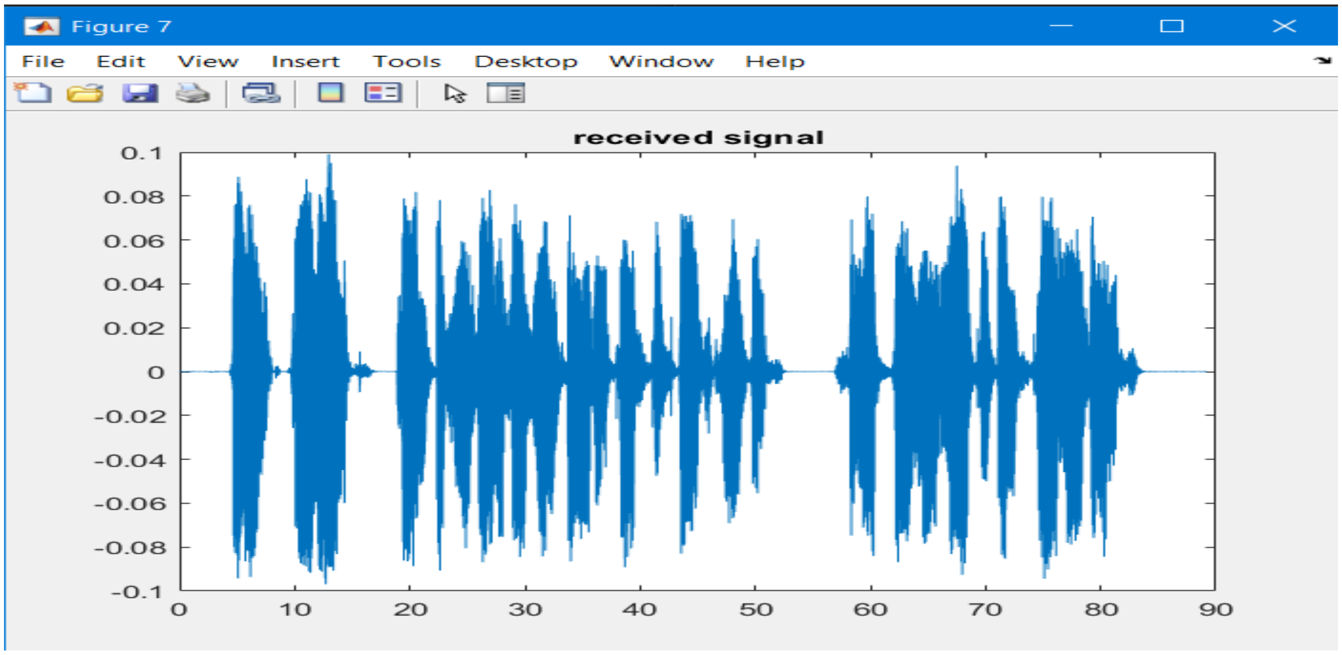
DSB-SC spectrum



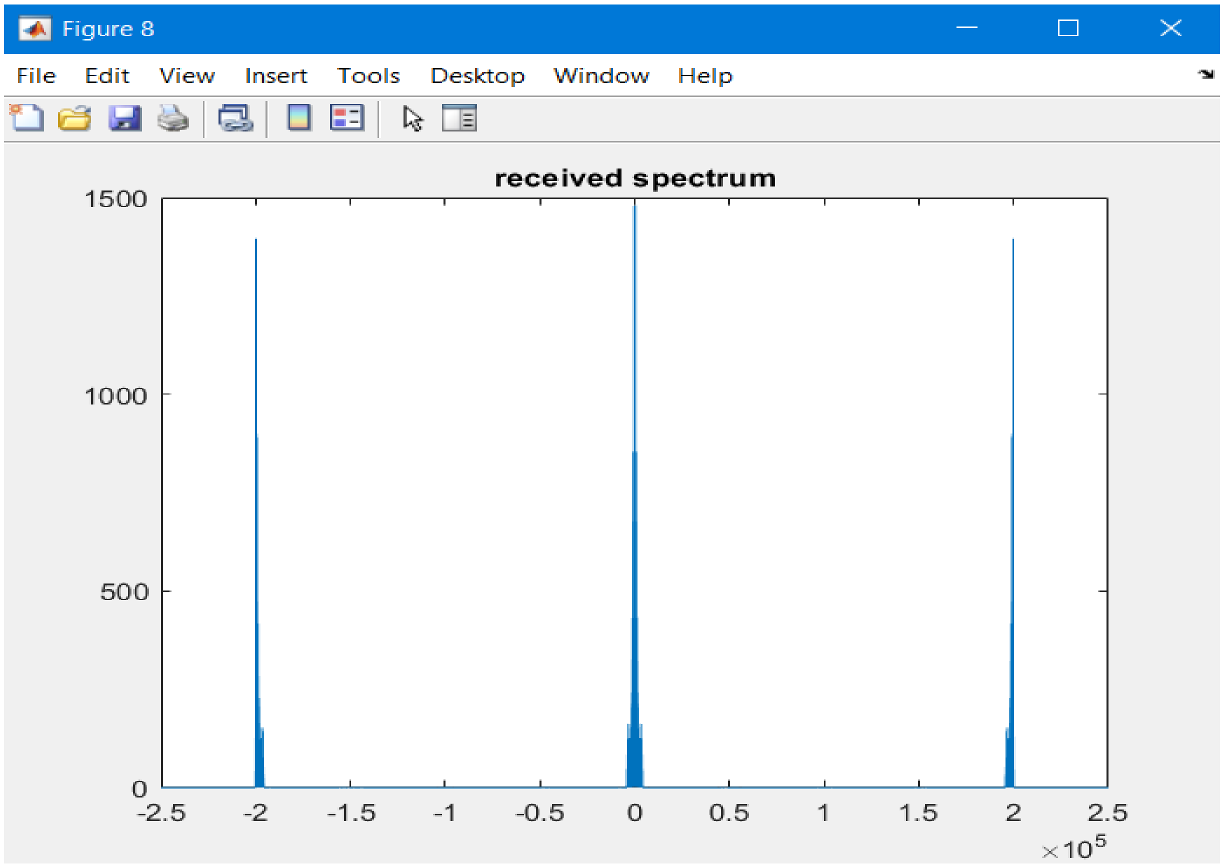
SSB-SC (LSB) spectrum



