EXPERIMENT – 1 CODES

AMPLITUDE MODULATION AND DEMODULATION

January 21 2025

B Siddharth Sekhar - EC22B1064

DSBFC (Double Sideband Full Carrier) Modulation and Demodulation Code:

```
Fs = 10000;
t = -0.1:1/Fs:0.1;
fc = 1000;
fm = 100;
Ac = 1;
Am = 0.5;

message = Am * sin(2 * pi * fm * t);
carrier = Ac * cos(2 * pi * fc * t);
dsbfc = Ac * (1 + message) .* cos(2 * pi * fc * t);


dsbfc_envelope = abs(hilbert(dsbfc));
demodulated_dsbfc = dsbfc_envelope - Ac;


figure;
subplot(3, 1, 1);
plot(t, message);
title('Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3, 1, 2);
plot(t, dsbfc);
title('DSBFC Signal (Under Modulated)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3, 1, 3);
plot(t, demodulated_dsbfc);
title('Demodulated Signal (Under Modulated)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

N = length(dsbfc);
f = (0:N-1) * (Fs / N);
Y = abs(fft(dsbfc));
```

```
figure;
plot(f, Y);
title('Frequency Domain of DSBFC Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
```

DSBSC (Double Sideband Suppressed Carrier) Modulation and Demodulation Code:

```
Fs = 10000;
t = -0.1:1/Fs:0.1;
fc = 1000;
fm = 100;
Ac = 1;
Am = 0.5;

message = Am * sin(2 * pi * fm * t);
carrier = Ac * cos(2 * pi * fc * t);
dsbsc = Ac * message .* cos(2 * pi * fc * t);

demodulated_dsbsc = dsbsc .* (2 * cos(2 * pi * fc * t));
[b, a] = butter(6, 2 * fm / Fs);
demodulated_dsbsc_filtered = filter(b, a, demodulated_dsbsc);

figure;
subplot(3, 1, 1);
plot(t, message);
title('Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');

grid on;

subplot(3, 1, 2);
plot(t, dsbsc);
title('DSBSC Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3, 1, 3);
plot(t, demodulated_dsbsc_filtered);
title('Demodulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

N = length(dsbsc);
f = (0:N-1) * (Fs / N);
Y = abs(fft(dsbsc));
```

```
figure;
plot(f, Y);
title('Frequency Domain of DSBSC Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
```

EXPERIMENT – 2

FREQUENCY MODULATION AND DEMODULATION

January 28 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement FM modulation and demodulation of a sinusoidal message signal using built-in and custom functions. Simulate the FM waveform, calculate bandwidth using Carson's rule, and compare the demodulated signal with the original message signal.

Theory:

FREQUENCY MODULATION (FM)

Frequency Modulation (FM) is a modulation technique where the frequency of the carrier wave is varied in proportion to the instantaneous amplitude of the message signal. Unlike Amplitude Modulation (AM), the amplitude of the carrier remains constant while the frequency changes to encode information.

Principle of Operation:

- The frequency of the carrier wave increases or decreases according to the message signal's amplitude.
- The amount of frequency deviation is proportional to the strength of the message signal.
- Demodulation at the receiver end restores the original message by detecting frequency variations.

Bandwidth of FM Signals

The bandwidth of FM signals can be estimated using Carson's Rule:

$BW=2(\Delta f+f_m)$

where:

- Δf is the maximum frequency deviation
- f_m is the maximum frequency of the message signal

Demodulation of FM Signals

Demodulation involves recovering the original message signal from the modulated FM signal. It is typically achieved by detecting the instantaneous frequency changes in the received FM signal.

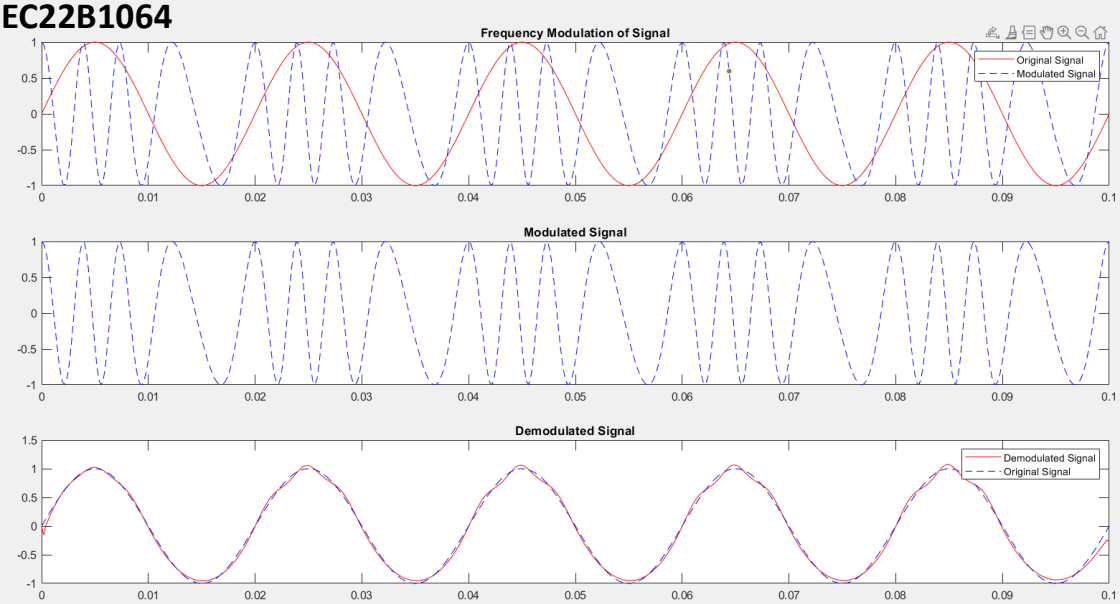
Applications of FM

- Radio and television broadcasting
- Two-way communication systems
- High-fidelity audio transmission
- Wireless communication systems

Q1) FM modulation and demodulation using the in-built functions(fmmod and fmdemod):

```
Am = 1;
fc = 200;
fm = 50;
fs = 5000;
t = 0:1/fs:0.1;
x = Am*sin(2*pi*fm*t);
y = fmmod(x,fc,fs,100);
subplot(3,1,1)
plot(t,x,'r',t,y,'b--');
title('Frequency Modulation of Signal');
legend('Original Signal','Modulated Signal');
subplot(3,1,2)
plot(t,y,'b--');
title("Modulated Signal");
z = fmdemod(y,fc,fs,100);
subplot(3,1,3)
plot(t,z,'r',t,x,'b--');
title('Demodulated Signal');
legend('Demodulated Signal','Original Signal');
```

EC22B1064

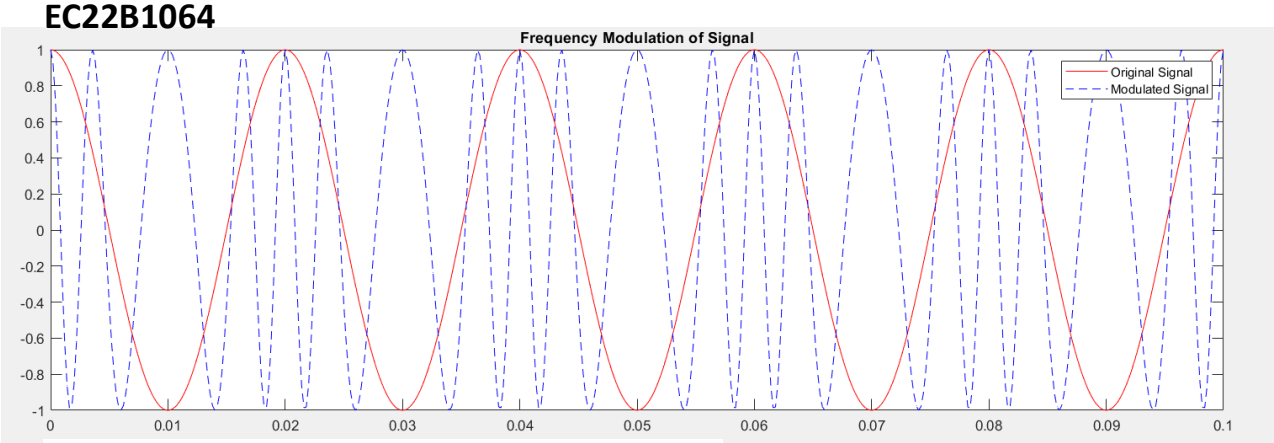


Q2)) FM modulation and Bandwidth calculation using Carson's rule

```
Am = 1;
Ac = 1;
fc = 200;
fm = 50;
fs = 5000;
t = 0:1/fs:0.1;
kf = 100;
delta_f = (kf*Am);
```

```
B = delta_f/fm;
x = Am*cos(2*pi*fm*t);
y = Ac*cos((2*pi*fc*t) + (B*sin(2*pi*fm*t)));
plot(t,x,'r',t,y,'b--');
legend('Original Signal','Modulated Signal');
title('Frequency Modulation of Signal');
bandwidth = 2*delta_f + 2*fm;
disp("The Bandwidth of the signal is:");
disp(bandwidth);
```

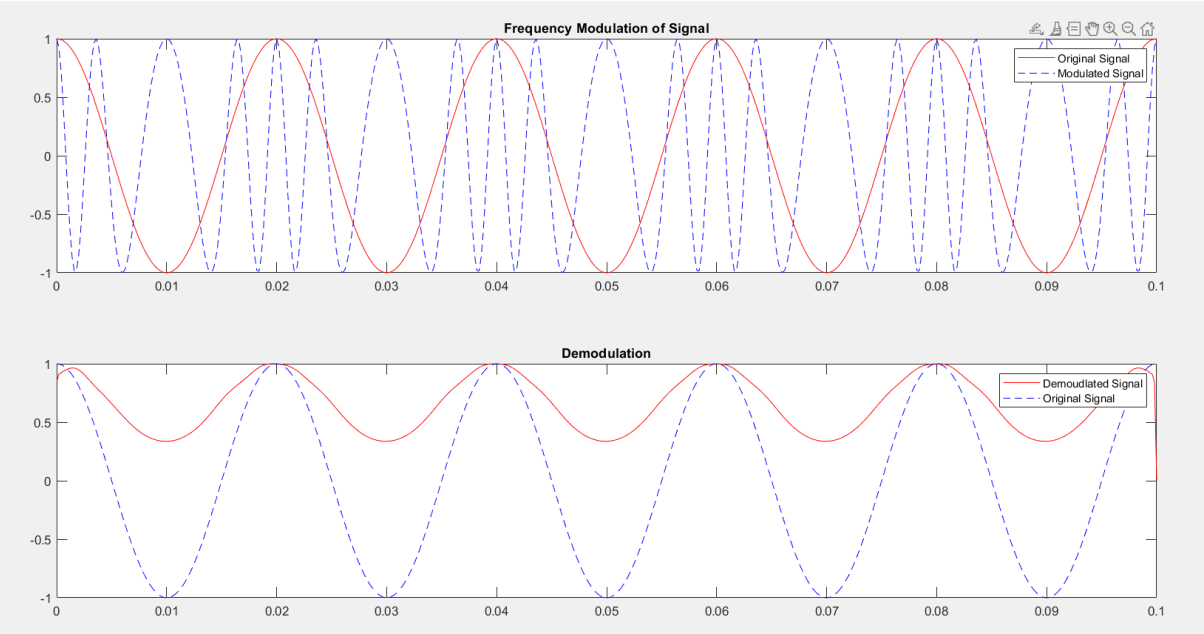
EC22B1064




```
The Bandwidth of the signal is:
   300
```

