Nile University

# ECEN314: Fundamentals of Communications

**Course Project**

Omar Samir Mohamed (211000372)

Mahmoud Reda Abdel Rauf (211000489)

Marwan Mohamed Zaki (211001365)

Abdullah El Khateeb (211001730)

Nile University

ECEN314

Spring 2024

**1.**

2. **Introduction of SSB** The Single Sideband Suppressed Carrier (SSB-SC) modulation technique is a method used in communication systems to reduce bandwidth consumption while maintaining signal integrity. This report presents a detailed analysis and implementation of an SSB-SC modulator, including the theoretical background, mathematical representation, and practical implementation steps.
3. **Theoretical Background of SSB** SSB-SC modulation eliminates one of the sidebands along with the carrier signal, effectively halving the bandwidth required for transmission compared to Double Sideband Suppressed Carrier (DSB-SC) modulation. The Hilbert transform operation plays a crucial role in generating SSB signals by introducing a phase shift of $\pm\ \pi\pi/2$ for positive and negative frequencies, respectively. The mathematical representation of an SSB signal is given by:

$uu(tt)$=$AA_{cc}$m($tt$) cos(2$\pi\pi ff_{cc}tt$) $\mp$ $AA_{cc}mm(tt)$ sin(2$\pi\pi ff_{cc}tt$)

where $mm(tt)$ is the Hilbert transform of the message signal, and the choice of the plus or minus sign determines the sideband (upper or lower).

4. **Introduction of QAM** Quadrature Amplitude Modulation (QAM) is a modulation technique widely used in communication systems for transmitting digital data over analog channels. This report provides a detailed analysis and implementation of QAM modulation, focusing on the theoretical background, mathematical representation, and practical implementation steps.
5. **Theoretical Background of QAM** QAM modulation combines both amplitude and phase modulation, allowing for the transmission of multiple bits per symbol. Unlike Single Sideband Suppressed Carrier (SSB-SC) modulation, which eliminates one sideband and the carrier signal, QAM utilizes both sidebands and the carrier. The mathematical representation of a QAM signal can be expressed as:

$uu(tt)$= $AA_{cc}\cdot mm(tt)\cdot$cos(2$\pi\pi ff_{cc}$t+$\phi\phi$)

represents the message signal, $AA_{cc}$ is the carrier amplitude, $ff_{cc}$ is the carrier frequency, and $\phi\phi$ is the phase offset.

## 6. Implementation of SSB Modulation

a. **SSB Modulator Design**: The SSB modulator is implemented using the Hilbert transform operation to generate the upper and lower sidebands.

b. **Modulation Process**: One of the audio signals is modulated into the upper sideband, while the other is modulated into the lower sideband of a carrier signal with a frequency of 100 kHz.

c. **SSB Demodulator Design**: An SSB demodulator is implemented to recover the original audio signals from the modulated SSB signal.

**6. Implementation of QAM Modulation**

a. **QAM Modulator Design**: A Quadrature Amplitude Modulation (QAM) modulator is implemented to modulate the audio signals into the I and Q channels.

b. **Modulation Process**: One audio signal is modulated into the I-channel, and the other into the Q-channel of a carrier signal with a frequency of 100 kHz.

c. **QAM Demodulator Design**: A QAM demodulator is implemented to recover the original audio signals from the modulated QAM signal.

**7. Effects of Phase and Frequency Errors** Phase and frequency errors between the transmitter and receiver local oscillators can distort demodulated signals in both SSB-SC and QAM modulation systems. In SSB-SC, phase errors cause shifts, while frequency errors introduce frequency offsets. In QAM, phase errors rotate constellation points, leading to symbol errors, while frequency errors induce inter-symbol interference. Ensuring accurate synchronization between oscillators is vital to minimize these effects.

**Conclusion** This report has provided a thorough examination of the SSB-SC modulation technique, including its theoretical foundations, implementation steps for modulation and demodulation, and comparison with other modulation techniques. The analysis of phase and frequency errors on demodulated signals highlights the importance of precise synchronization in communication systems. Overall, the report serves as a comprehensive guide for understanding and implementing SSB-SC modulation in communication systems.

Roles :

SSB Implementation for Marwan & Mahmoud

QAM Implementation for Omar & Abdullah

Report for All of us

The Code :

1)SSB

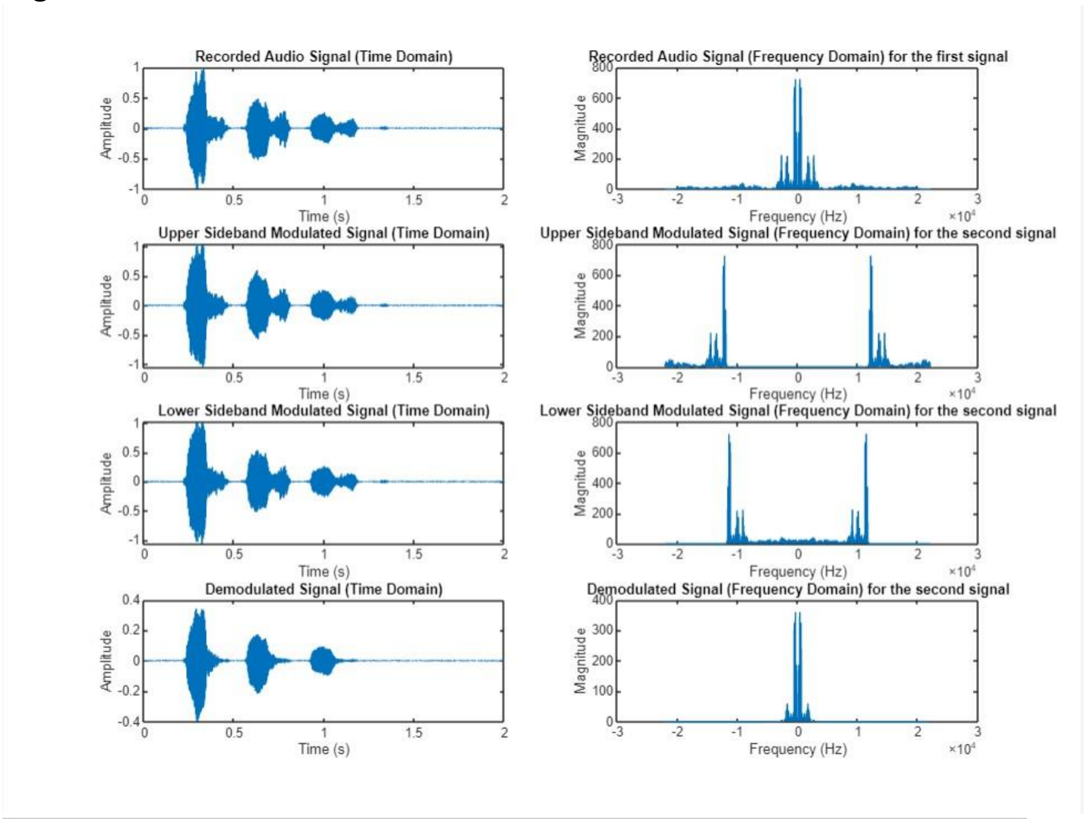
```
% Define recording parameters
recording_duration = 2; % Duration of recording in seconds
Fs = 44100; % Sampling frequency
num_bits = 16; % Bit depth num_channels
= 1; % Mono recording
% Initialize recording
recObj = audiorecorder(Fs, num_bits, num_channels);
% Record audio disp('Start
speaking.');
recordblocking(recObj, recording_duration);
disp('End of recording.'); % Retrieve
recorded audio data m1 =
getaudiodata(recObj);
% Play back the original recorded audio disp('Playing
original recorded audio...'); sound(m1, Fs);
% Initialize recording for the second signal recObj2
= audiorecorder(Fs, num_bits, num_channels);
% Record audio for the second signal disp('Start
speaking for the second signal.');
recordblocking(recObj2, recording_duration);
disp('End of recording for the second signal.'); %
Retrieve recorded audio data for the second signal
m2 = getaudiodata(recObj2);
% Play back the original recorded audio for the second signal disp('Playing
original recorded audio for the second signal...'); sound(m2, Fs);

% Normalize recorded audio signal m1
= m1 / max(abs(m1));

% Perform Hilbert transform on the recorded message signal m1_hilbert
= imag(hilbert(m1));

% Define carrier parameters Ac = 1; %
Carrier amplitude fc = 100e3; % Carrier
frequency t = (0:length(m1)-1) / Fs; %
Time vector

% Modulate recorded signal into upper sideband
usb = Ac * m1 .* cos(2*pi*fc*t.') - Ac * m1_hilbert .* sin(2*pi*fc*t.'); % Upper
sideband
%Modulate recorded signal into lower sideband
lsb = Ac * m1 .* cos(2*pi*fc*t.') + Ac * m1_hilbert .* sin(2*pi*fc*t.');
%SSB Demodulation of the signal
v=usb.*Ac.*cos(2*pi*fc*t.');
demodulation=lowpass(v,40,Fs);

% Normalize recorded audio signal for the second signal m2
= m2 / max(abs(m2));
```

```
% Perform Hilbert transform on the recorded message signal for the second signal
m2_hilbert = imag(hilbert(m2));

% Modulate recorded signal into upper sideband for the second signal usb2 = Ac *
m2 .* cos(2*pi*fc*t.') - Ac * m2_hilbert .* sin(2*pi*fc*t.'); % Upper sideband
for the second signal
% Modulate recorded signal into lower sideband for the second signal lsb2
= Ac * m2 .* cos(2*pi*fc*t.') + Ac * m2_hilbert .* sin(2*pi*fc*t.');
% SSB Demodulation of the second signal
v2=usb2.*Ac.*cos(2*pi*fc*t.');
demodulation2=lowpass(v2,40,Fs);
 figure;

% Plot recorded audio signal for the first signal in the time domain
subplot(4,2,1); plot(t, m1);
title('Recorded Audio Signal (Time Domain)');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot recorded audio signal for the first signal in the frequency domain
subplot(4,2,2); m1_fft = fftshift(fft(m1));
frequencies = linspace(-Fs/2, Fs/2, length(m1));
plot(frequencies, abs(m1_fft));
title('Recorded Audio Signal (Frequency Domain) for the first signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot upper sideband modulated signal for the first signal in the time domain
subplot(4,2,3); plot(t, usb);
title('Upper Sideband Modulated Signal (Time Domain)');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot upper sideband modulated signal for the first signal in the frequency
domain subplot(4,2,4);
usb_fft = fftshift(fft(usb)); plot(frequencies,
abs(usb_fft));
title('Upper Sideband Modulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot lower sideband modulated signal for the first signal in the time domain
subplot(4,2,5); plot(t, lsb);
title('Lower Sideband Modulated Signal (Time Domain)');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot lower sideband modulated signal for the first signal in the frequency
domain subplot(4,2,6);
lsb_fft = fftshift(fft(lsb)); plot(frequencies,
abs(lsb_fft));
title('Lower Sideband Modulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot demodulated signal for the first signal in the time domain
subplot(4,2,7); plot(t, demodulation);
title('Demodulated Signal (Time Domain)');
xlabel('Time (s)'); ylabel('Amplitude');
```

```
% Plot demodulated signal for the first signal in the frequency domain
subplot(4,2,8);
demodulation_fft = fftshift(fft(demodulation));
plot(frequencies, abs(demodulation_fft));
title('Demodulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');
% Write upper sideband modulated signal to an audio file
%audiowrite('usb_signal.wav', usb, Fs);|

figure;
% Plot recorded audio signal for the second signal in the time domain
subplot(4,2,1); plot(t, m2);
title('Recorded Audio Signal (Time Domain) for the second signal');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot recorded audio signal for the second signal in the frequency domain
subplot(4,2,2); m2_fft = fftshift(fft(m2));
frequenciesm2 = linspace(-Fs/2, Fs/2, length(m2));
plot(frequenciesm2, abs(m2_fft));
title('Recorded Audio Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot upper sideband modulated signal for the second signal
subplot(4,2,3); plot(t, usb2);
title('Upper Sideband Modulated Signal (Time Domain) for the second signal');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot upper sideband modulated signal for the second signal in the frequency
domain subplot(4,2,4);
usb2_fft = fftshift(fft(usb2)); plot(frequenciesm2,
abs(usb2_fft));
title('Upper Sideband Modulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot lower sideband modulated signal for the second signal
subplot(4,2,5); plot(t, lsb2);
title('Lower Sideband Modulated Signal (Time Domain) for the second signal');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot lower sideband modulated signal for the second signal in the frequency
domain subplot(4,2,6);
lsb2_fft = fftshift(fft(lsb2)); plot(frequenciesm2,
abs(lsb2_fft));
title('Lower Sideband Modulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

% Plot demodulated signal for the second signal
subplot(4,2,7); plot(t, demodulation2);
title('Demodulated Signal (Time Domain) for the second signal');
xlabel('Time (s)'); ylabel('Amplitude');

% Plot demodulated signal for the second signal in the frequency domain
subplot(4,2,8);
demodulation2_fft = fftshift(fft(demodulation2));
plot(frequenciesm2, abs(demodulation2_fft));
```

```
title('Demodulated Signal (Frequency Domain) for the second signal');
xlabel('Frequency (Hz)'); ylabel('Magnitude');
% Write upper sideband modulated signal to an audio file for the second signal
%audiowrite('usb_signal_2.wav', usb2, Fs);
```