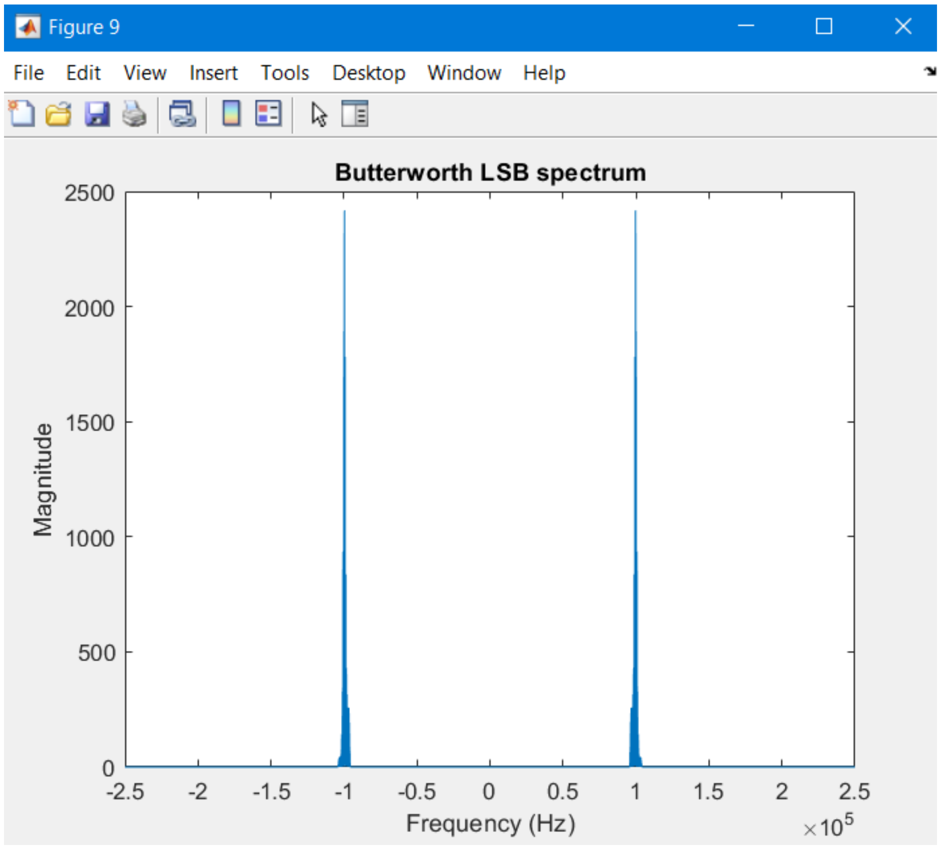
Received signal



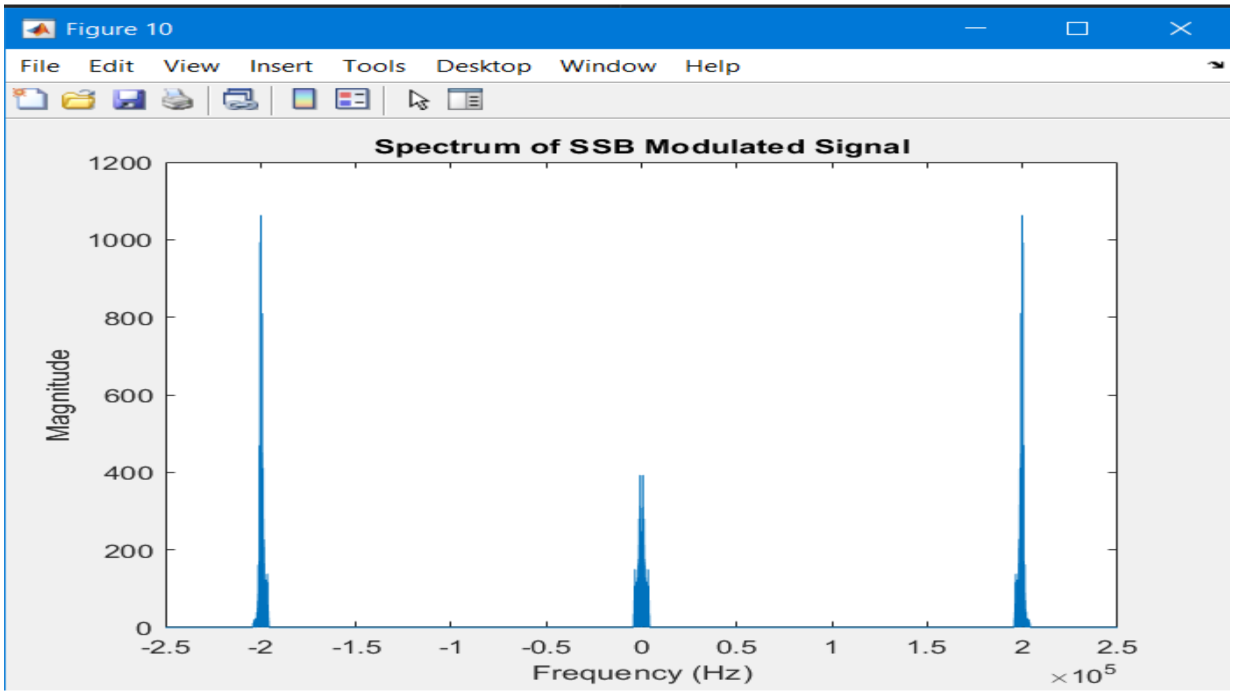
Received spectrum



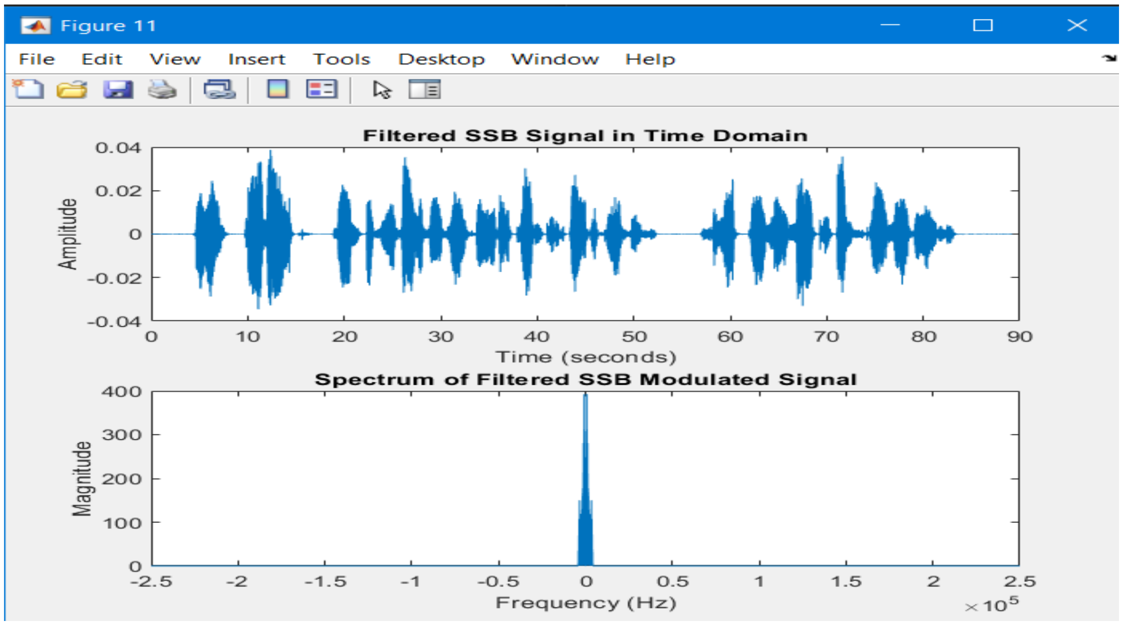