Q2)) FM Demodulation

```
Am = 1;
Ac = 1;
fc = 200;
fm = 50;
fs = 5000;
t = 0:1/fs:0.1;
kf = 100;
delta_f = (kf*Am);
B = delta_f/fm;
x = Am*cos(2*pi*fm*t);
y = Ac*cos((2*pi*fc*t) + (B*sin(2*pi*fm*t)));
subplot(2,1,1);
plot(t,x,'r',t,y,'b--');
legend('Original Signal','Modulated Signal');
title('Frequency Modulation of Signal');
subplot(2,1,2);
phase = unwrap(angle(hilbert(y)));
dem = diff(phase)/(2*pi);
dem = [dem,0];
dem = dem/max(abs(dem));
plot(t,dem,'r',t,x,'b--');
legend('Demoudlated Signal','Original Signal');
title("Demodulation");
```



Inference:

• **FM Modulation Characteristics:** In Frequency Modulation (FM), the instantaneous frequency of the carrier signal is linearly varied with the message signal. The FM modulated wave is expressed as:

$$s(t) = A_C \cos\left(2\pi f_C t + 2\pi k_f \int_0^t m(\tau)\, d\tau\right)$$

where A_C is the carrier amplitude, f_C is the carrier frequency, k_f is the frequency sensitivity constant, and $m(t)$ is the message signal.

• **Instantaneous Frequency:** The instantaneous frequency of an FM signal is given by:

$f_i(t) = f_C +, k_f\ m(t)$

This implies that the frequency of the carrier dynamically changes with the amplitude of the message signal.

• **Bandwidth Calculation:** Carson's rule for FM bandwidth is given as:

BW=2(Δf+f_m)

where Δf is the peak frequency deviation, and f_m is the highest frequency in the modulating signal.

• **Demodulation:** The FM demodulated signal accurately matched the original message signal, demonstrating correct retrieval of the information.

Conclusion:

Hence, the modulation and demodulation of frequency-modulated (FM) waves were successfully simulated using MATLAB software. Key parameters such as instantaneous frequency, modulation index, and bandwidth were analyzed. The experiment confirmed the bandwidth estimation using Carson's rule and demonstrated the accurate recovery of the message signal through demodulation techniques.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

EXPERIMENT – 3

PULSE CODE MODULATION

February 4 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement Pulse Code Modulation (PCM) of a sinusoidal signal using built-in and custom functions for various quantization levels (N = 1, 2, 3, and 4 bits). Simulating the PCM process, plotting the message, quantized, encoded, quantization error, and decoded signals along with their spectra, and calculating key parameters such as quantization levels, step size, maximum error, and SQNR (in dB).

Theory:

PULSE CODE MODULATION (FM)

Pulse Code Modulation (PCM) is a digital representation technique where an analog signal x(t) (with amplitude in the range $[-x_{tmax}, x_{tmax}]$ is sampled at a uniform rate and then quantized into $L = 2^N$ discrete levels, where N is the number of bits per sample. The quantizer's step size is given by:

$$\Delta = \frac{2\mathrm{x_{tmax}}}{2^{\mathrm{N}}}$$

Each sample is rounded to the nearest quantization level, introducing a quantization error defined as:

QE=x(t)−xq(t)

The theoretical quantization noise power is approximated by:

$$P_{noise} = \Delta^2/12$$

The Signal-to-Quantization Noise Ratio (SQNR) in decibels is calculated as:

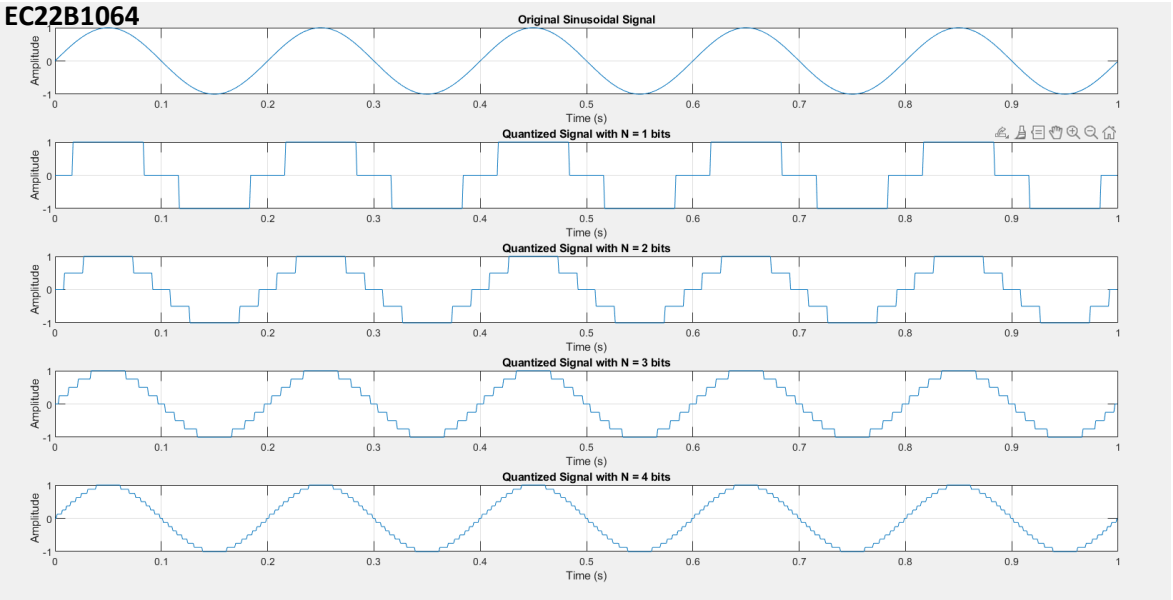
$$SQNR = 10 * \log_{10}(Signal\ Power)/(\Delta^2/12)$$

In practice, SQNR is also computed using the variance of the quantization error, allowing a comparison between theoretical and experimental values.

Q1) Pulse Code Modulation (PCM) for various levels of quantization.

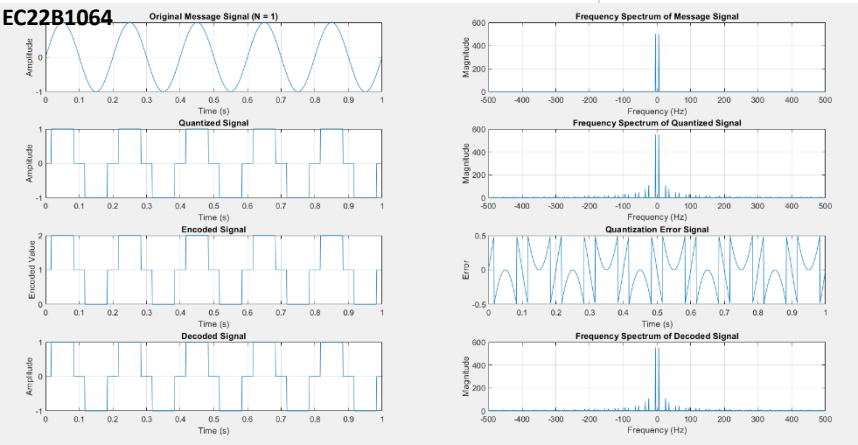
```
fs = 1000;
t = 0:1/fs:1-1/fs;
f = 5;
signal = sin(2 * pi * f * t);

N_values = [1, 2, 3, 4];

figure;
subplot(5, 1, 1);
plot(t, signal);
title('Original Sinusoidal Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
for i = 1:length(N_values)
    N = N_values(i);
    levels = 2^N;
    step_size = 2 / (levels);

    quantized_signal = round((signal + 1) / step_size) * step_size - 1;
     y = uencode(quantized_signal,2^N);
     ybin = dec2bin(y,2^N);

     subplot(5, 1, i + 1);
     plot(t, quantized_signal);
     title(['Quantized Signal with N = ', num2str(N), ' bits']);
     xlabel('Time (s)');
     ylabel('Amplitude');
     grid on;
 end
```