Figures :



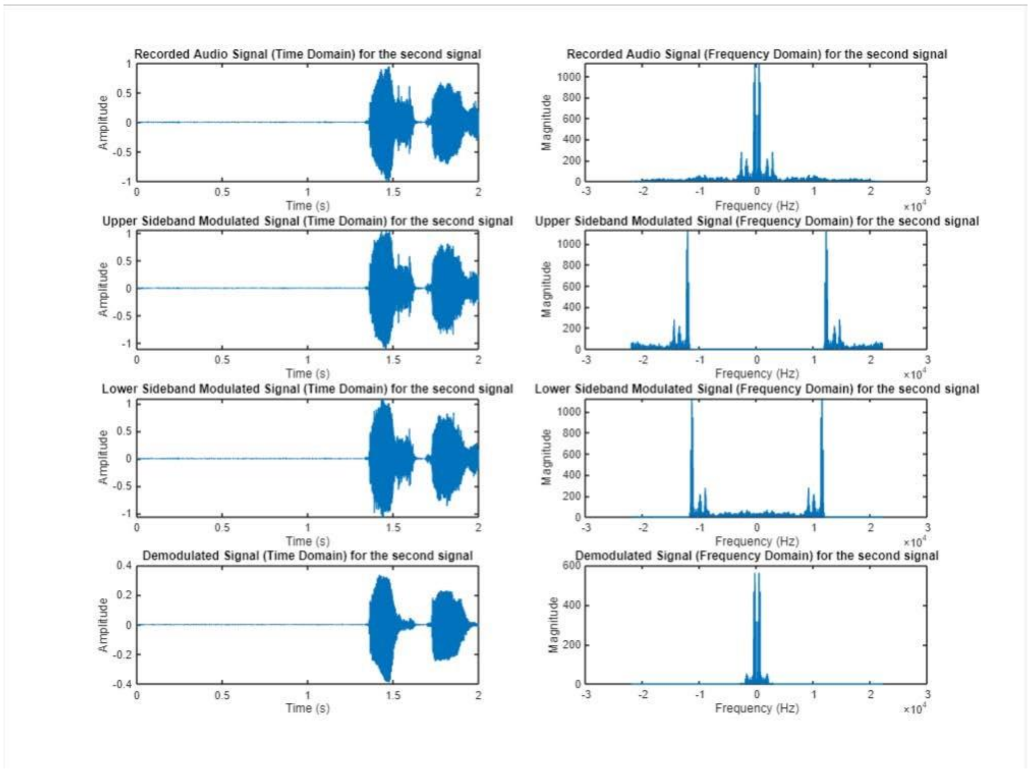


**Figure 1** represents the recorded audio signal for the first signal in the SSB system. It displays the amplitude of the recorded audio signal in the time domain before any modulation or processing related to QAM modulation is applied.

## 2) QAM

```
% Define recording parameters
recording_duration = 5; % Duration of recording in seconds
Fs = 44100; % Sampling frequency
num_bits = 16; % Bit depth num_channels
= 1; % Mono recording

% Initialize recording for first voice signal recObj1
= audiorecorder(Fs, num_bits, num_channels);

% Record the first voice signal
disp('Start speaking for the first voice signal.');
recordblocking(recObj1, recording_duration); disp('End
of recording for the first voice signal.');

% Retrieve recorded audio data for the first voice signal
input_signal1 = getaudiodata(recObj1);

% Normalize input signal for the first voice signal input_signal1
= input_signal1 / max(abs(input_signal1));

% Initialize recording for second voice signal recObj2
= audiorecorder(Fs, num_bits, num_channels);

% Record the second voice signal
disp('Start speaking for the second voice signal.');
recordblocking(recObj2, recording_duration); disp('End
of recording for the second voice signal.');

% Retrieve recorded audio data for the second voice signal input_signal2
= getaudiodata(recObj2);

% Normalize input signal for the second voice signal input_signal2
= input_signal2 / max(abs(input_signal2));
 %
Parameters
fc = 100e3; % Carrier frequency M = 16;
% Number of constellation points

% Determine the length of the longer input signal
N = max(length(input_signal1), length(input_signal2));

% Time vector based on the longer input signal t
= (0:N-1) / Fs;
```

```
% Generate QAM constellation points
constellation_points = zeros(1, M); for
k = 1:M
    constellation_points(k) = exp(1i*(2*pi/M)*(k-1)); end

% Combine two voice signals
input_signal = input_signal1 + 1i * input_signal2;

% Modulate the combined voice signals qam_signal
= input_signal .* exp(1i*2*pi*fc*t.');

% Demodulate the modulated signal
demodulated_signal = qam_signal .* exp(-1i*2*pi*fc*t.');

% Separate the demodulated signals received_signal1
= real(demodulated_signal); received_signal2 =
imag(demodulated_signal);

% Plot original, modulated, and demodulated signals
figure; subplot(2,1,1); plot(t,
real(input_signal1)); title('Original Audio Signal
1'); xlabel('Time (s)'); ylabel('Amplitude');
 subplot(2,1,2); plot(t,
real(input_signal2));
title('Original Audio Signal 2');
xlabel('Time (s)');
ylabel('Amplitude');
 figure; subplot(1,1,1);
plot(t, real(qam_signal));
title('Modulated Audio Signal (Combined)');
xlabel('Time (s)'); ylabel('Amplitude');
 figure;
subplot(2,1,1);
plot(t, real(received_signal1)); title('Demodulated
Audio Signal 1');
xlabel('Time (s)'); ylabel('Amplitude');
 subplot(2,1,2);
plot(t, real(received_signal2)); title('Demodulated
Audio Signal 2');
xlabel('Time (s)'); ylabel('Amplitude');

% Play original, modulated, and demodulated signals disp('Playing
original audio signal1...'); sound(real(input_signal1), Fs);
pause(length(input_signal1)/Fs + 1); % Pause to ensure first original audio
playback completes before playing the second one disp('Playing original
audio signal2...'); sound(real(input_signal2), Fs);
pause(length(input_signal2)/Fs + 1); % Pause to ensure second original audio
playback completes before playing modulated signal disp('Playing modulated
audio signal...'); sound(real(qam_signal), Fs);
pause(length(qam_signal)/Fs + 1); % Pause to ensure modulated audio playback
completes before playing demodulated signals disp('Playing demodulated audio
signal1...');
sound(received_signal1, Fs);
pause(length(received_signal1)/Fs + 1);
disp('Playing demodulated audio signal2...');
```

```
sound(received_signal2, Fs);
```

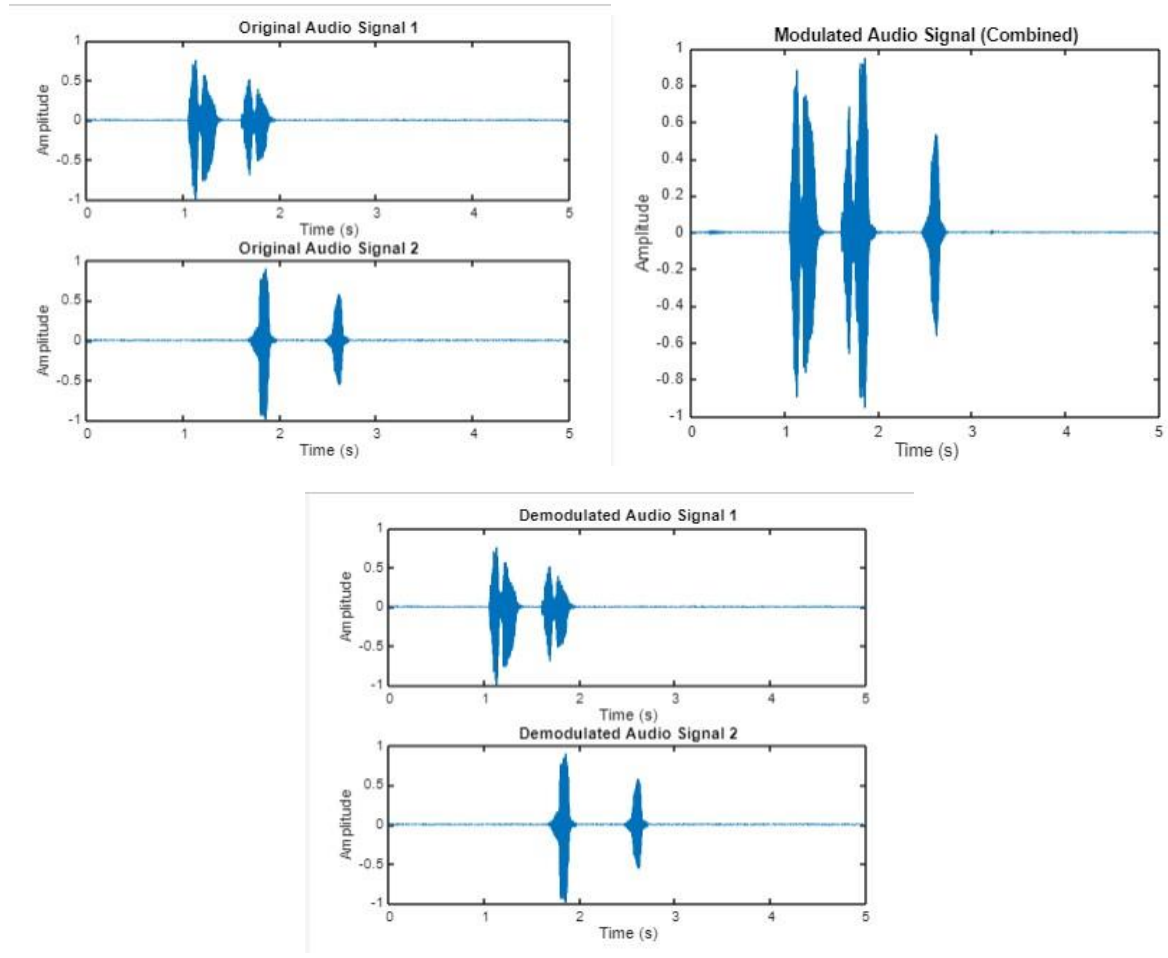


**Figure 2** illustrate the progression of Quadrature Amplitude Modulation (QAM) applied to two input voice signals. They show the original audio signals, the modulated combined signal, and the demodulated signals for each input, providing insight into the modulation and demodulation processes in QAM communication systems.

## Phase 2

# Introduction

**Introduction to Double-Sideband Transmitted Carrier (DSBTC)** and Frequency Modulation (FM)
In the field of analog communication systems, modulation techniques are essential for transmitting information over various media. Two prominent methods are Double-Sideband Transmitted Carrier (DSBTC) and Frequency Modulation (FM). Each technique has unique characteristics and applications that make it suitable for specific communication needs.