Butterworth LSB spectrum



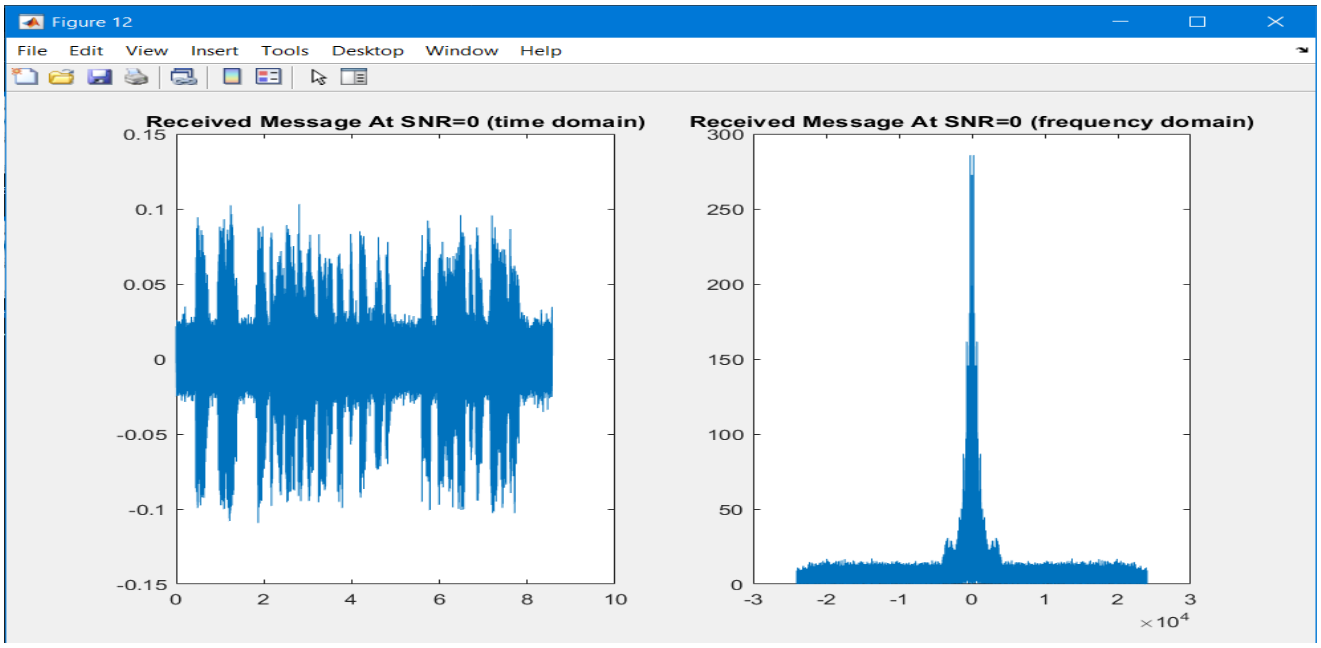
Spectrum of SSB modulated signal