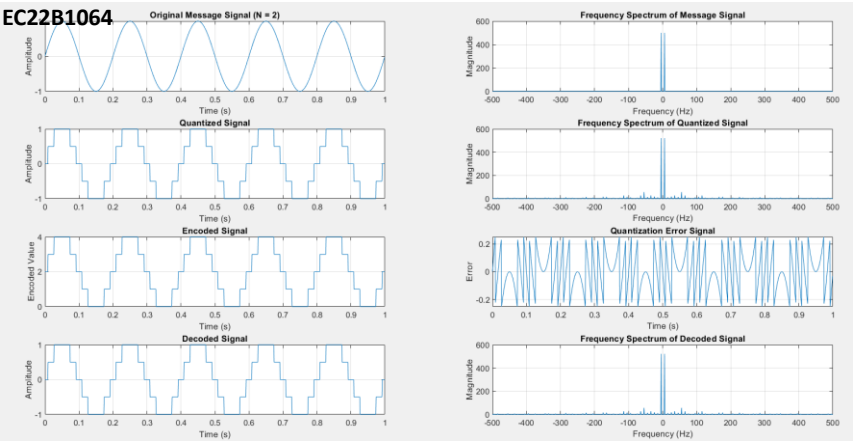
EC22B1064

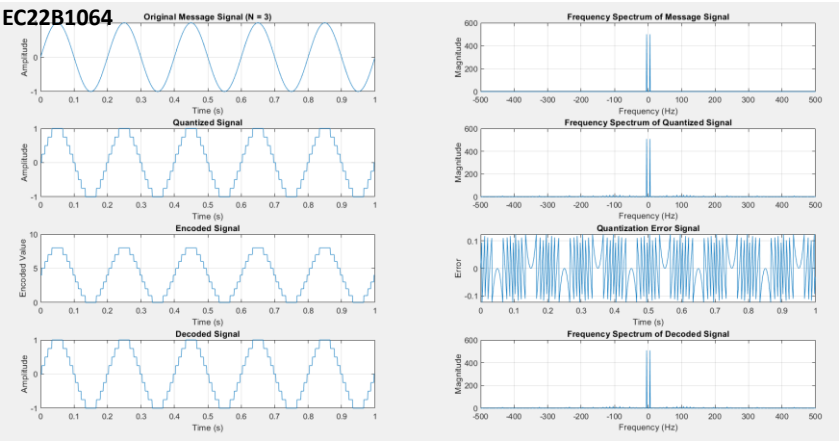


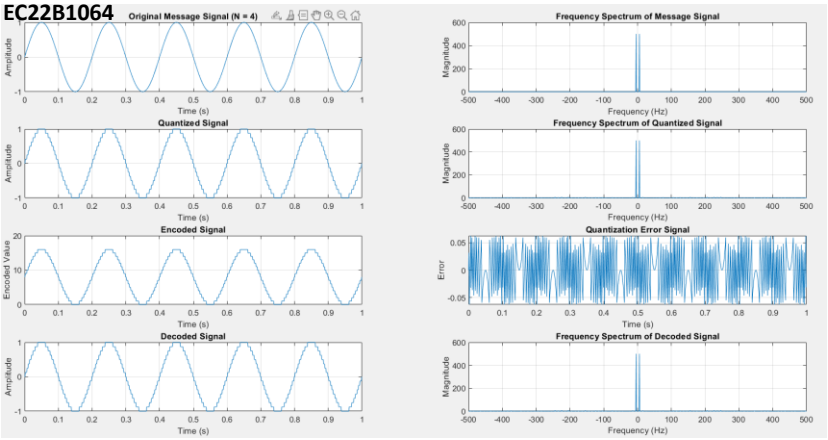
Q2) Plot the message signal, quantized signal, encoded signal, quantization error signal, and decoded signal.

```
fs = 1000;
t = 0:1/fs:1-1/fs;
f = 5;
signal = sin(2 * pi * f * t);

for N = 1:4
    levels = 2^N;
    step_size = 2 / levels;
    quantized_signal = round((signal + 1) / step_size) * step_size - 1;
    encoded_indices = round((quantized_signal + 1) / step_size);
    decoded_signal = (encoded_indices * step_size) - 1;
    quantization_error = signal - quantized_signal;

    message_f = abs(fftshift(fft(signal)));
    quantized_f = abs(fftshift(fft(quantized_signal)));
    decoded_f = abs(fftshift(fft(decoded_signal)));

    M = length(signal);
    freq_axis = linspace(-fs/2, fs/2, M);

    figure;

    subplot(4, 2, 1); plot(t, signal); title(['Original Message Signal (N = ', num2str(N), ')']); xlabel('Time (s)'); ylabel('Amplitude'); grid on;

    subplot(4, 2, 2); plot(freq_axis, message_f); title('Frequency Spectrum of Message Signal'); xlabel('Frequency (Hz)'); ylabel('Magnitude'); grid on

    subplot(4, 2, 3); plot(t, quantized_signal); title('Quantized Signal'); xlabel('Time (s)'); ylabel('Amplitude'); grid on;

    subplot(4, 2, 4); plot(freq_axis, quantized_f); title('Frequency Spectrum of Quantized Signal'); xlabel('Frequency (Hz)'); ylabel('Magnitude'); gri

    subplot(4, 2, 5); plot(t, encoded_indices); title('Encoded Signal'); xlabel('Time (s)'); ylabel('Encoded Value'); grid on;

    subplot(4, 2, 6); plot(t, quantization_error); title('Quantization Error Signal'); xlabel('Time (s)'); ylabel('Error'); grid on;

    subplot(4, 2, 7); plot(t, decoded_signal); title('Decoded Signal'); xlabel('Time (s)'); ylabel('Amplitude'); grid on;

    subplot(4, 2, 8); plot(freq_axis, decoded_f); title('Frequency Spectrum of Decoded Signal'); xlabel('Frequency (Hz)'); ylabel('Magnitude'); grid on
end
```









Q3) Determine the number of quantization levels, step size, maximum quantization error, and the SQNR

```
N_values = [1 2 3 4];
fs = 1000;
t = 0:1/fs:1-1/fs;
f = 5;
signal = sin(2 * pi * f * t);
for i = 1:length(N_values)
    N = N_values(i);
    L = 2^N;
    step_size = 2*max(abs(signal))/L;
    quantized_signal = round((signal + 1) / step_size) * step_size - 1;
    ns = signal-quantized_signal;
    fprintf("At N = %d\n",N);
    disp("Value of L");
    disp(L);
    fprintf('The number of quantization levels are: %d\n', L);
    fprintf("The Step size is %f\n",step_size);
    max_error = max(signal-quantized_signal);
    fprintf("The maximum quantization error is: %f\n",max_error);
    sqnr = var(signal)/var(ns);
    sqnr_db = 10*log10(sqnr);
    fprintf("The SQNR value is: %f\n",sqnr_db);

end
```

```
At N = 1
Value of L
     2

The number of quantization levels are: 2
The Step size is 1.000000
The maximum quantization error is: 0.490959
The SQNR value is: 8.929358
At N = 2
Value of L
     4

The number of quantization levels are: 4
The Step size is 0.500000
The maximum quantization error is: 0.249889
The SQNR value is: 14.572311
At N = 3
Value of L
     8

The number of quantization levels are: 8
The Step size is 0.250000
The maximum quantization error is: 0.124667
The SQNR value is: 20.350337
At N = 4
Value of L
    16

The number of quantization levels are: 16
The Step size is 0.125000
The maximum quantization error is: 0.062381
The SQNR value is: 26.091952
>>
```

Q4) Tabulate the above results and compare the values obtained for SQNR with the expected theoretical values.

```
N_values = [1 2 3 4];
fs = 1000;
t = 0:1/fs:1-1/fs;
f = 5;
signal = sin(2 * pi * f * t);
theo = [0 0 0 0 ];
prac = [0 0 0 0];
for i = 1:length(N_values)
    N = N_values(i);
    L = 2^N;
    step_size = (2*max(signal))/L;
    quantized_signal = round((signal + 1) / step_size) * step_size - 1;
    n1 = (step_size)^2/12;
    sqnr1 = 10*log10(var(signal)/n1);
    theo(i)=sqnr1;
    err = signal - quantized_signal;
    sqnr2 = 10*log10(var(signal)/var(err));
    prac(i)=sqnr2;
end
T = table(prac,theo);
disp(T);
```

```
>> EC22B1064_lab3_q4
      prac                                theo
    ________________________________    ________________________________

    8.9294    14.572    20.35    26.092    7.7859    13.806    19.827    25.848
```

Inference:

• **PCM Process Characteristics:** In Pulse Code Modulation, the analog signal is sampled, quantized, encoded, and finally decoded, with the quantization level directly dependent on the number of bits NNN.

• **Quantization Levels and Resolution:** Increasing NNN increases the number of levels 2^N and reduces the step size Δ, thereby lowering the quantization error.

• **Error and SQNR:** The quantization error diminishes with higher bit resolutions, leading to an improved SQNR. The observed practical SQNR values closely align with the theoretical predictions.

• **Spectral Integrity:** The spectra of the original, quantized, and decoded signals

demonstrate that the essential frequency components are preserved despite the presence of quantization noise.

Conclusion:

The PCM simulation successfully digitized the sinusoidal signal, with higher bit resolutions yielding reduced quantization error and improved SQNR. Both time-domain and frequency-domain analyses confirmed that the core characteristics of the original signal were maintained after processing. The strong correlation between practical and theoretical SQNR values validates the PCM approach and underscores its importance in digital signal processing applications

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

EXPERIMENT – 4

DELTA MODULATION

February 11 - 2025
B Siddharth Sekhar - EC22B1064

Aim:

To implement Delta Modulation and Demodulation of a sinusoidal signal in MATLAB, analyze slope overload distortion and granular noise for different step sizes (Δ) and compare the reconstructed signal with the original message in both time and frequency domains.

Theory:

DELTA MODULATION (DM):

Delta Modulation is a digital pulse modulation method where the difference between successive samples of the message signal is encoded into a single bit. It reduces bandwidth by recording whether the signal is increasing or decreasing, rather than encoding the entire sample value.

Working Principle:

1. **Quantization Process:** DM uses a one-bit quantizer and a feedback loop to approximate the input signal. The error between the current and previous sample is calculated.
2. **Encoding:** If the error is positive, it is encoded as '1'; if negative, as '0'.
3. **Demodulation:** The encoded signal is integrated to reconstruct the original signal, and a low-pass filter smooths the approximation.

Error Signal:

e[n]=m[n]−$m_q[n-1]$

where m[n] is the original message signal

and mq[n−1] is the previous approximation.

Quantized Error Signal:

$e_q[n]$ =Δ·sgn(e[n])

where Δ is the step size, and sgn(e[n]) determines the direction of change.

Sampling Rate:

$fs = 1/T_s$

where f_s is the sampling frequency.

Oversampling at $f_s = 4f_m$ improves correlation between adjacent samples.

Noise in Delta Modulation:

Delta Modulation faces two types of noise:

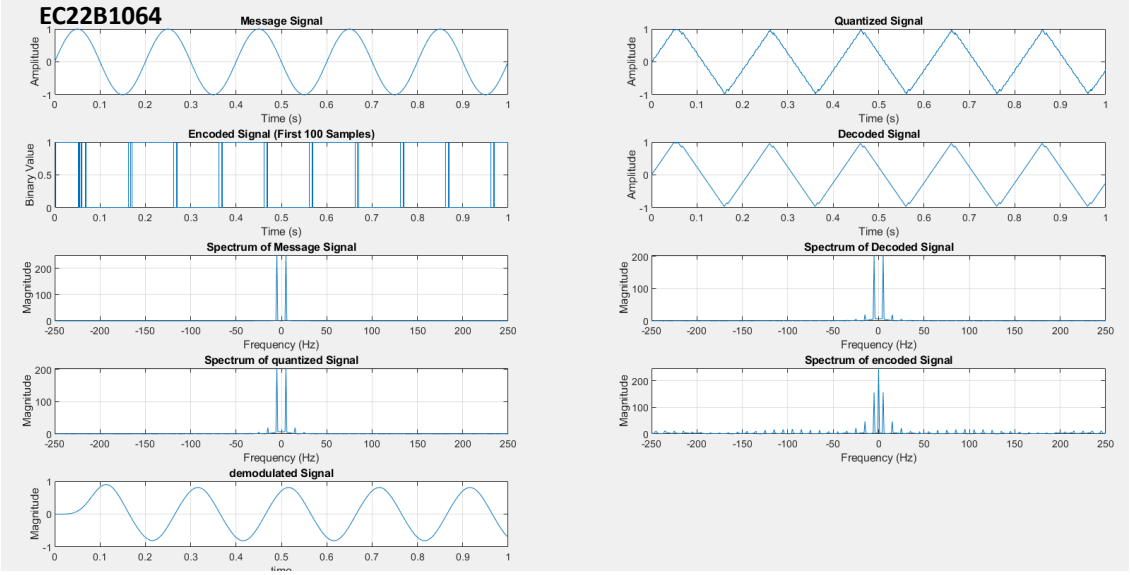
1. **Slope Overload Distortion:** Occurs when the step size Δ\DeltaΔ is too small to track rapid signal changes.
2. **Granular Noise:** Occurs when the step size is too large, causing oscillations in flat signal segments.