**Double-Sideband Transmitted Carrier (DSBTC)**
DSBTC is a type of amplitude modulation (AM) where both the upper and lower sidebands, along with the carrier signal, are transmitted. This method ensures the carrier's presence in the signal, simplifying the demodulation process at the receiver. While DSBTC is straightforward and reliable, it requires more bandwidth and power compared to other AM techniques, such as Single-Sideband (SSB) modulation.

**Frequency Modulation (FM)**
FM is a technique where the frequency of the carrier wave varies in accordance with the amplitude of the message signal. Unlike AM, FM maintains a constant amplitude and instead modulates the carrier's frequency. This method offers superior noise immunity and is widely used for high-fidelity audio broadcasting and communication systems. FM signals occupy a larger bandwidth but provide better signal integrity and quality, making them ideal for applications like FM radio and television sound transmission.

Understanding DSBTC and FM modulation is crucial for designing and optimizing communication systems to meet various application requirements. Each technique offers distinct advantages, with DSBTC being valued for its demodulation simplicity and FM for its robustness against noise and ability to deliver high-quality audio.

## 1)DSBTC Code

```
% Parameters
fs = 1000;  % Sampling frequency
fm = 10;  % Frequency of the message signal
fc = 100;  % Frequency of the carrier signal
Am = 1;  % Amplitude of the message signal
Ac = 1;  % Amplitude of the carrier signal
t = linspace(0, 2, 2*fs);  % Time vector

% Generate message signal
mt = Am * cos(2*pi*fm*t);

% Generate carrier signal
ct = Ac * cos(2*pi*fc*t);

% Signal power
signal_power = ((Ac^2)/2) * (((Am^2)/2) + k^2);

% Generate noise
SNR_1 = 0;  % SNR in dB
noise_power = signal_power / (10^(SNR_1/10));
noise = sqrt(noise_power) * randn(size(mt));

% Add noise to the message signal
mt_noisy = mt + noise .* sqrt(noise_power);

% Amplitude Modulation
k=Ac;
s=(k+mt).*ct;
s_noise= s + noise .* sqrt(noise_power);
```

```matlab
% Amplitude Demodulation
d1 = s .* ct;
d_lpf = lowpass(d1, 20, fs, 'Steepness', 0.95);
d2 = d_lpf/((Ac)^2/2);
d=d2-k;

d1_noise = s_noise .* ct;
d_lpf_noise = lowpass(d1_noise, 20, fs, 'Steepness', 0.95);
d2_noise = d_lpf_noise/((Ac)^2/2);
d_noise=d2_noise-k;

% Frequency Domain
f = linspace(-fs/2, fs/2, 2*fs);

% Perform FFT
Mf = fftshift(fft(mt)) / fs;
Mf_NOISY = fftshift(fft(mt_noisy)) / fs;
S = fftshift(fft(s)) / fs;
S_NOISE = fftshift(fft(s_noise)) / fs;
D = fftshift(fft(d)) / fs;
D_NOISE = fftshift(fft(d_noise)) / fs;

% Plotting
figure;

subplot(7,1,1);
plot(t, mt);
title('Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,2);
plot(t, noise);
title('White Noise');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,3);
plot(t, mt_noisy);
title('Message Signal with White Noise');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,4);
plot(t, s);
title('Modulated Signal Without Noise');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,5);
plot(t, s_noise);
title('Modulated Signal With Noise');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,6);
plot(t, d);
title('Demodulated Signal');
xlabel('Time (s)');
```

```matlab
ylabel('Amplitude');

subplot(7,1,7);
plot(t, d_noise);
title('Demodulated Signal With Noise');
xlabel('Time (s)');
ylabel('Amplitude');

%frequency plotting

figure;

subplot(7,1,1);
plot(f, abs(Mf));
title('Message Signal (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,2);
plot(f, abs(noise));
title('White Noise (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,3);
plot(f, abs(Mf_NOISY));
title('Message Signal with White Noise (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,4);
plot(f, abs(S));
title('Modulated Signal Without Noise (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,5);
plot(f, abs(S_NOISE));
title('Modulated Signal With Noise (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,6);
plot(f, abs(D));
title('Demodulated Signal (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(7,1,7);
plot(f, abs(D_NOISE));
title('Demodulated Signal With Noise (Frequency Domain)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

## AT 0 DB

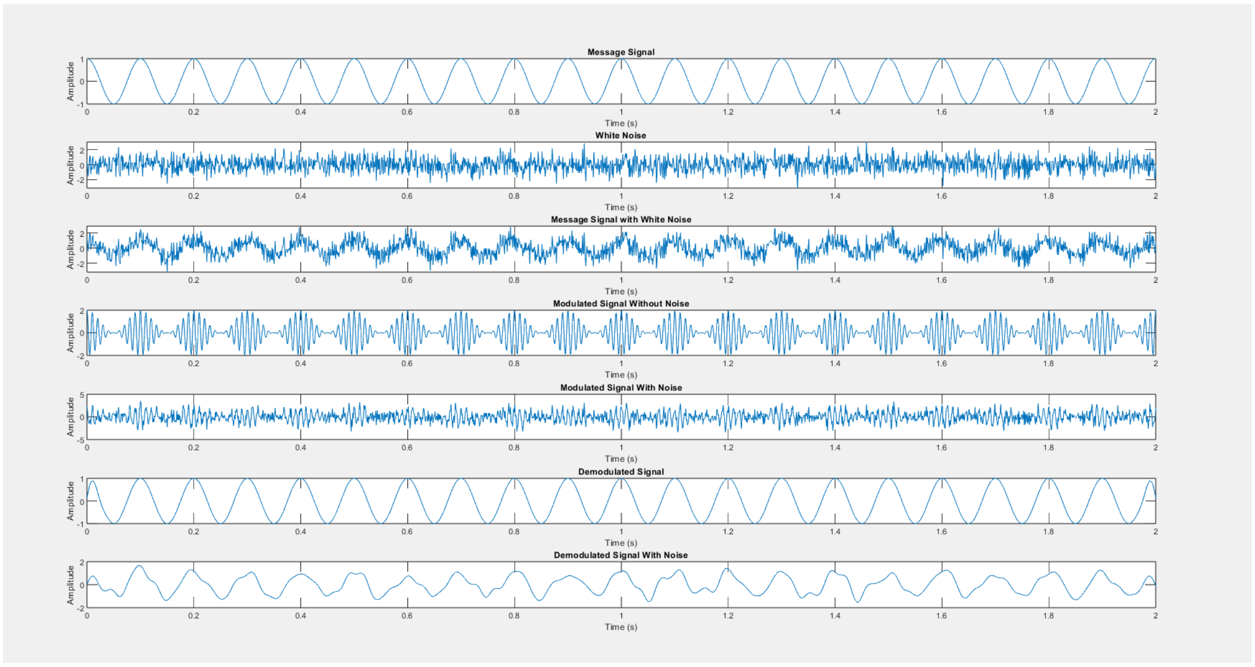


*Figure 1 Time-Domain Analysis with Built-in Functions, SNR = 0 dB*

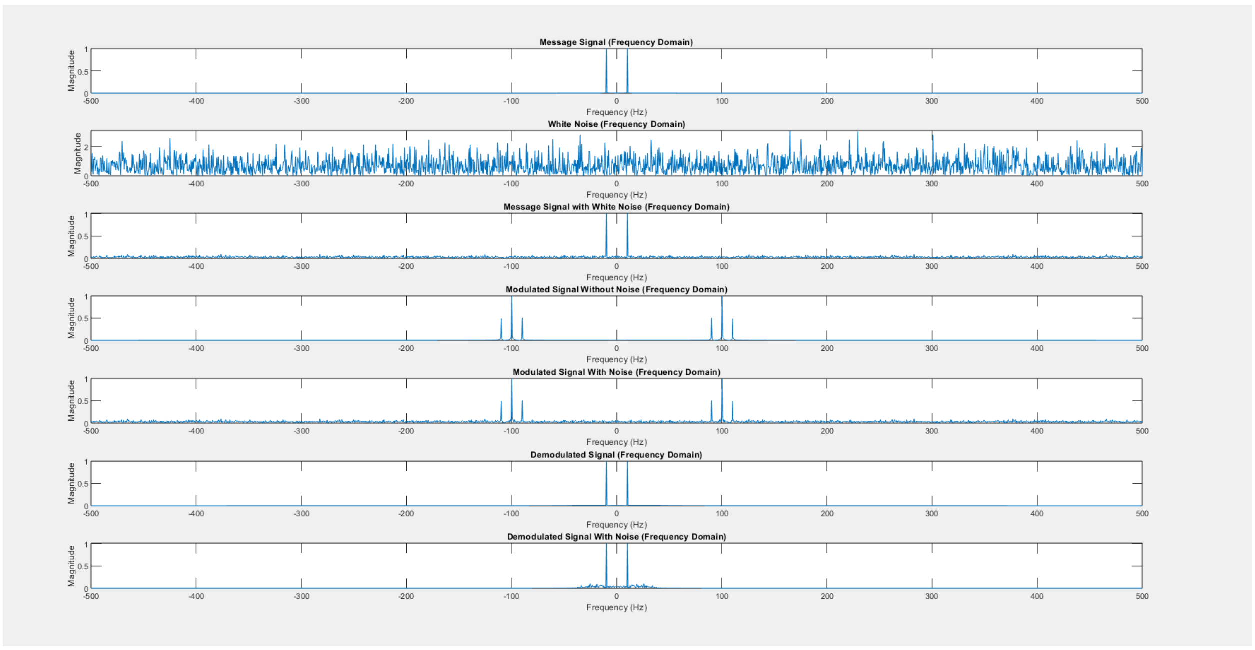


*Figure 2Frequency-Domain Analysis with Built-in Functions, SNR = 0 dB*

## AT 10 DB

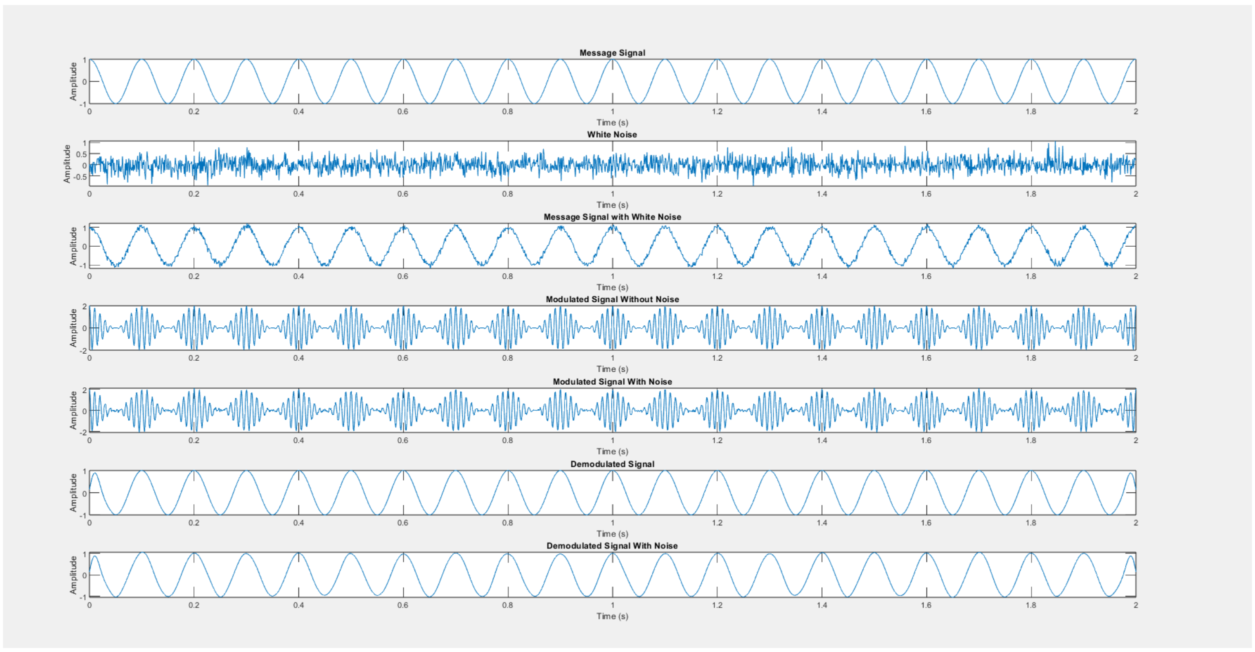


*Figure 3Time-Domain Analysis with Built-in Functions, SNR = 10 dB*

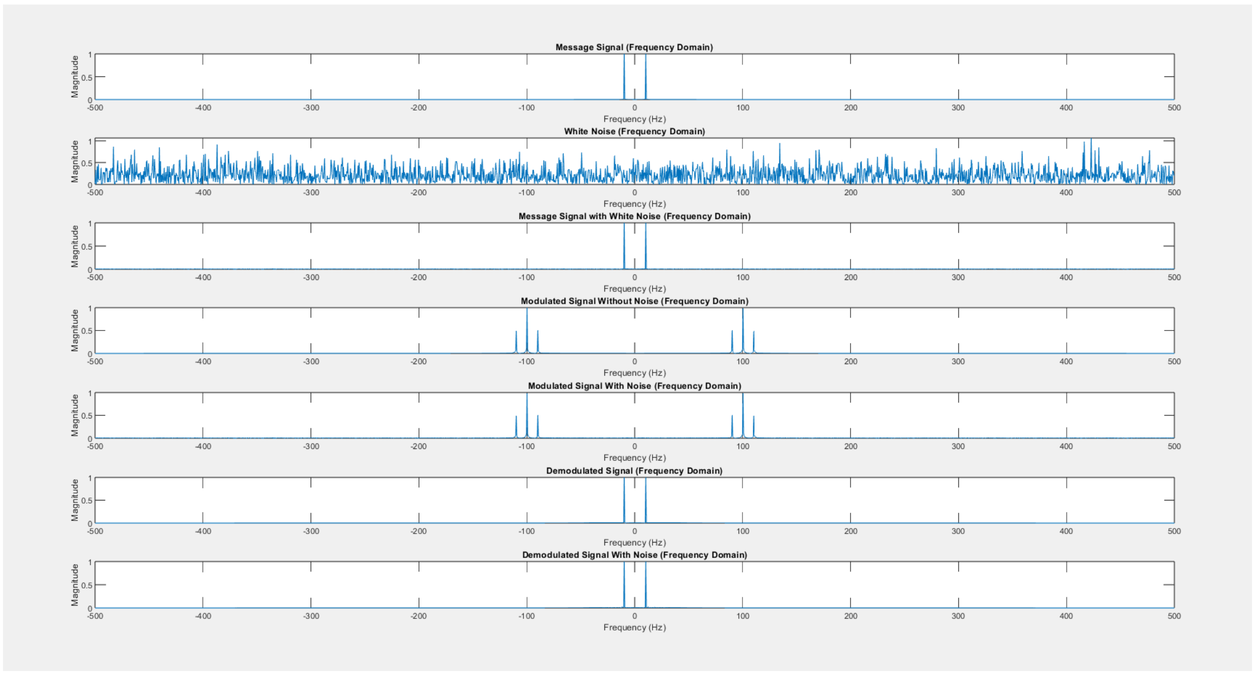


*Figure 4Frequency-Domain Analysis with Built-in Functions, SNR = 10 dB*

## AT 20 DB

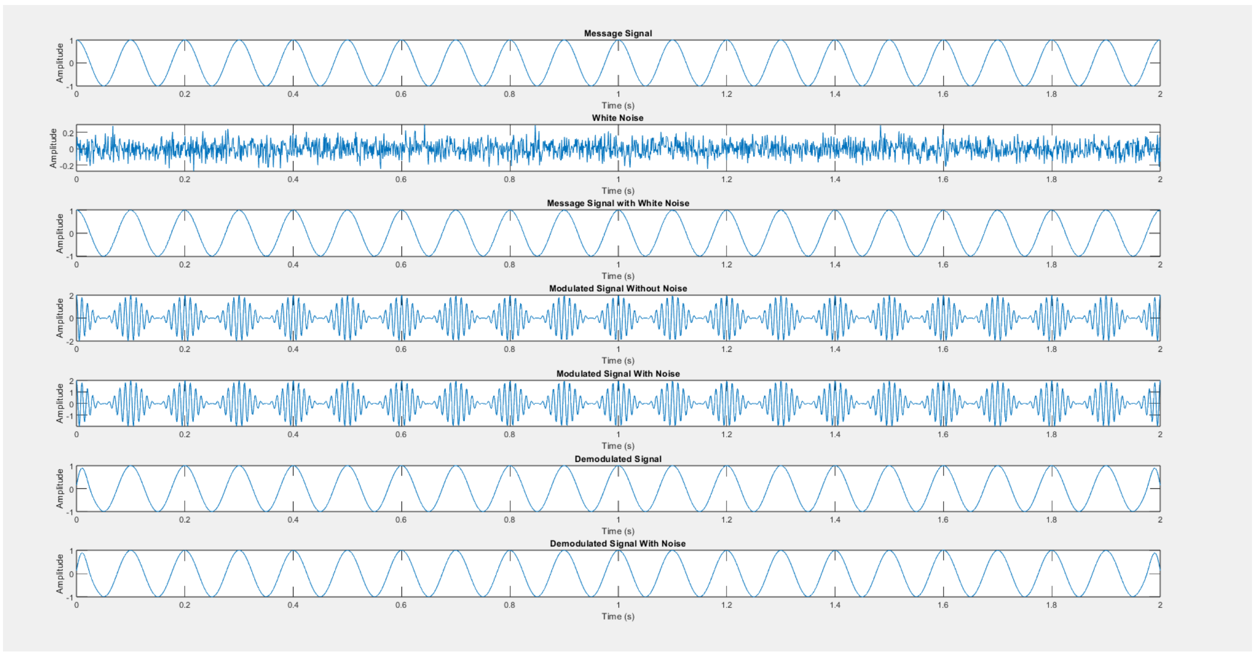


*Figure 5Figure 1 Time-Domain Analysis with Built-in Functions, SNR =20 dB*

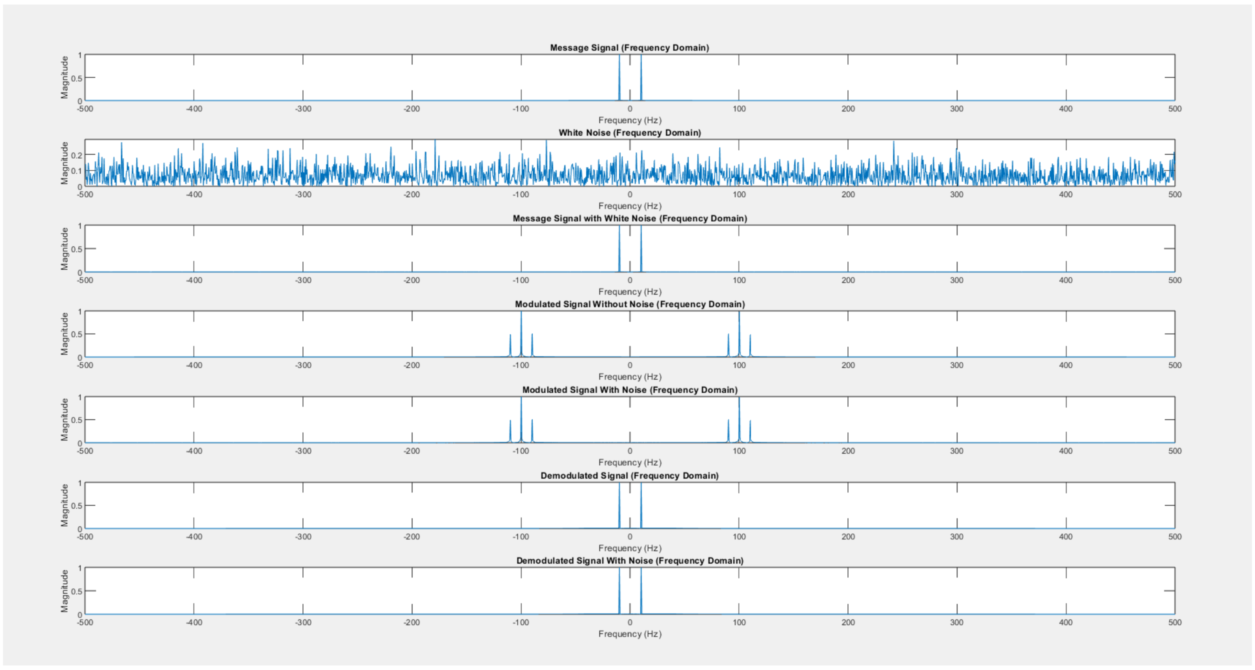


*Figure 6 Frequency-Domain Analysis with Built-in Functions, SNR = 20 dB*

## Impact of Changing SNR

**SNR = 0 dB**
Noise Power: At 0 dB, the noise power is equal to the signal power. This means that the noise is as strong as the signal itself.
Impact on Signal: The message signal with noise (mt_noisy) will be heavily corrupted by noise, making it difficult to distinguish the original message signal from the noise. The demodulated signal with noise (d_noise) will also be significantly distorted.

**SNR = 10 dB**
Noise Power: At 10 dB, the noise power is one-tenth of the signal power. This means the signal is 10 times stronger than the noise.
Impact on Signal: The message signal with noise will be less corrupted compared to the 0 dB case. The demodulated signal with noise will still have some noise, but the original message signal will be more distinguishable compared to the 0 dB case.

**SNR = 20 dB**
Noise Power: At 20 dB, the noise power is one-hundredth of the signal power. This means the signal is 100 times stronger than the noise.
Impact on Signal: The message signal with noise will be minimally corrupted by noise. The demodulated signal with noise will be very close to the original message signal, with noise being almost negligible.

## 2)FM Code with built-in functions