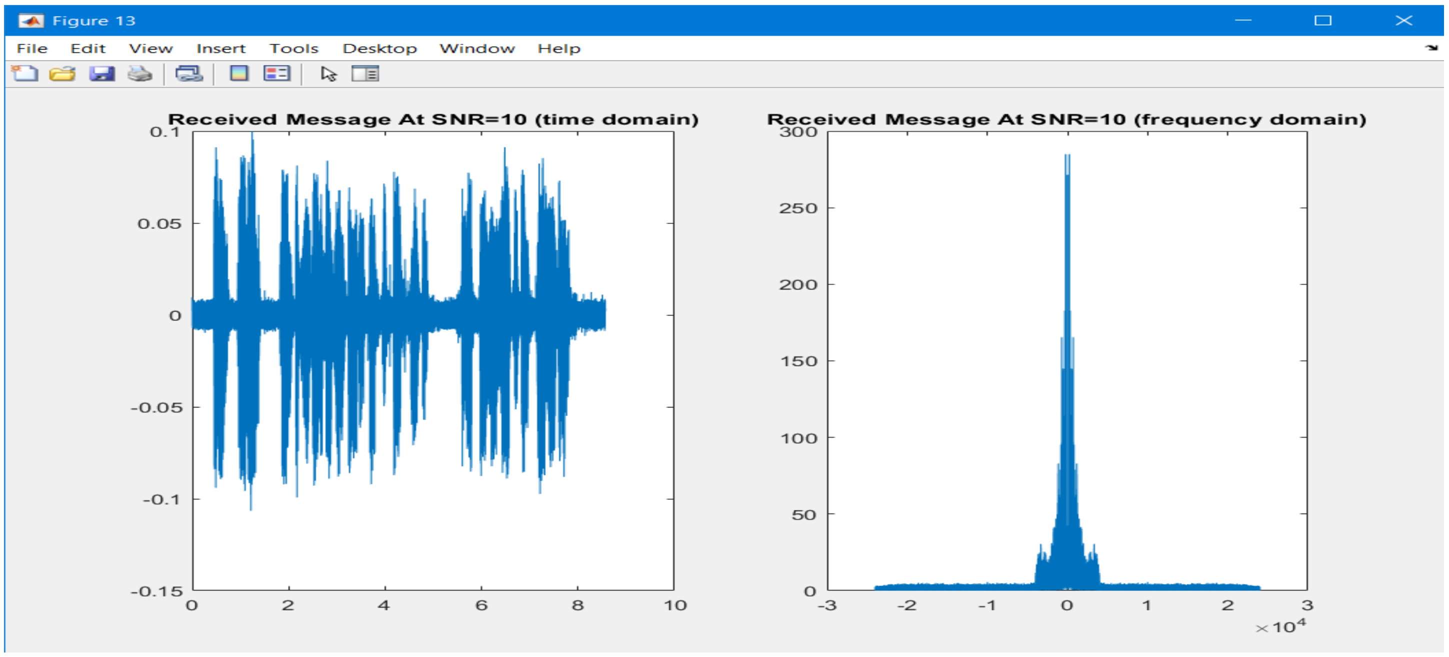
Filtered SSB Signal in Time Domain & Spectrum of filtered SSB Modulated signal



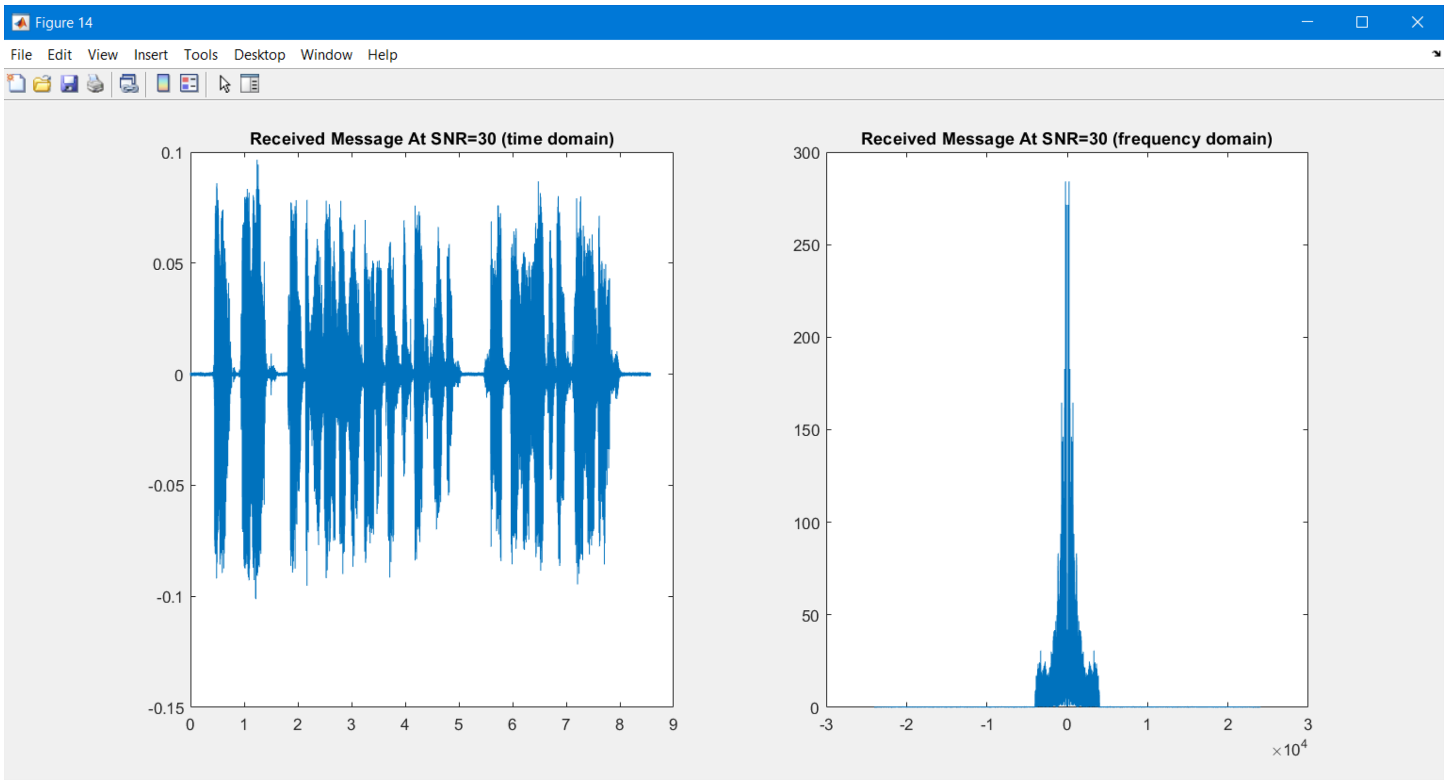