To avoid slope overload distortion:

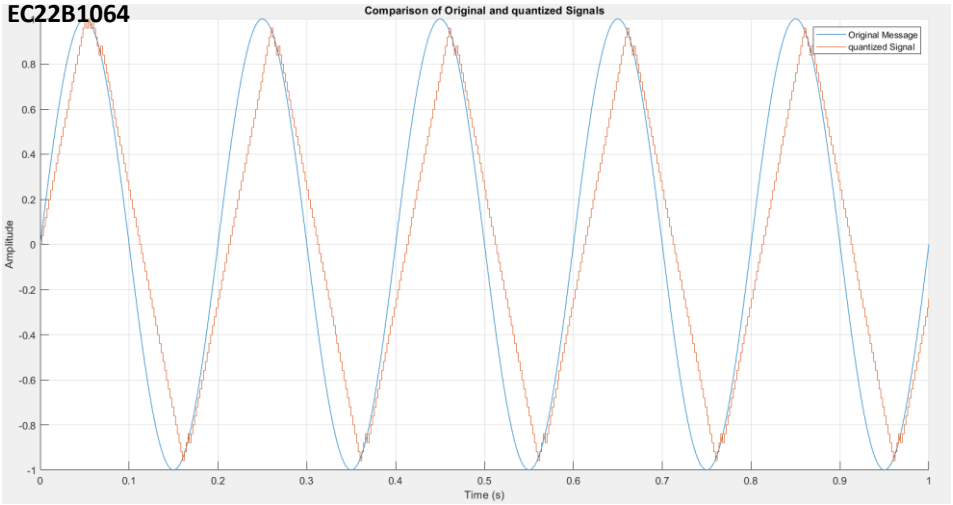
$$\frac{\Delta}{T_s} \geq \max\left(\frac{dm(t)}{dt}\right)$$

Q1) Implementation of Delta modulation and demodulation and plotting of signals with their spectra

```
fs = 1000;
t = 0:1/fs:1;
f = 5;
message_signal = sin(2*pi*f*t);
max_slope = 2*pi*f;
delta = (max_slope/fs);

delta_slope_overload = delta*0.8;
delta_granular = delta*10;

y_slope = zeros(1, length(message_signal));
y_granular = zeros(1, length(message_signal));
encoded_slope = zeros(length(message_signal));
encoded_granular = zeros(length(message_signal));

for i = 2:length(message_signal)

    if message_signal(i) >= y_slope(i-1)
        y_slope(i) = y_slope(i-1) + delta_slope_overload;
        encoded_slope(i) = 1;
    else
        y_slope(i) = y_slope(i-1) - delta_slope_overload;
    end

    if message_signal(i) >= y_granular(i-1)
        y_granular(i) = y_granular(i-1) + delta_granular;
        encoded_granular(i) = 1;
    else
        y_granular(i) = y_granular(i-1) - delta_granular;
        encoded_granular(i) = 0;
    end
end


figure;
subplot(3,1,1);
plot(t, message_signal);
title('Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3,1,2);
plot(t, y_slope);
title('Quantized Signal (Slope Overload Distortion)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3,1,3);
plot(t, y_granular);
title('Quantized Signal (Granular Noise)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```



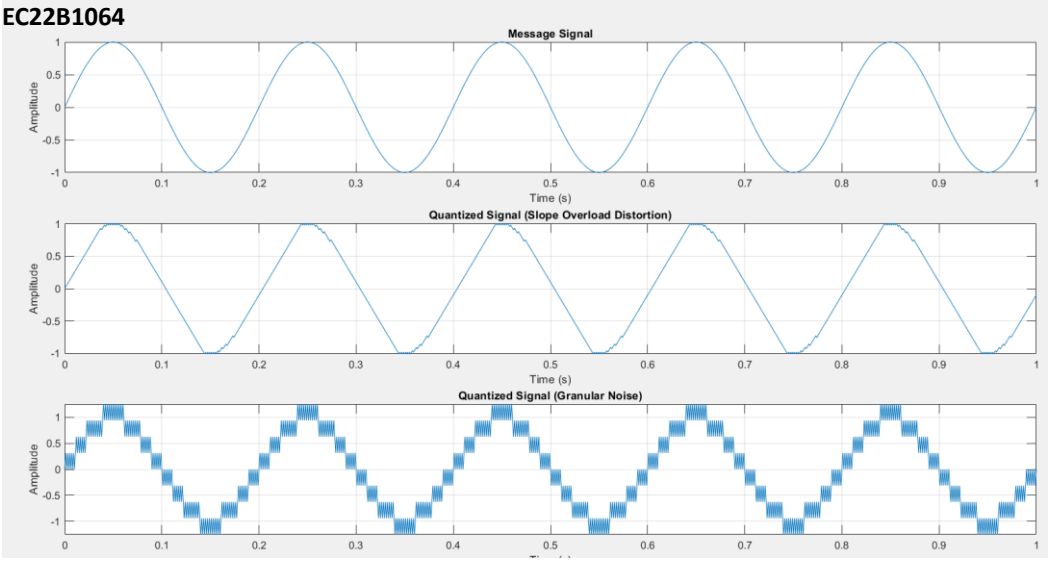
EC22B1064



Q2)slope overload distortion and granular noise

```
fs = 1000;
t = 0:1/fs:1;
f = 5;
message_signal = sin(2*pi*f*t);
max_slope = 2*pi*f;
delta = (max_slope/fs);

delta_slope_overload = delta*0.8;
delta_granular = delta*10;

y_slope = zeros(1, length(message_signal));
y_granular = zeros(1, length(message_signal));
encoded_slope = zeros(length(message_signal));
encoded_granular = zeros(length(message_signal));

for i = 2:length(message_signal)

    if message_signal(i) >= y_slope(i-1)
        y_slope(i) = y_slope(i-1) + delta_slope_overload;
        encoded_slope(i) = 1;
    else
        y_slope(i) = y_slope(i-1) - delta_slope_overload;
        encoded_slope(i) = 0;
    end

    if message_signal(i) >= y_granular(i-1)
        y_granular(i) = y_granular(i-1) + delta_granular;
        encoded_granular(i) = 1;
    else
        y_granular(i) = y_granular(i-1) - delta_granular;
        encoded_granular(i) = 0;
    end
end


figure;
subplot(3,1,1);
plot(t, message_signal);
title('Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3,1,2);
plot(t, y_slope);
title('Quantized Signal (Slope Overload Distortion)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(3,1,3);
plot(t, y_granular);
title('Quantized Signal (Granular Noise)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

EC22B1064



Inference:

- **Delta Modulation Working:** DM effectively approximates a sinusoidal message signal using a stepwise approach, encoding only the difference between successive samples.
- **Effect of Step Size (Δ):** A small Δ leads to slope overload distortion, while a large Δ results in granular noise. Choosing an optimal Δ minimizes errors.
- **Reconstruction Accuracy:** The demodulated signal closely resembles the original signal when the correct step size and sampling rate are used.
- **Spectral Analysis:** The frequency domain representation of the message, encoded, quantized, and decoded signals shows the impact of modulation and demodulation.

Conclusion:

The Delta Modulation experiment successfully demonstrated the encoding and decoding of a sinusoidal signal. By selecting an appropriate step size, we minimized slope overload distortion and granular noise. The reconstructed signal retained the primary characteristics of the original signal, verifying the effectiveness of Delta Modulation in digital communication.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

EXPERIMENT – 5

NOISE REALIZATION

February 18 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To generate and analyze a Gaussian random process in MATLAB, compute its auto-correlation function, and study the frequency spectrum of the auto-correlation function. Additionally, to evaluate the effect of noise on a message signal with different amplitude values, analyze its auto-correlation, and observe the corresponding frequency spectra.

Theory:

NOISE REALIZATION:

Gaussian Random Process:

• A Gaussian random process has a normal distribution with mean (μ) and variance (σ^2).

• The power spectral density of white noise is given by:

$$S_{x(f)} = \frac{N_0}{2}$$

Autocorrelation Function:

• Measures signal similarity at different time shifts:

$$R\_x(\tau) = E[X(t)X(t+\tau)]$$

• The frequency spectrum of the autocorrelation function is obtained via Fourier Transform.

Message Signal with Noise:

• The message signal is given by:

$$x(t) = A\cos(2\pi f_c t + \theta) + w(t)$$

where θ ~ U(-π, π) and w(t) is white noise.

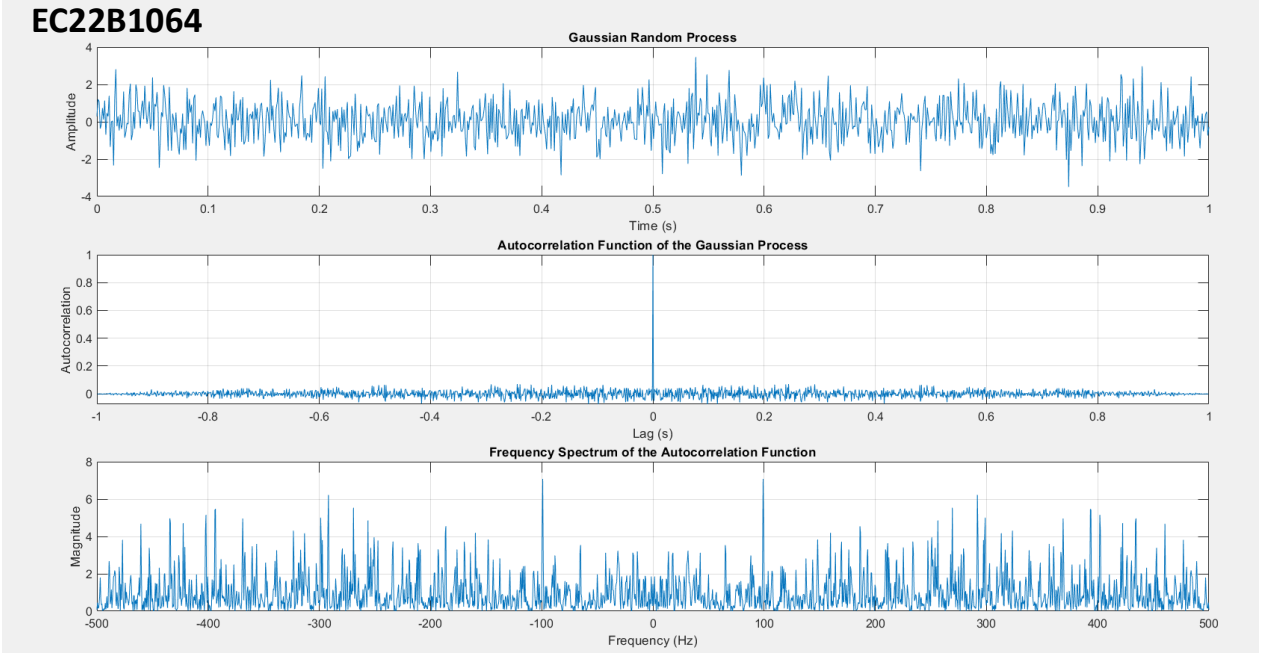
• The effect of noise on the signal is analyzed for different amplitudes (A = 10, 1, 0.1).