```
Fs = 5000; % Sampling frequency
fc = 1000; % Carrier frequency
t = (0:1/Fs:0.2)'; % Time vector
x = sin(2*pi*30*t) + 2*sin(2*pi*60*t); % Message signal

fDev = 250; % Frequency deviation

% FM modulation using fmmod
xfm = fmmod(x, fc, Fs, fDev);

% Calculate the instantaneous power of the modulated signal
instantaneous_power = abs(xfm).^2;

% Numerical integration over the available time interval
integrated_power = trapz(t, instantaneous_power);

% Average power over the interval
average_power = integrated_power / (t(end) - t(1));

% Display the result of the average power
disp(['Average power of the FM modulated signal: ', num2str(average_power)]);

% Define SNR values in dB
SNR_values = [10, 20, 30];

% Pre-allocate space for noisy signals
xfm_noisy = zeros(length(xfm), length(SNR_values));

% Add noise based on integrated power
for i = 1:length(SNR_values)
    % Convert SNR from dB to linear scale
    snr_linear = 10^(SNR_values(i)/10);
    % Calculate noise power based on the SNR
    noise_power = average_power / snr_linear;

    % Generate white Gaussian noise with the calculated noise power
    noise = sqrt(noise_power) * randn(size(xfm));

    % Add noise to the FM modulated signal
    xfm_noisy(:, i) = xfm + noise;
end

% FM demodulation of the original modulated signal (without noise)
x_demod = fmdemod(xfm, fc, Fs, fDev);

% Plotting
for i = 1:length(SNR_values)
    figure;

    % Plot Original Signal
    subplot(6, 1, 1);
    plot(t, x, 'c');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Original Signal');
    grid on;

    % Plot FM Modulated Signal without Noise
    subplot(6, 1, 2);
    plot(t, xfm, 'r');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('FM Modulated Signal without Noise');
    grid on;

    % Plot Demodulated Signal without Noise
    subplot(6, 1, 3);
    plot(t, x_demod, 'g');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Demodulated Signal without Noise');
    grid on;

    % Plot Noisy FM Modulated Signal
    subplot(6, 1, 4);
    plot(t, xfm_noisy(:, i), 'm');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title(['Noisy FM Modulated Signal with SNR = ', num2str(SNR_values(i)), '
dB']);
    grid on;

    % FM demodulation of the noisy signal
    x_demod_noisy = fmdemod(xfm_noisy(:, i), fc, Fs, fDev);