Received Message At SNR=0 (time domain) & Received Message At SNR=0 (frequency domain)



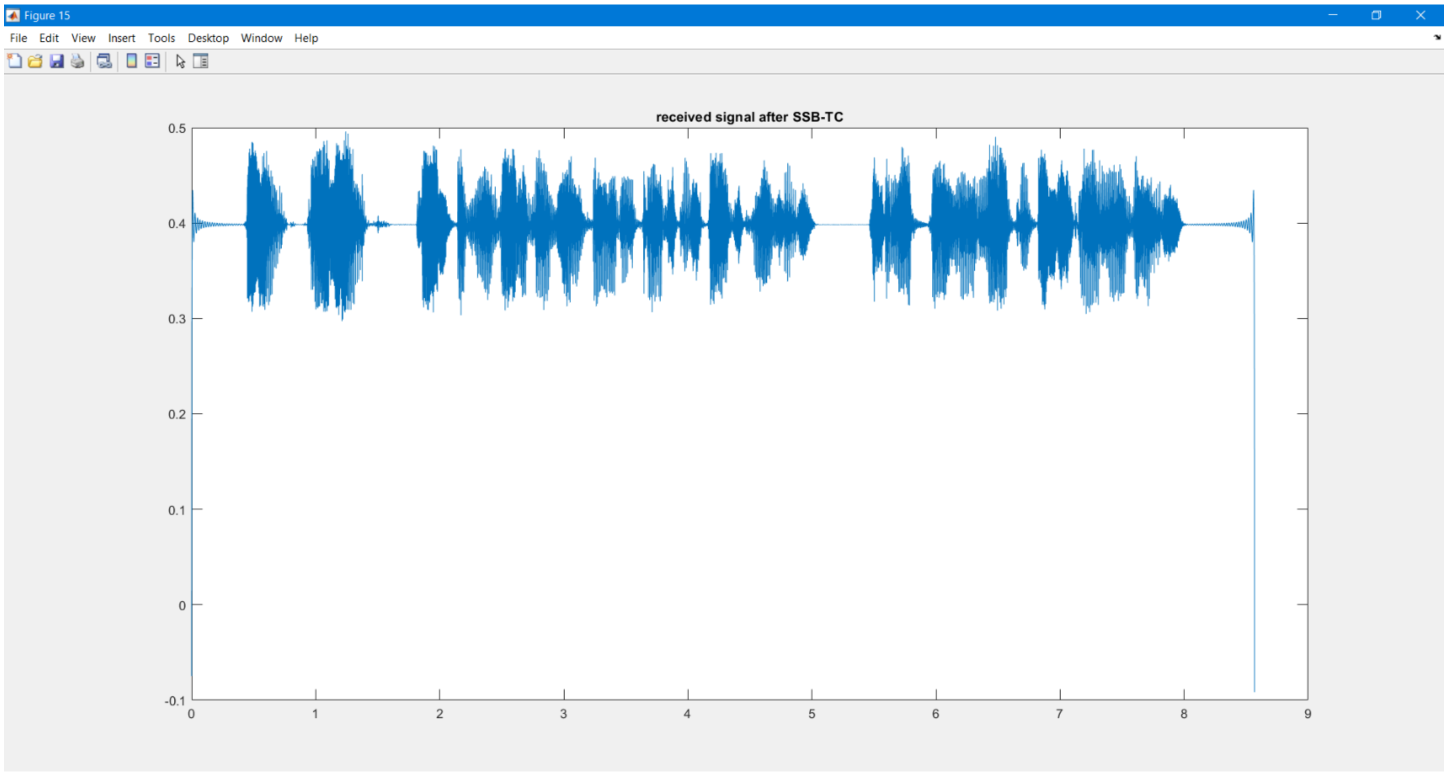