Q1) Plotting the Gaussian random process and its auto-correlation function and its frequency spectrum

```
N = 1000;
T = 1;
fs = N/T;
t = linspace(0, T, N);
x =randn(1, N);
figure;
plot(t, x);
title('Gaussian Random Process');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

```
[Rxx, lags] = xcorr(x, 'normalized');
tau = lags / fs;
figure;
plot(tau, Rxx);
title('Autocorrelation Function of the Gaussian Process');
xlabel('Lag (s)');
ylabel('Autocorrelation');
grid on;
Rxx_fft = abs(fftshift(fft(Rxx)));
freq = linspace(-fs/2, fs/2, length(Rxx));
figure;
plot(freq, Rxx_fft);
title('Frequency Spectrum of the Autocorrelation Function');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
```

EC22B1064



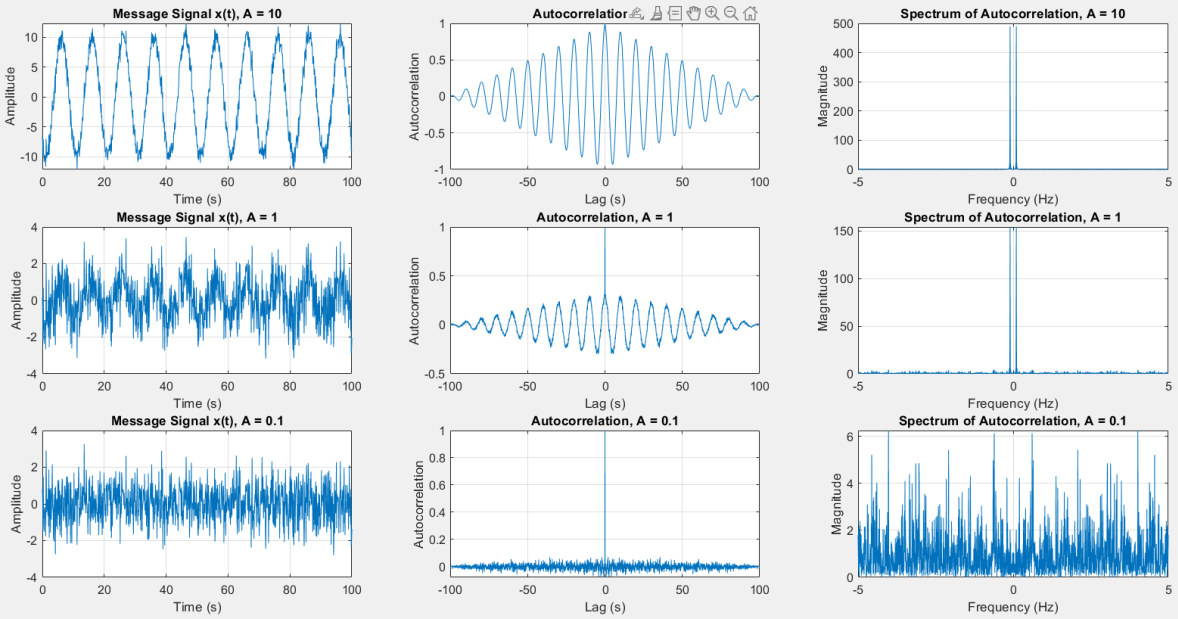
Q2) Plotting auto-correlations functions and spectra for the given equation at different values of the Amplitude

```
N = 1000;
T = 100;
fs = N / T;
t = linspace(0, T, N);
fc = 100;
w = randn(1, N);
theta = -pi + (2 * pi) * rand;
A_values = [10, 1, 0.1];
figure;
for i = 1:length(A_values)
    A = A_values(i);
    x = A * cos(2 * pi * fc * t + theta) + w;
```

```
    subplot(3, 3, (i - 1) * 3 + 1);
    plot(t, x);
    title(['Message Signal x(t), A = ', num2str(A)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;

    % Compute and plot autocorrelation function
    [Rxx, lags] = xcorr(x, 'normalized');
    tau = lags / fs;

    subplot(3, 3, (i - 1) * 3 + 2);
    plot(tau, Rxx);
    title(['Autocorrelation, A = ', num2str(A)]);
    xlabel('Lag (s)');
    ylabel('Autocorrelation');
    grid on;
```

```
    % Compute and plot frequency spectrum of the autocorrelation function
    Rxx_fft = abs(fftshift(fft(Rxx)));
    freq = linspace(-fs/2, fs/2, length(Rxx));

    subplot(3, 3, (i - 1) * 3 + 3);
    plot(freq, Rxx_fft);
    title(['Spectrum of Autocorrelation, A = ', num2str(A)]);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    grid on;
end

sgtitle('Message Signal, Autocorrelation, and Spectrum for Different Amplitudes');
```

EC22B1064



Inference:

• The Gaussian random process exhibits random variations but follows a normal distribution.

• The autocorrelation function shows the degree of similarity between signal instances over time.

• The power spectrum analysis reveals the frequency distribution of noise and signal components.

• The message signal's shape changes with different A values, affecting its autocorrelation and frequency spectrum.

Conclusion:

The experiment successfully demonstrated noise realization and its impact on signal characteristics. The autocorrelation and spectral analysis provided insights into how noise alters a message signal, emphasizing the significance of amplitude variations in signal processing.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

□

EXPERIMENT – 6

Binary-ASK And M-ary ASK

March 10 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement Binary Amplitude Shift Keying (BASK) and M-ary ASK (M = 4, 8) for a given binary message sequence. Verify the decoded bits at the receiving end and plot the corresponding waveforms.

Theory:

Amplitude Shift Keying (ASK)

is a digital modulation technique where the amplitude of a carrier signal varies according to the digital data. The frequency and phase of the carrier remain constant while the amplitude changes to represent binary data.

1. **Binary ASK (BASK):**
   - Also known as On-Off Keying (OOK), BASK uses two distinct amplitude levels to represent binary '1' and '0'.
   - The modulated BASK signal is given by: $s(t) = b(t).A_c\cos(2\pi f_c t)$
     where is the carrier amplitude, is the carrier frequency, and is the binary data signal.
2. **M-ary ASK:**
   - Extends BASK by allowing M different amplitude levels, where $M = 2^k$ and k is the number of bits per symbol.
   - The modulated M-ASK signal is expressed as: $s(t) = A_m\cos(2\pi f_c t)$
     where takes one of M discrete amplitude values.
   - Example:
     4-ASK: Uses 4 amplitude levels to represent 2 bits per symbol.
     8-ASK: Uses 8 amplitude levels to represent 3 bits per symbol.

Advantages of M-ASK:

Higher data rates as multiple bits are transmitted per symbol.

Improved bandwidth efficiency compared to BASK.

Disadvantages of M-ASK:

Increased complexity in modulation and demodulation.

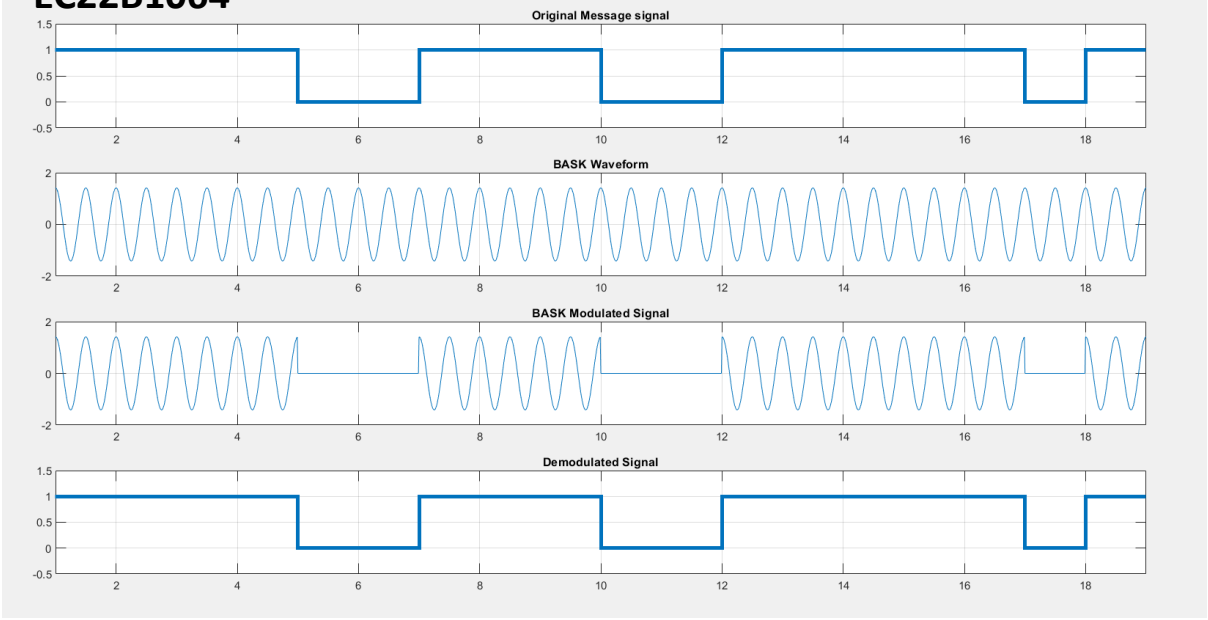
Higher noise susceptibility as M increases, leading to greater chances of symbol errors.

Q1)Perform Binary Amplitude Shift Keying (BFSK) and decode the signal.