    % Plot Demodulated Signal with Noise
    subplot(6, 1, 5);
    plot(t, x_demod_noisy, 'b');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title(['FM Demodulation with Noise (SNR = ', num2str(SNR_values(i)), ' dB)']);
    grid on;

    % Extract and Plot Noise Component
    noise = xfm_noisy(:, i) - xfm;
    subplot(6, 1, 6);
    plot(t, noise, 'k');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Noise Component');
    grid on;

    % Frequency Domain Analysis with Noise
    figure;

    % FFT of the original signal
    N = length(t);
    X = fftshift(fft(x, N));
    X_magnitude = abs(X)/N;
    frequencies = (-N/2:N/2-1)*(Fs/N);

    % FFT of the noisy FM modulated signal
    XFM_noisy = fftshift(fft(xfm_noisy(:, i), N));
    XFM_noisy_magnitude = abs(XFM_noisy)/N;

    % FFT of the demodulated noisy signal
    X_demod_noisy = fftshift(fft(x_demod_noisy, N));
    X_demod_noisy_magnitude = abs(X_demod_noisy)/N;

    subplot(3, 1, 1);
    plot(frequencies, X_magnitude, 'c', 'DisplayName', 'Original');
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('Original Signal Frequency Spectrum');
    legend;
    grid on;

    subplot(3, 1, 2);
    plot(frequencies, XFM_noisy_magnitude, 'm', 'DisplayName', ['FM Modulated with
Noise SNR = ', num2str(SNR_values(i)), ' dB']);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title(['Noisy FM Modulated Signal Frequency Spectrum (SNR = ',
num2str(SNR_values(i)), ' dB)']);
    legend;
    grid on;

    subplot(3, 1, 3);
    plot(frequencies, X_demod_noisy_magnitude, 'b', 'DisplayName', ['Demodulated
Signal with Noise SNR = ', num2str(SNR_values(i)), ' dB']);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title(['Demodulated Signal Frequency Spectrum (SNR = ',
num2str(SNR_values(i)), ' dB)']);
    legend;
    grid on;
end
```

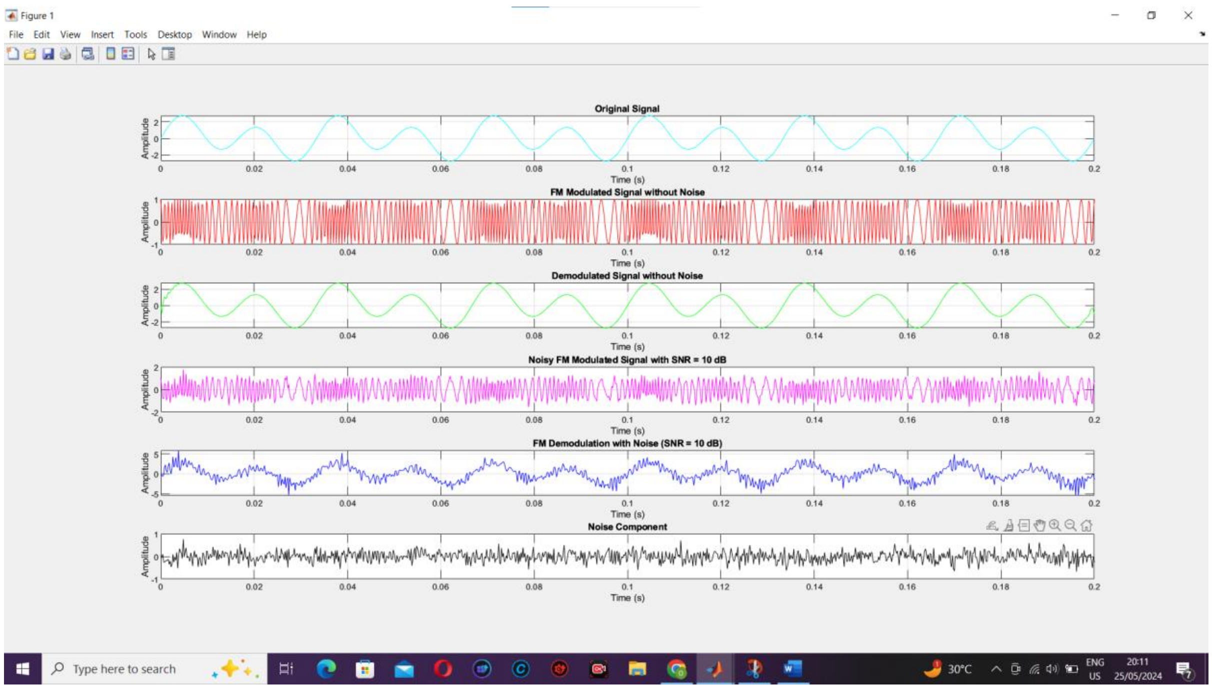


Figure 7 Time-Domain Analysis with Built-in Functions, SNR = 10 dB

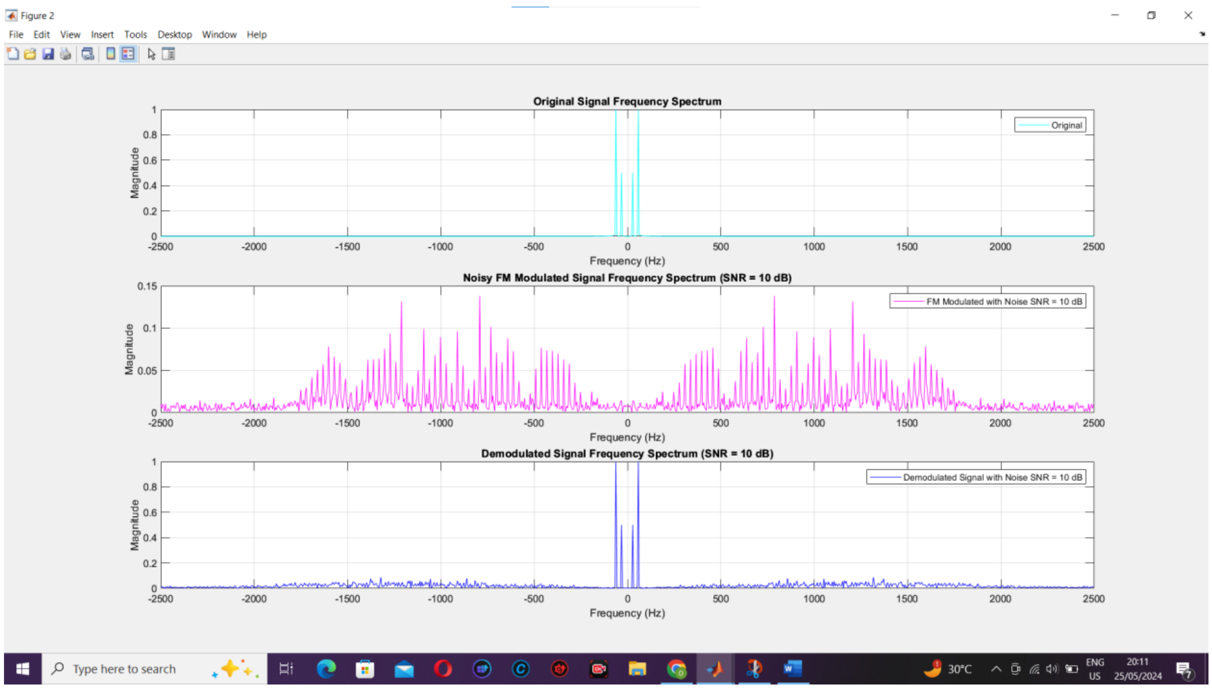


Figure 8 Frequency-Domain Analysis with Built-in Functions, SNR = 10 dB

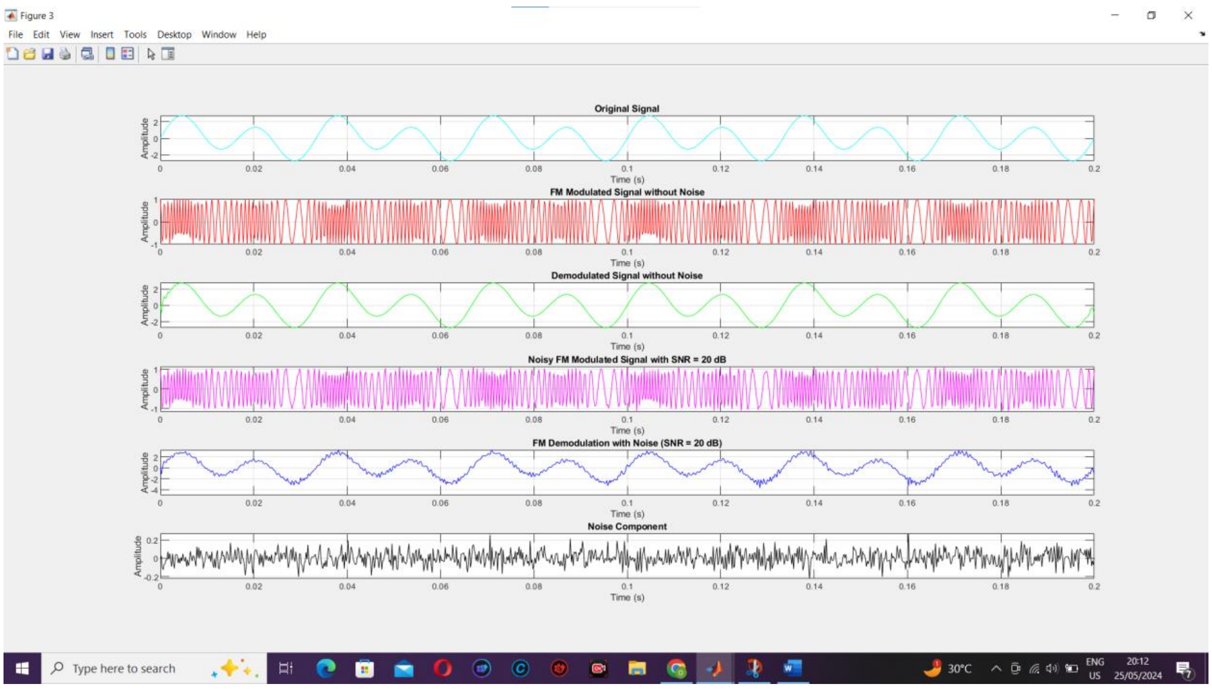


Figure 3
File Edit View Insert Tools Desktop Window Help
Original Signal
FM Modulated Signal without Noise
Demodulated Signal without Noise
Noisy FM Modulated Signal with SNR = 20 dB
FM Demodulation with Noise (SNR = 20 dB)
Noise Component
Amplitude
Time (s)

Figure 9 Time-Domain Analysis with Built-in Functions (Higher SNR), SNR = 20 dB

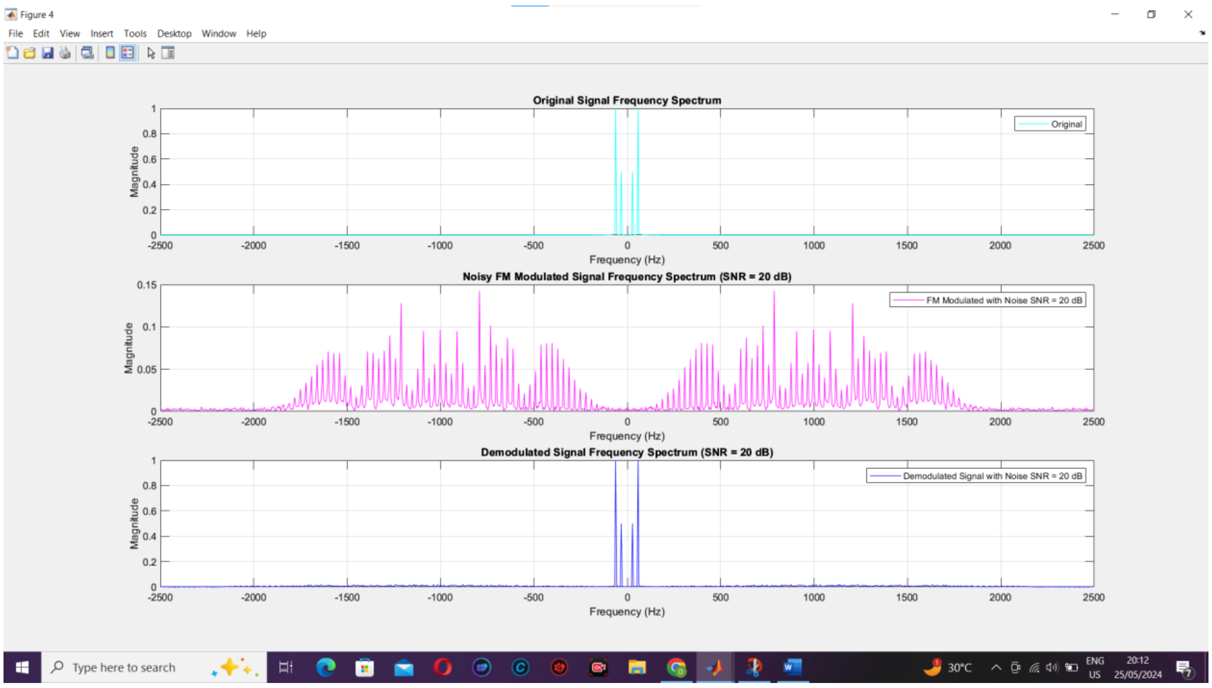


Figure 4
File Edit View Insert Tools Desktop Window Help
Original Signal Frequency Spectrum
Original
Noisy FM Modulated Signal Frequency Spectrum (SNR = 20 dB)
FM Modulated with Noise SNR = 20 dB
Demodulated Signal Frequency Spectrum (SNR = 20 dB)
Demodulated Signal with Noise SNR = 20 dB
Magnitude
Frequency (Hz)

Figure 10 Frequency-Domain Analysis with Built-in Functions, SNR = 20 dB

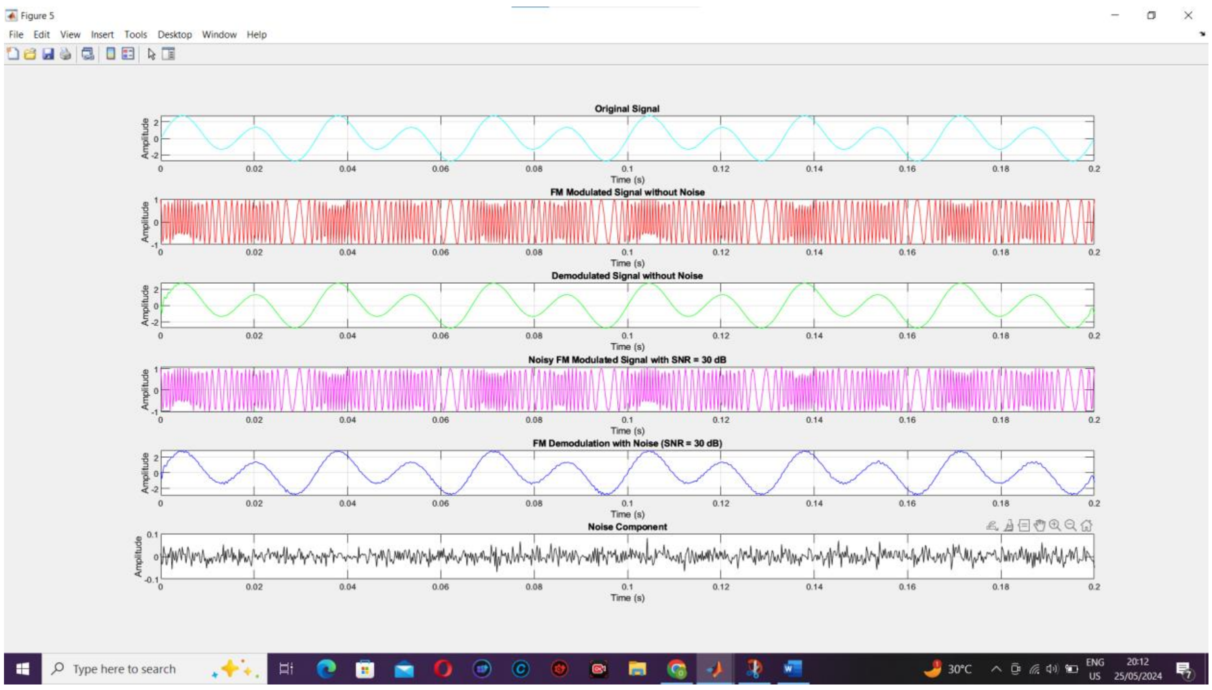


Figure 11 Time-Domain Analysis with Built-in Functions, SNR = 30 dB

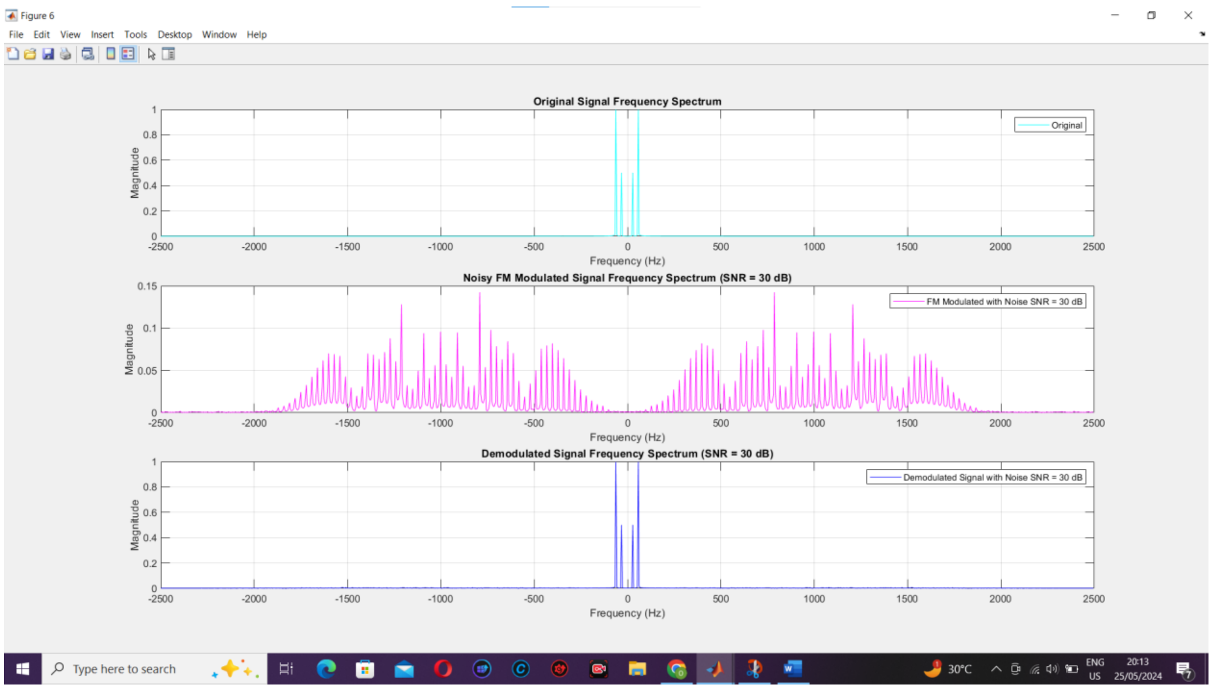


Figure 12 Frequency-Domain Analysis with Built-in Functions, SNR = 30 dB

## 2)FM code without built-in functions