Received Message At SNR=10 (time domain) & Received Message At SNR=10 (frequency domain)



Received Message At SNR=30 (time domain) & Received Message At SNR=30 (frequency domain)



received signal after SSB-TC



received spectrum after SSB-TC

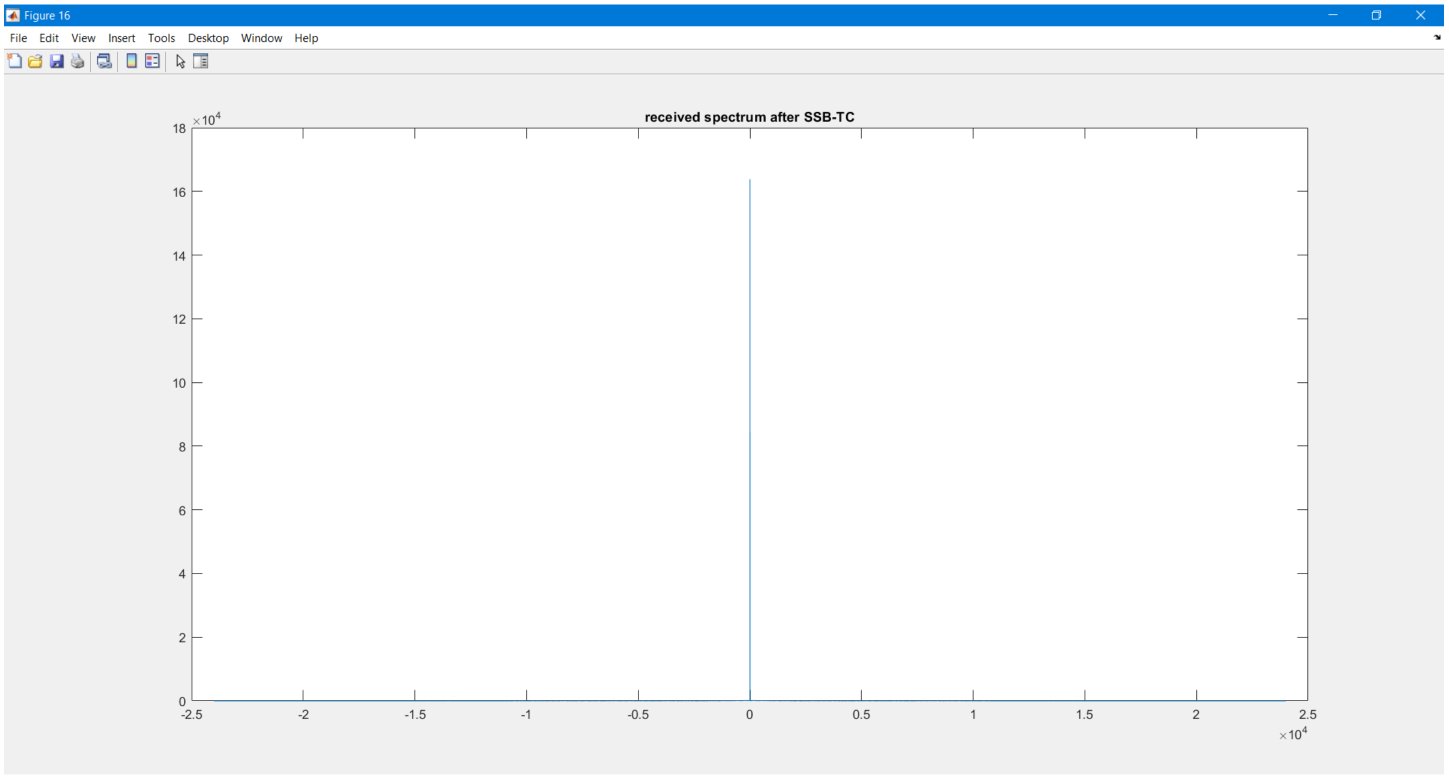


Figure 16
File Edit View Insert Tools Desktop Window Help
received spectrum after SSB-TC
×10^4
×10^4