```
fs=100;
fc = 2;
y = zeros(1,20);
Tb = 1;
Eb = 1;
N = length(y);
t = 0:1/fs:N*Tb-(1/fs);
y(1:5) = 1;
y(8:10) = 1;
y(13:17) = 1;
y(19:20) = 1;
subplot(4,1,1)
stairs(0:N-1,y,LineWidth=3);
ylim([-0.5, 1.5]);
xlim([1,19]);
grid on;
title("Original Message signal");
c = sqrt(2*Eb/Tb).*cos(2*pi*fc*t);
subplot(4,1,2)
plot(t,c);

title("BASK Waveform");
xlim([1,19]);
grid on;
s = zeros(1,length(t));
for i = 1:N
    if y(i) == 1
        s((i-1)*fs*Tb+1:i*fs*Tb) = sqrt(2*Eb/Tb) * cos(2*pi*fc*t((i-1)*fs*Tb+1:i*fs*Tb));
    end
end
subplot(4,1,3)
plot(t,s);
title("BASK Modulated Signal");
xlim([1,19]);
grid on;

N1 = length(s);
dem = zeros(1, N);

for i = 1:N
    % Extract signal segment corresponding to the i-th bit
    segment = s((i-1)*fs*Tb+1:i*fs*Tb);

    % Compute the envelope (absolute value)
    avg_amplitude = mean(abs(segment));


    if avg_amplitude >= 0.5
        dem(i) = 1;
    else
        dem(i) = 0;
    end
end

subplot(4,1,4)
stairs(0:N-1, dem, Linewidth=3);
```

```
ylim([-0.5, 1.5]);
xlim([1,19]);
grid on;
title("Demodulated Signal");
```

EC22B1064



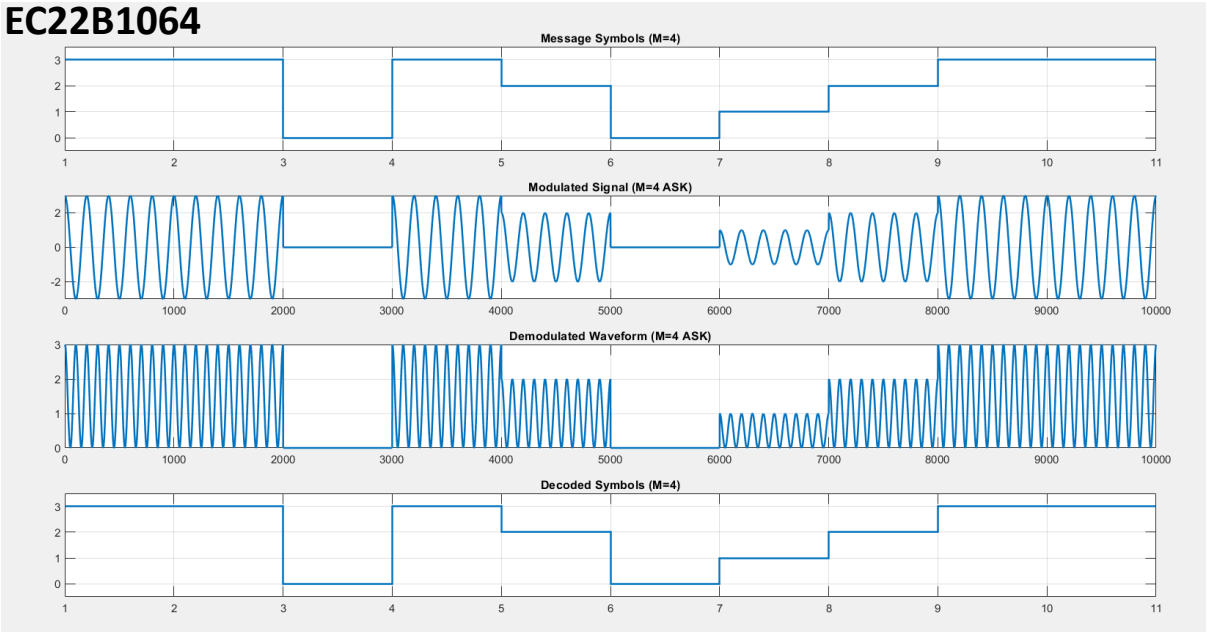
Q2) Perform M-ary Amplitude Shift Keying (M=4, M=8) and decode the signal.

```
fs = 1000;
Tb = 1;
fc = 5;
t = 0:1/fs:Tb-1/fs;
symbols_4 = [0 1 2 3];
symbols_8 = [0 1 2 3 4 5 6 7];

message_4 = randi([0 3], 1, 10);
message_8 = randi([0 7], 1, 10);

carrier = cos(2*pi*fc*t);

modulated_4 = [];
for i = 1:length(message_4)
    modulated_4 = [modulated_4, symbols_4(message_4(i)+1) * carrier];
end

modulated_8 = [];
for i = 1:length(message_8)
    modulated_8 = [modulated_8, symbols_8(message_8(i)+1) * carrier];
end

demodulated_wave_4 = modulated_4 .* repmat(carrier, 1, length(message_4));
```

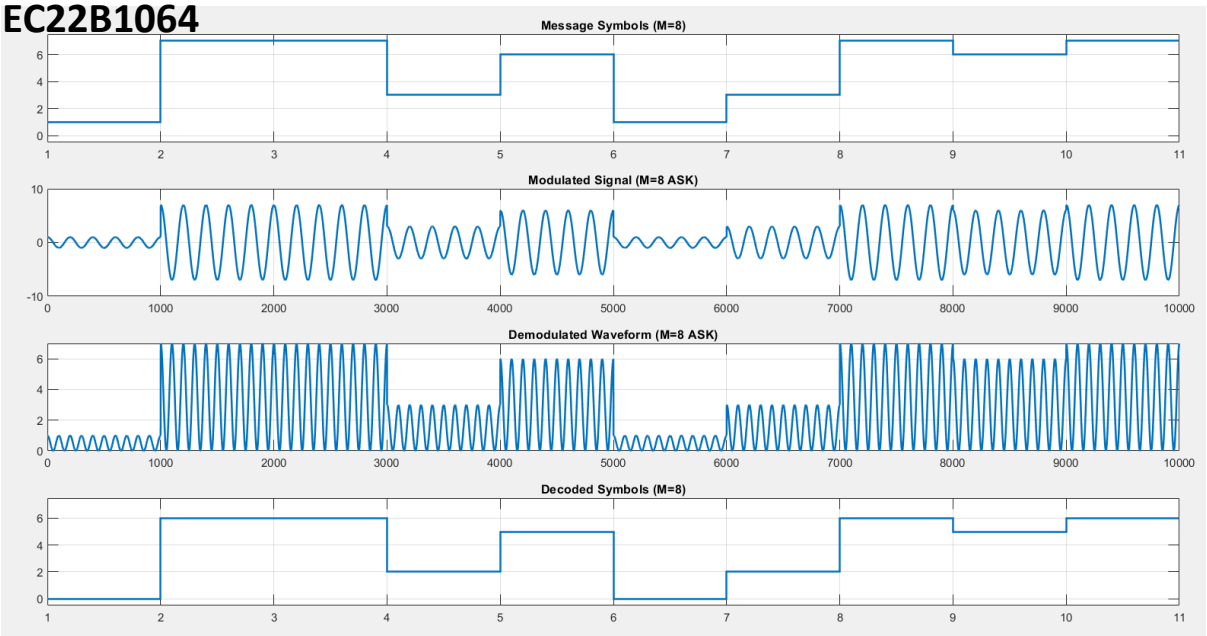
```
demodulated_wave_8 = modulated_8 .* repmat(carrier, 1, length(message_8));

demodulated_4 = [];
for i = 1:length(message_4)
    received = demodulated_wave_4((i-1)*length(t) + 1 : i*length(t));
    demodulated_4 = [demodulated_4, trapz(t, received)];
end

demodulated_8 = [];
for i = 1:length(message_8)
    received = demodulated_wave_8((i-1)*length(t) + 1 : i*length(t));
    demodulated_8 = [demodulated_8, trapz(t, received)];
end

decoded_4 = round((demodulated_4 - min(demodulated_4)) / (max(demodulated_4) / 3));
decoded_8 = round((demodulated_8 - min(demodulated_8)) / (max(demodulated_8) / 7));

figure;
subplot(4,1,1);
stairs([message_4 message_4(end)], 'LineWidth', 1.5);
ylim([-0.5 3.5]);
title('Message Symbols (M=4)');

 grid on;

 subplot(4,1,2);
 plot(modulated_4, 'LineWidth', 1.5);
 title('Modulated Signal (M=4 ASK)');
 grid on;

 subplot(4,1,3);
 plot(demodulated_wave_4, 'LineWidth', 1.5);
 title('Demodulated Waveform (M=4 ASK)');
 grid on;

 subplot(4,1,4);
 stairs([decoded_4 decoded_4(end)], 'LineWidth', 1.5);
 ylim([-0.5 3.5]);
 title('Decoded Symbols (M=4)');
 grid on;

 figure;
 subplot(4,1,1);
 stairs([message_8 message_8(end)], 'LineWidth', 1.5);
 ylim([-0.5 7.5]);
 title('Message Symbols (M=8)');

 grid on;

 subplot(4,1,2);
 plot(modulated_8, 'LineWidth', 1.5);
 title('Modulated Signal (M=8 ASK)');
 grid on;

subplot(4,1,3);
plot(demodulated_wave_8, 'LineWidth', 1.5);
title('Demodulated Waveform (M=8 ASK)');
grid on;
```