```
Fs = 1000; % Sampling frequency
fc = 200;  % Carrier frequency
t = (0:1/Fs:0.2)'; % Time vector
x = sin(2*pi*30*t) + 2*sin(2*pi*60*t); % Message signal

fDev = 35; % Frequency deviation

% FM modulation without using fmmod
int_x = cumsum(x) / Fs;
xfm = cos(2*pi*fc*t + 2*pi*fDev*int_x);

% FM demodulation without using fmdemod
t2 = t;
xfmq = hilbert(xfm) .* exp(-1i*2*pi*fc*t2);
z_no_noise = (1/(2*pi*fDev)) * [0; diff(unwrap(angle(xfmq)))*Fs];

% Calculate the instantaneous power of the modulated signal
instantaneous_power = abs(xfm).^2;

% Numerical integration over the available time interval
integrated_power = trapz(t, instantaneous_power);

% Average power over the interval
average_power = integrated_power / (t(end) - t(1));

% Display the result of the average power
disp(['Average power of the FM modulated signal: ', num2str(average_power)]);

% Define SNR values in dB
SNR_values = [10, 20, 30];

% Pre-allocate space for noisy signals
xfm_noisy = zeros(length(xfm), length(SNR_values));

% Add noise based on integrated power
for i = 1:length(SNR_values)
    % Convert SNR from dB to linear scale
    snr_linear = 10^(SNR_values(i)/10);

    % Calculate noise power based on the SNR
    noise_power = average_power / snr_linear;

    % Generate white Gaussian noise with the calculated noise power
    noise = sqrt(noise_power) * randn(size(xfm));

    % Add noise to the FM modulated signal
    xfm_noisy(:, i) = xfm + noise;
end

% FM demodulation without using fmdemod
xfmq_no_noise = hilbert(xfm) .* exp(-1i*2*pi*fc*t);
z_no_noise = (1/(2*pi*fDev)) * [0; diff(unwrap(angle(xfmq_no_noise)))*Fs];

% Plotting
for i = 1:length(SNR_values)
    figure;
    % Plot Original Signal
    subplot(6, 1, 1);
    plot(t, x, 'c');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Original Signal');
    grid on;

    % Plot FM Modulated Signal without Noise
    subplot(6, 1, 2);
    plot(t, xfm, 'r');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('FM Modulated Signal without Noise');
    grid on;

    % Plot Demodulated Signal without Noise
    subplot(6, 1, 3);
    plot(t2, z_no_noise, 'g');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Demodulated Signal without Noise');
    grid on;

    % Plot Noisy FM Modulated Signal
    subplot(6, 1, 4);
    plot(t, xfm_noisy(:, i), 'm');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title(['Noisy FM Modulated Signal with SNR = ', num2str(SNR_values(i)), '
dB']);
    grid on;

    % Extract and Plot Noise Component
    noise = xfm_noisy(:, i) - xfm;
    subplot(6, 1, 5);
    plot(t, noise, 'k');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Noise Component');
    grid on;

    % FM demodulation without using fmdemod for the noisy signal
    xfmq_noisy = hilbert(xfm_noisy(:, i)) .* exp(-1i*2*pi*fc*t);
    z_noisy = (1/(2*pi*fDev)) * [0; diff(unwrap(angle(xfmq_noisy)))*Fs];

    % Plot Demodulated Signal with Noise
    subplot(6, 1, 6);
    plot(t2, z_noisy, 'b');
    xlabel('Time (s)');
    ylabel('Amplitude');
    title(['FM Demodulation with Noise (SNR = ', num2str(SNR_values(i)), ' dB)']);
    grid on;

    % Frequency Domain Analysis
    figure;

    % FFT of the original signal
    N = length(t);
    X = fftshift(fft(x, N));
    X_magnitude = abs(X)/N;
    frequencies = (-N/2:N/2-1)*(Fs/N);

    % FFT of the noisy FM modulated signal
    XFM_noisy = fftshift(fft(xfm_noisy(:, i), N));
    XFM_noisy_magnitude = abs(XFM_noisy)/N;