```
subplot(4,1,4);
stairs([decoded_8 decoded_8(end)], 'LineWidth', 1.5);
ylim([-0.5 7.5]);
title('Decoded Symbols (M=8)');
grid on;
```

EC22B1064



EC22B1064



Inference:

- In BASK, the carrier is switched on and off, making it simple but prone to noise.
- In M-ASK, multiple amplitude levels allow transmission of more data per symbol, increasing efficiency but requiring better signal processing.
- As M increases, the system becomes more complex and susceptible to noise

Conclusion:

- ASK is an efficient modulation technique for digital communication, widely used in optical fiber and wireless communication.
- M-ASK improves data transmission rates but requires better signal-to-noise ratio and more complex demodulation techniques.
- The experiment successfully demonstrates BASK and M-ASK, verifying the transmitted and received signals.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

□

EXPERIMENT – 7

BINARY FREQUENCY SHIFT KEYING BFSK

March 18 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement Binary Frequency Shift Keying (BFSK) on a specified binary message sequence, decode the signal, and compare it with the original message. Additionally, analyze the effect of noise on the BFSK demodulation process and plot the corresponding waveforms.

Theory:

Binary Frequency Shift Keying (BFSK):

Binary Frequency Shift Keying (BFSK) is a digital modulation technique where the frequency of the carrier signal changes based on the binary data. The two frequencies represent binary '1' and '0'. The amplitude and phase of the carrier remain constant.

Mathematical Representation

The BFSK modulated signal is given by:

$$s(t) = \begin{cases} A\cos(2\pi f_1 t) & if\ b(t) = 1 \\ A\cos(2\pi f_2 t) & if b(t) = 0 \end{cases}$$

where:

- A is the carrier amplitude,
- f_1 and f_2 are the two different carrier frequencies,
- $b(t)$ is the binary message signal.

Advantages of BFSK:

- Less susceptible to noise compared to ASK.
- Suitable for non-coherent demodulation.

Disadvantages of BFSK:

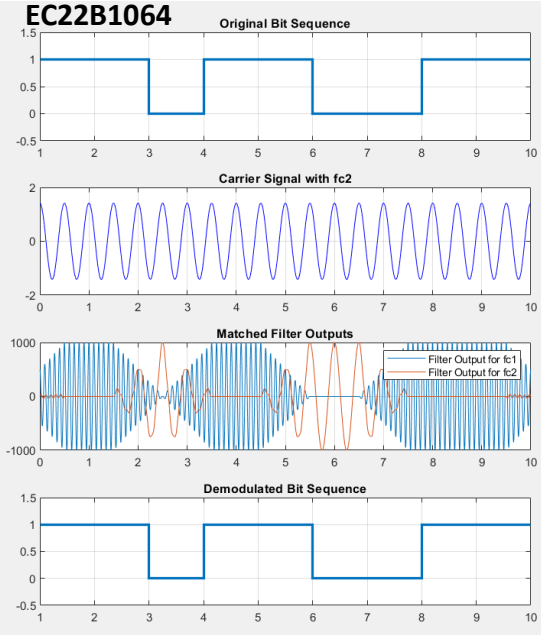
- Requires a wider bandwidth compared to ASK.
- More complex circuitry compared to BASK.

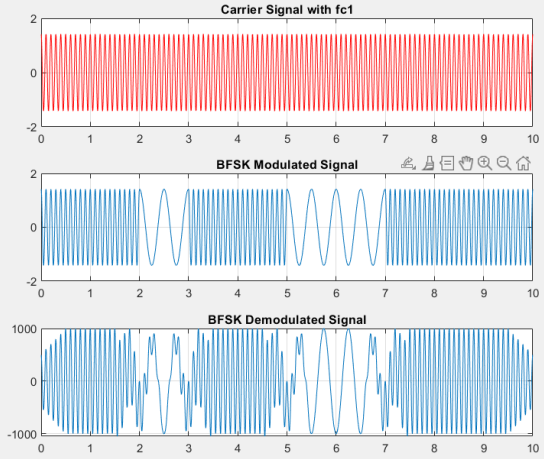
Q1)Perform Binary Frequency Shift Keying (BFSK) and decode the signal.

```
clc; clear; close all;
fs = 1000;
Tb = 1;
Eb = 1;
fc1 = 10;
fc2 = 2;
N = 10;

bits = randi([0 1], 1, N);

t = 0:1/fs:Tb-1/fs;
t_total = 0:1/fs:N*Tb-1/fs;

A = sqrt(2*Eb/Tb);
carrier1 = A * cos(2*pi*fc1*t);
carrier1_ = A * cos(2*pi*fc1*t_total);
carrier2 = A * cos(2*pi*fc2*t);
carrier2_ = A * cos(2*pi*fc2*t_total);

matched_filter1 = fliplr(carrier1);
matched_filter2 = fliplr(carrier2);


modulated_signal = [];
for i = 1:N
    if bits(i) == 1
        modulated_signal = [modulated_signal, carrier1];
    else
        modulated_signal = [modulated_signal, carrier2];
    end
end

filtered_signal1 = conv(modulated_signal, matched_filter1, 'same');
filtered_signal2 = conv(modulated_signal, matched_filter2, 'same');

received_bits = zeros(1, N);

for i = 1:N
    index_start = (i-1)*length(t) + 1;
    index_end = i*length(t);

    y1 = sum(filtered_signal1(index_start:index_end).^2);
    y2 = sum(filtered_signal2(index_start:index_end).^2);

    if (y1-y2) > 0


        received_bits(i) = 1;
    else
        received_bits(i) = 0;
    end
end

figure;
subplot(4,2,1);
stairs(bits, 'linewidth', 2);
```

```
ylim([-0.5 1.5]);
title('Original Bit Sequence');
grid on;

subplot(4,2,2);
plot(t_total, carrier1_, 'r');
title('Carrier Signal with fc1');
grid on;

subplot(4,2,3);
plot(t_total, carrier2_,'b');
title('Carrier Signal with fc2');
grid on;

subplot(4,2,4);
plot(t_total, modulated_signal);
title('BFSK Modulated Signal');
grid on;

subplot(4,2,5);
plot(t_total, filtered_signal1); hold on;
plot(t_total, filtered_signal2); hold off;
title('Matched Filter Outputs');
legend('Filter Output for fc1', 'Filter Output for fc2');
grid on;

subplot(4,2,6);
plot(t_total, filtered_signal1 - filtered_signal2);
title('BFSK Demodulated Signal');
grid on;

subplot(4,2,7);
stairs(received_bits, 'linewidth', 2);
ylim([-0.5 1.5]);
title('Demodulated Bit Sequence');
grid on;
```

EC22B1064



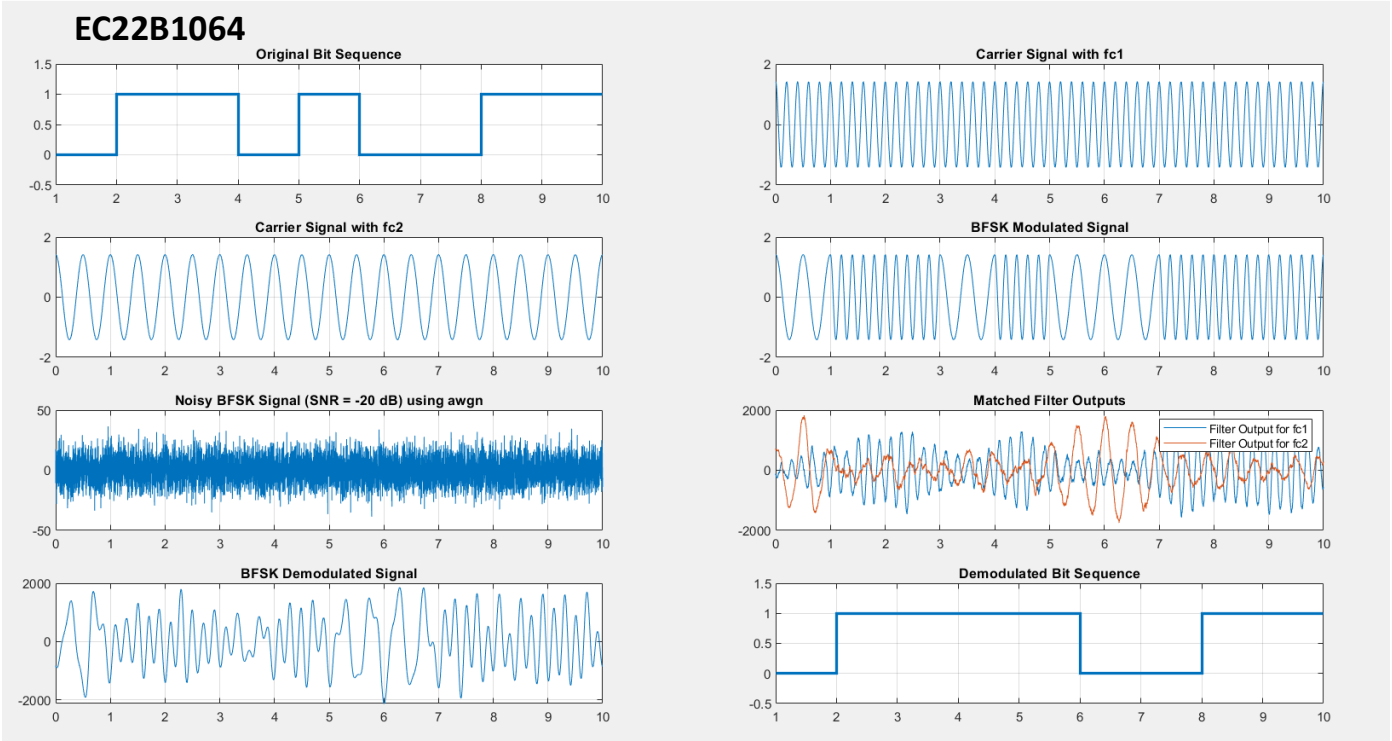


Q2) Analyze the effect of noise on BFSK demodulation process by adding AWGN to the BFSK Modulated signal

```
clc; clear; close all;

fs = 1000;
Tb = 1;
Eb = 1;
fc1 = 5;
fc2 = 2;
N = 10;
SNR_dB = -20;

bits = randi([0 1], 1, N);

t = 0:1/fs:Tb-1/fs;
t_total = 0:1/fs:N*Tb-1/fs;

A = sqrt(2*Eb/Tb);
carrier1 = A * cos(2*pi*fc1*t);
carrier1_ = A * cos(2*pi*fc1*t_total);
carrier2 = A * cos(2*pi*fc2*t);
carrier2_ = A * cos(2*pi*fc2*t_total);

matched_filter1 = fliplr(carrier1);
matched_filter2 = fliplr(carrier2);
modulated_signal = [];
for i = 1:N
    if bits(i) == 1
        modulated_signal = [modulated_signal, carrier1];
    else
        modulated_signal = [modulated_signal, carrier2];
    end
end

received_signal = awgn(modulated_signal, SNR_dB, 'measured');

filtered_signal1 = conv(received_signal, matched_filter1, 'same');
filtered_signal2 = conv(received_signal, matched_filter2, 'same');

received_bits = zeros(1, N);

for i = 1:N
    index_start = (i-1)*length(t) + 1;
    index_end = i*length(t);

    y1 = sum(filtered_signal1(index_start:index_end).^2);
    y2 = sum(filtered_signal2(index_start:index_end).^2);

   decision_signal(i) = y1 - y2;

   if (y1-y2) > 0
       received_bits(i) = 1;
   else
       received_bits(i) = 0;
   end
end


figure;
subplot(4,2,1);
stairs(bits, 'linewidth', 2);
ylim([-0.5 1.5]);
title('Original Bit Sequence');
grid on;
```