    % FFT of the demodulated noisy signal
    Z_noisy = fftshift(fft(z_noisy, N));
    Z_noisy_magnitude = abs(Z_noisy)/N;

    subplot(3, 1, 1);
    plot(frequencies, X_magnitude, 'c', 'DisplayName', 'Original');
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('Original Signal Frequency Spectrum');
    legend;
    grid on;

    subplot(3, 1, 2);
    plot(frequencies, XFM_noisy_magnitude, 'm', 'DisplayName', ['FM Modulated with
Noise SNR = ', num2str(SNR_values(i)), ' dB']);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title(['Noisy FM Modulated Signal Frequency Spectrum (SNR = ',
num2str(SNR_values(i)), ' dB)']);
    legend;
    grid on;

    subplot(3, 1, 3);
    plot(frequencies, Z_noisy_magnitude, 'b', 'DisplayName', ['Demodulated Signal
with Noise SNR = ', num2str(SNR_values(i)), ' dB']);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title(['Demodulated Signal Frequency Spectrum (SNR = ',
num2str(SNR_values(i)), ' dB)']);
    legend;
    grid on;
end
```

**Effects of Changing SNR Values**

**Time-Domain Effects:**

**Noisy FM Modulated Signal:** As the SNR value increases from 10 dB to 20 dB to 30 dB, the amount of noise in the FM modulated signal decreases. At 10 dB, the noise is relatively significant, while at 20 dB, it becomes less noticeable, and at 30 dB, the noise is minimal.

**Noise Component:** The noise component extracted from the noisy FM modulated signal decreases as SNR increases. The noise amplitude is higher at 10 dB and becomes progressively smaller at 20 dB and 30 dB.
Demodulated Signal with Noise: The quality of the demodulated signal improves with increasing SNR. At 10 dB, the demodulated signal has noticeable distortions due to noise, but as the SNR increases to 20 dB and 30 dB, the demodulated signal becomes cleaner and more similar to the original message signal.

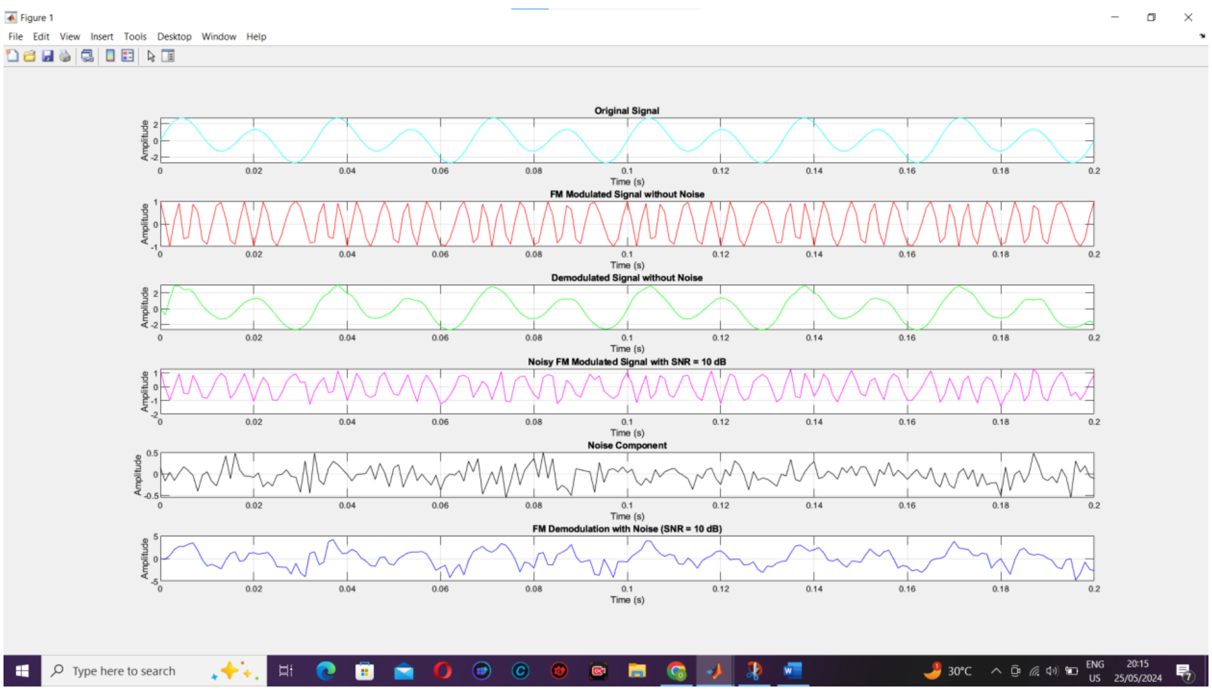


Figure 13 Time-Domain Analysis with Built-in Functions, SNR = 10 dB

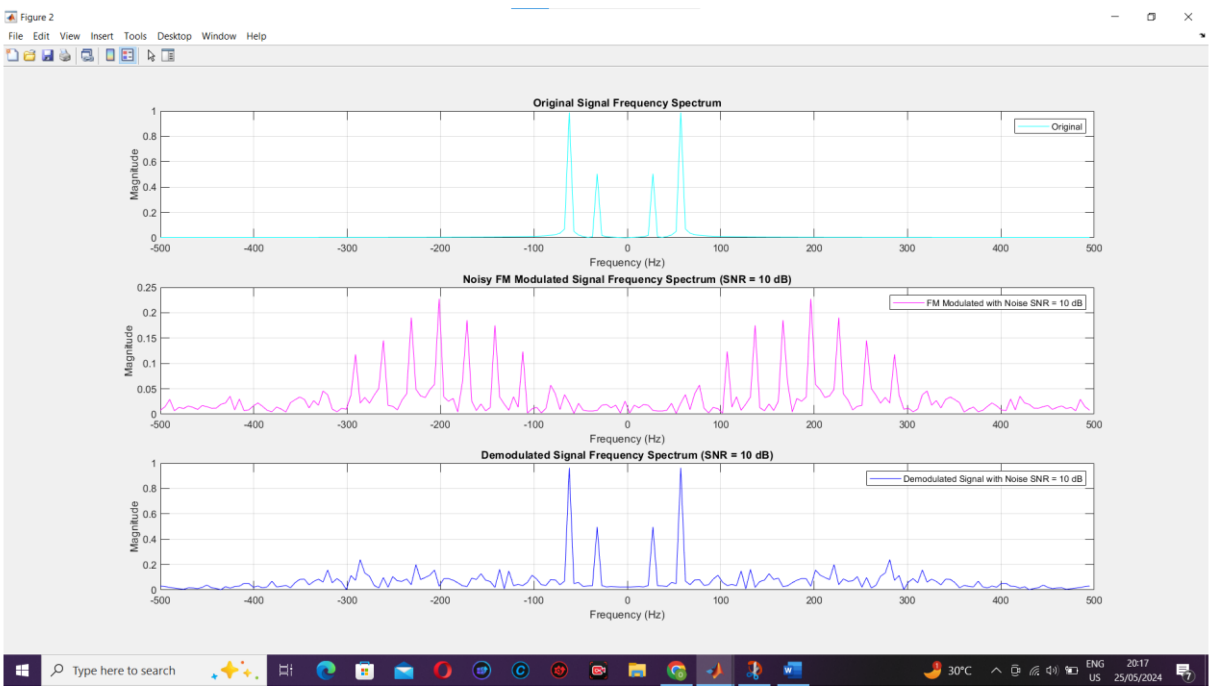


Figure 14 Frequency-Domain Analysis with Built-in Functions, SNR = 10 dB

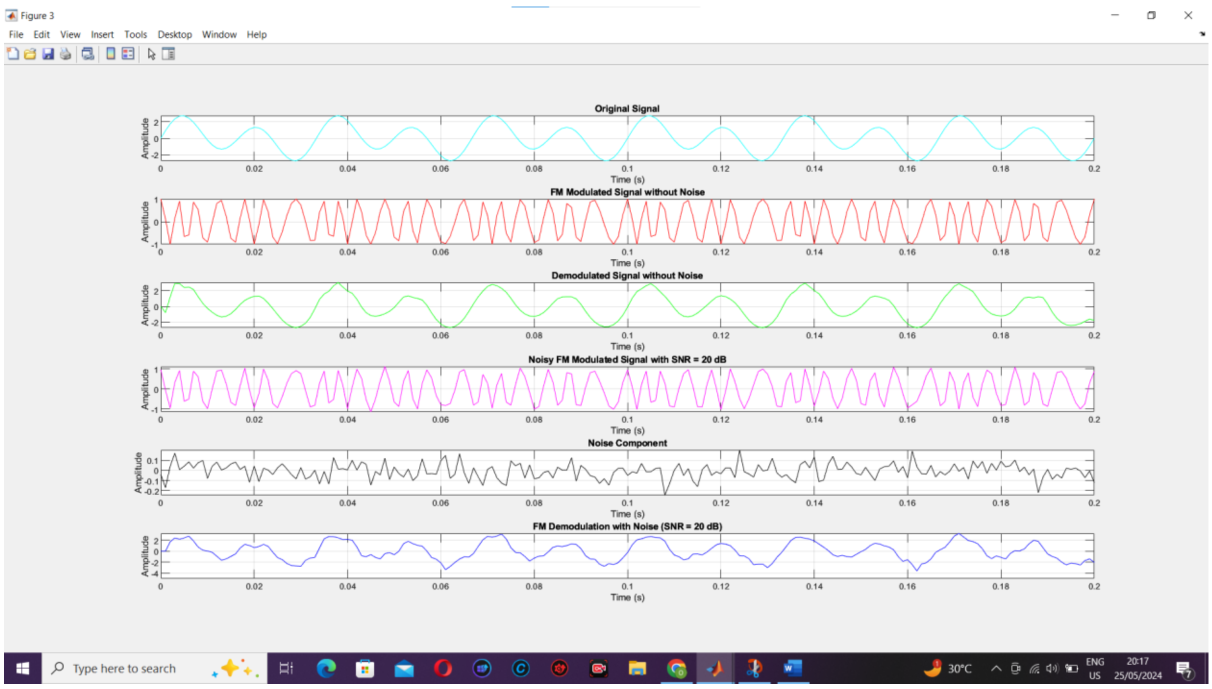


Figure 15 Time-Domain Analysis with Built-in Functions (Higher SNR), SNR = 20 dB

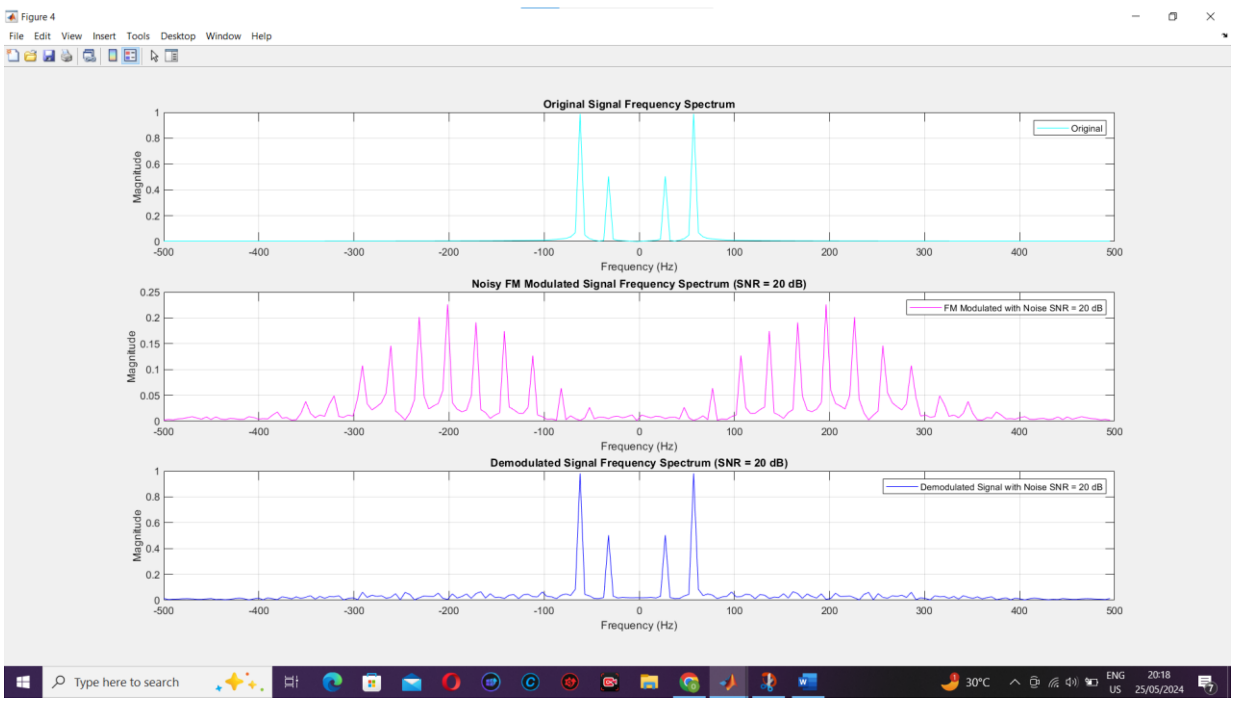


Figure 16 Figure 4 Frequency-Domain Analysis with Built-in Functions, SNR = 20 dB

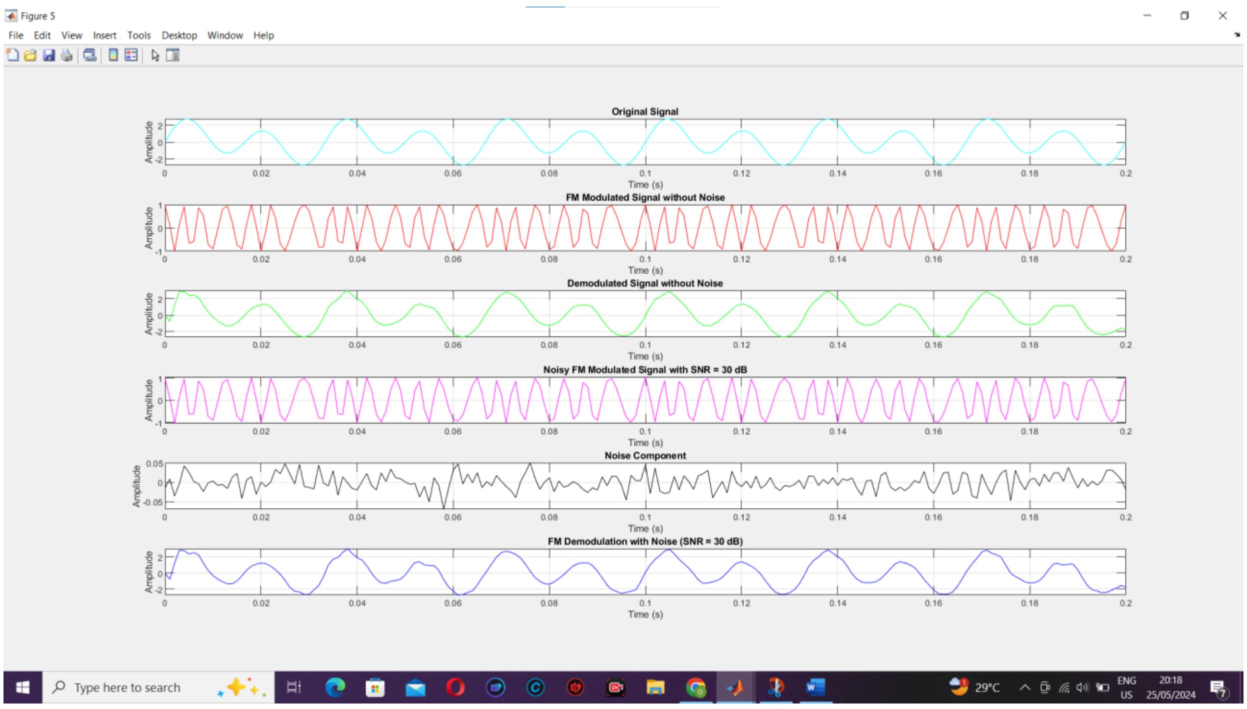


Figure 5
File Edit View Insert Tools Desktop Window Help
Original Signal
FM Modulated Signal without Noise
Demodulated Signal without Noise
Noisy FM Modulated Signal with SNR = 30 dB
Noise Component
FM Demodulation with Noise (SNR = 30 dB)
Amplitude
Time (s)
Type here to search
29°C
ENG US
20:18
25/05/2024

Figure 17 Time-Domain Analysis with Built-in Functions, SNR = 30 dB

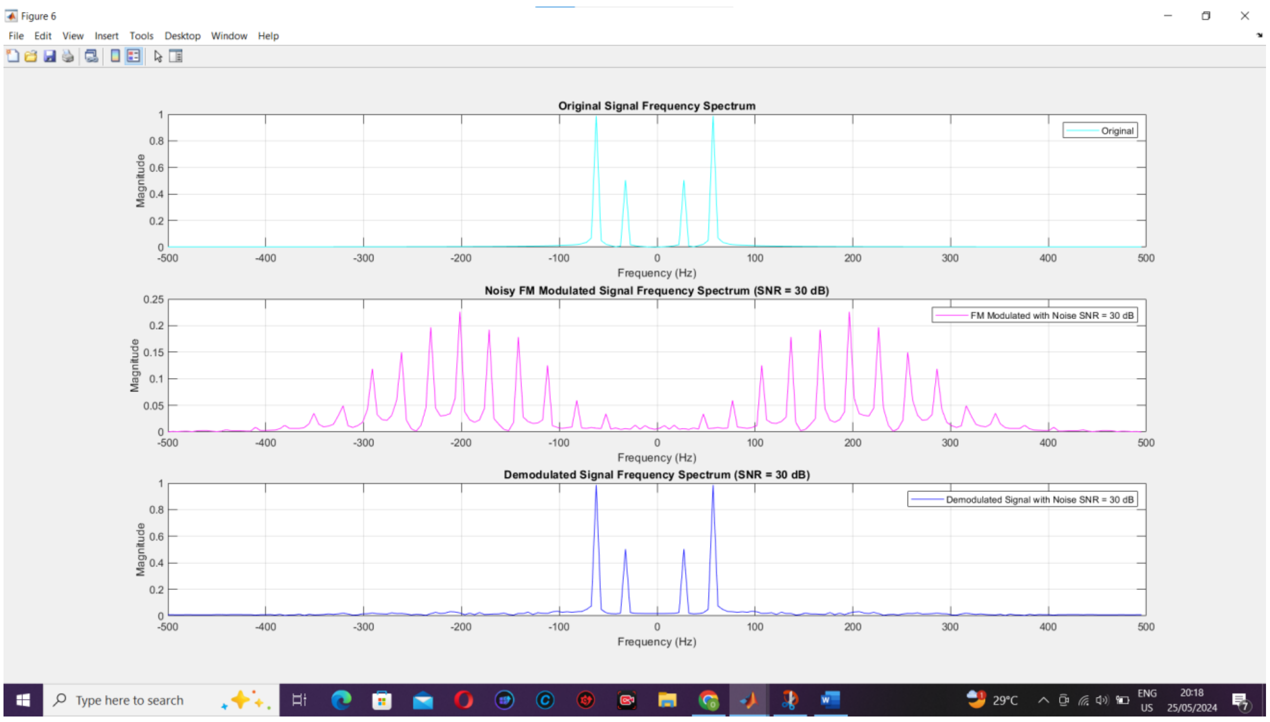
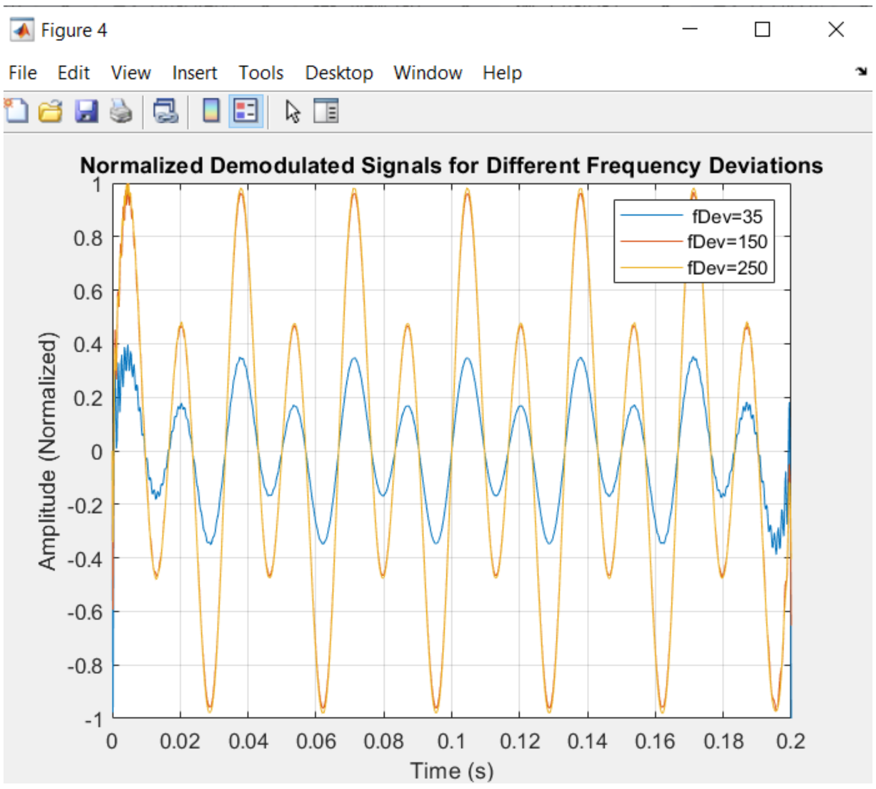


Figure 6
File Edit View Insert Tools Desktop Window Help
Original Signal Frequency Spectrum
Original
Noisy FM Modulated Signal Frequency Spectrum (SNR = 30 dB)
FM Modulated with Noise SNR = 30 dB
Demodulated Signal Frequency Spectrum (SNR = 30 dB)
Demodulated Signal with Noise SNR = 30 dB
Magnitude
Frequency (Hz)
Type here to search
29°C
ENG US
20:18
25/05/2024

Figure 18 Frequency-Domain Analysis with Built-in Functions, SNR = 30

**Study the impact of varying the frequency deviation in the FM modulator on the demodulated FM signal in the time domain.**



**1. Effect on FM Modulated Signal**

**Frequency Deviation**

**Low Deviation (35 Hz):**

The carrier frequency varies by a smaller amount in response to the message signal.

The bandwidth of the FM signal is relatively narrow.

The signal might not be as robust against noise, but the spectrum is less spread out.

**Medium Deviation (150 Hz):**

The carrier frequency varies more significantly in response to the message signal compared to the low deviation.

The bandwidth of the FM signal increases.

The signal becomes more robust against noise with a wider spread in the spectrum

**High Deviation (250 Hz):**

The carrier frequency varies by a larger amount in response to the message signal.

The bandwidth of the FM signal is the widest.

The signal is more robust against noise, but it occupies a larger portion of the frequency spectrum.

**2. Effect on Demodulated Signal**

**Low Deviation (35 Hz):**

The demodulated signal may be less distinct, especially in the presence of noise, as the carrier frequency does not deviate much.

The signal recovery might be less accurate.

**Medium Deviation (150 Hz):**

The demodulated signal improves in clarity compared to the low deviation case.

Better noise immunity and more accurate signal recovery.

**High Deviation (250 Hz):**

The demodulated signal is the clearest and most distinct, even in the presence of noise.

The signal has the best noise immunity and most accurate recovery, but it requires a larger bandwidth.

## Conclusion

Both DSBTC and FM modulation techniques play crucial roles in analog communication systems. DSBTC is valued for its straightforward demodulation process, making it ideal for applications that prioritize simplicity and reliability. On the other hand, FM modulation is favored for its robust noise immunity and ability to deliver high-quality audio, making it suitable for broadcasting and communication systems where signal integrity is paramount. Understanding these modulation methods is essential for designing effective communication systems tailored to specific needs.

## References

1. https://www.electronics-notes.com/articles/radio/modulation/quadrature-amplitudemodulation-qam-modulator-demodulator.php
2. https://www.faststreamtech.com/products/qam-modulator-anddemodulator/#:~:text=The%20local%20oscillator%20generates%20the,thus%20doubl ing%20a%20system's%20bandwidth.
3. https://www.electronics-notes.com/articles/radio/modulation/single-sideband-ssbdemodulation reception.php#:~:text=SSB%20demodulation%20basics,suppressed%20or%20reduced%20in%20level.&text=In%20order%20to%20demodulate%20single,necessary%20to%20reintroduce%20the%20carrier