```
subplot(4,2,2);
plot(t_total, carrier1_');
title('Carrier Signal with fc1');
grid on;

subplot(4,2,3);
plot(t_total, carrier2_);
title('Carrier Signal with fc2');
grid on;

subplot(4,2,4);
plot(t_total, modulated_signal);
title('BFSK Modulated Signal');
grid on;

subplot(4,2,5);
plot(t_total, received_signal);
title(['Noisy BFSK Signal (SNR = ' num2str(SNR_dB) ' dB) using awgn']);
grid on;

 subplot(4,2,6);
 plot(t_total, filtered_signal1); hold on;
 plot(t_total, filtered_signal2); hold off;
 title('Matched Filter Outputs');
 legend('Filter Output for fc1', 'Filter Output for fc2');
 grid on;

 subplot(4,2,7);
 plot(t_total, filtered_signal1 - filtered_signal2);
 title('BFSK Demodulated Signal');
 grid on;

 subplot(4,2,8);
 stairs(received_bits, 'linewidth', 2);
 ylim([-0.5 1.5]);
 title('Demodulated Bit Sequence');
 grid on;
```

EC22B1064



Inference:

- In BFSK, different carrier frequencies represent binary symbols, making it more robust against noise than ASK.
- The presence of noise (AWGN) affects the demodulation process, increasing the probability of bit errors.
- Trade-off: BFSK requires more bandwidth but is more reliable in noisy environments

Conclusion:

- Binary Frequency Shift Keying (BFSK) is an efficient modulation technique used in wireless communication.
- BFSK provides better noise immunity than ASK but at the cost of increased bandwidth.
- The experiment successfully demonstrates BFSK modulation and demodulation, verifying the transmitted and received signals.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

□

EXPERIMENT – 8

BINARY PHASE SHIFT KEYING BPSK

March 25 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement Binary Phase Shift Keying (BPSK) on a specified binary message sequence, decode the signal, and compare it with the original message. Additionally, analyze the effect of noise on the BPSK demodulation process and plot the corresponding waveforms.

Theory:

BINARY PHASE SHIFT KEYING (BFSK):

Binary Phase Shift Keying (BPSK) is a digital modulation technique where the phase of the carrier signal changes based on the binary data. The carrier frequency remains constant, while the phase shifts by 180 degrees depending on whether the bit is '1' or '0'.

Mathematical Representation

The BPSK modulated signal is given by:

$$s(t) = Acos(2\pi f_c t) \quad if\ b(t) = 1$$
$$s(t) = -Acos(2\pi f_c t) \quad if\ b(t) = 0$$

where: A is the carrier amplitude, f_c is the carrier frequency, b(t) is the binary message signal.

Advantages of BPSK:

High noise immunity compared to ASK and BFSK. Simple coherent detection using matched filtering. Disadvantages of BPSK: Requires coherent detection for demodulation. More complex circuitry compared to ASK.

Q1)Perform Binary Phase Shift Keying (BFSK) and decode the signal.

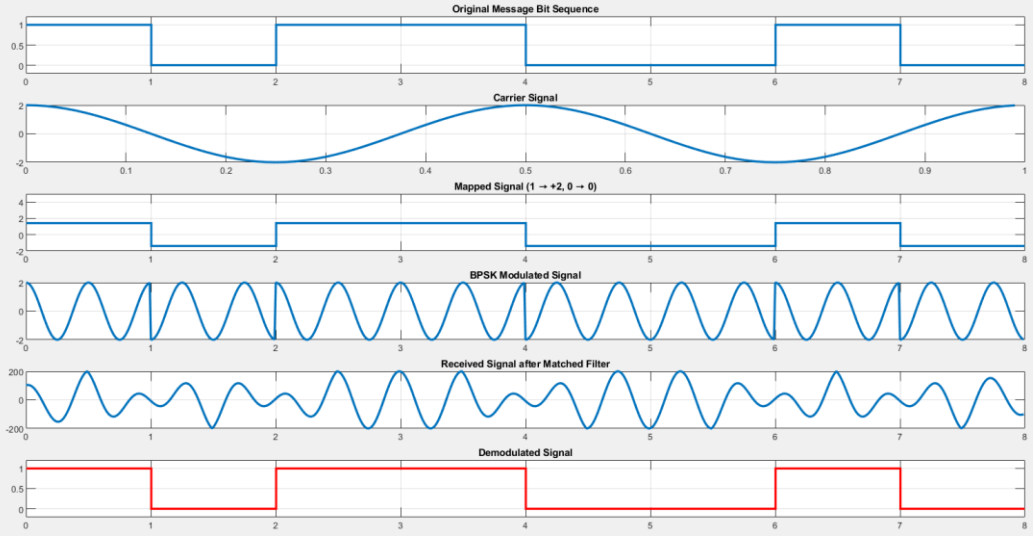
```
Tb = 1;
fc = 2;
fs = 100;
t = 0:1/fs:Tb-1/fs;
A = sqrt(2/Tb);
Eb = 2;

message = [1 0 1 1 0 0 1 0];
N = length(message);
t_total = 0:1/fs:N*Tb-1/fs;

carrier = A * sqrt(Eb) * cos(2*pi*fc*t);

mapped_signal = sqrt(Eb) * (2*message - 1);
bpsk_signal = [];
```

```
for i = 1:N
    if message(i) == 1
        bpsk_signal = [bpsk_signal, carrier];
    else
        bpsk_signal = [bpsk_signal, -carrier];
    end
end

matched_filter = fliplr(carrier);

received_signal = conv(bpsk_signal, matched_filter, 'same') ;

demodulated_bits = zeros(1, N);
for i = 1:N
    segment = received_signal((i-1)*fs+1:i*fs);
    correlation = sum(segment .* carrier);
    demodulated_bits(i) = correlation > 0;
end
```

```
figure;
subplot(6,1,1);
stairs((0:N)*Tb, [message message(end)], 'LineWidth', 2);
title('Original Message Bit Sequence'); ylim([-0.2 1.2]); grid on;

subplot(6,1,2);
plot(t, carrier,'LineWidth', 2);
title('Carrier Signal'); grid on;

subplot(6,1,3);
stairs((0:N)*Tb, [mapped_signal mapped_signal(end)], 'LineWidth', 2);
title('Mapped Signal (1 → +2, 0 → 0)'); ylim([-2 5]); grid on;

subplot(6,1,4);
plot(t_total, bpsk_signal, 'LineWidth', 2);
title('BPSK Modulated Signal'); grid on;

subplot(6,1,5);
plot(t_total, received_signal,'LineWidth', 2);
title('Received Signal after Matched Filter'); grid on;
```

```
subplot(6,1,6);
stairs((0:N)*Tb, [demodulated_bits demodulated_bits(end)], 'r', 'LineWidth', 2);
title('Demodulated Signal'); ylim([-0.2 1.2]); grid on;

disp('Original Message:');
disp(message);
disp('Demodulated Bits:');
disp(demodulated_bits);
```

EC22B1064



```
Original Message:
  Columns 1 through 7

     1     0     1     1     0     0     1

  Column 8

     0

Demodulated Bits:
  Columns 1 through 7

     1     0     1     1     0     0     1

  Column 8

     0
```

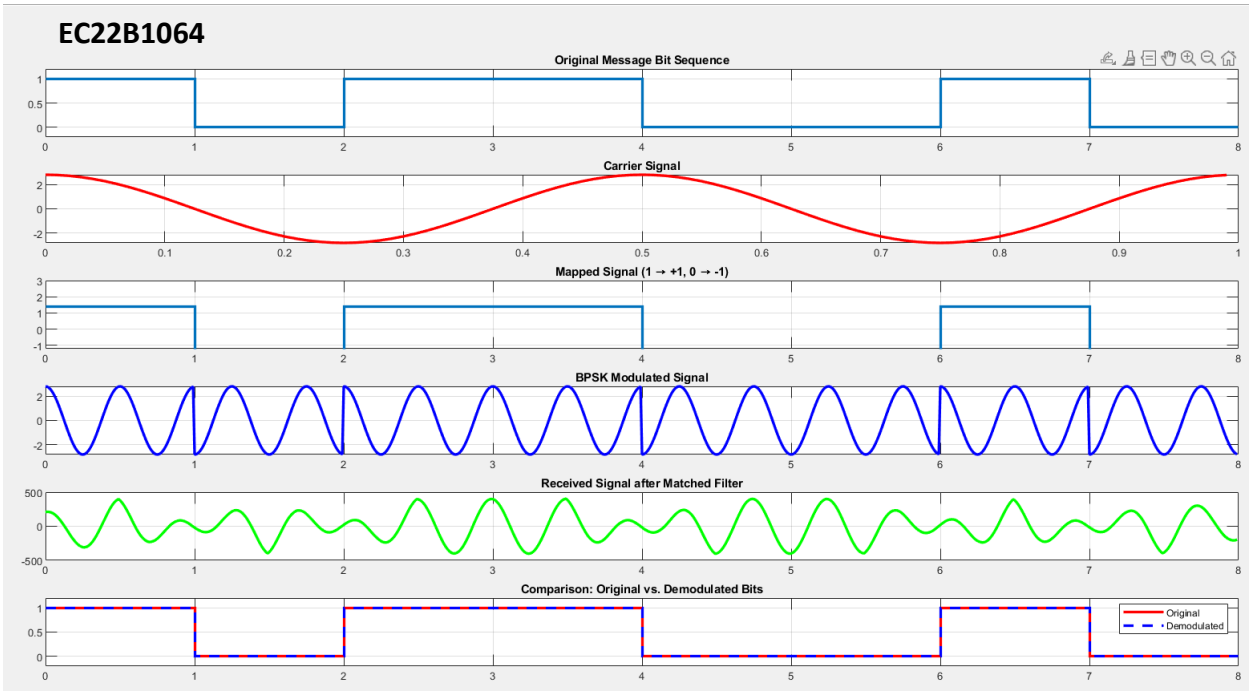
EC22B1064

Q2) Analyze the effect of noise on BPSK demodulation process by adding AWGN to the BPSK Modulated signal

```
Tb = 1;
fc = 2;
fs = 100;
t = 0:1/fs:Tb-1/fs;
A = sqrt(2/Tb);
Eb = 2;

message = [1 0 1 1 0 0 1 0];
N = length(message);
t_total = 0:1/fs:N*Tb-1/fs;

carrier = A * Eb * cos(2*pi*fc*t);

mapped_signal = sqrt(Eb) * (2*message - 1);

bpsk_signal = [];
for i = 1:N
    if message(i) == 1
        bpsk_signal = [bpsk_signal, carrier];
    else
          bpsk_signal = [bpsk_signal, -carrier];
      end
  end

  snr_db = 20;
  bpsk_signal_noisy = awgn(bpsk_signal, snr_db, 'measured');

  matched_filter = fliplr(carrier);
  received_signal = conv(bpsk_signal_noisy, matched_filter, 'same');


  demodulated_bits = zeros(1, N);
  for i = 1:N
      segment = received_signal((i-1)*fs+1:i*fs);
      correlation = sum(segment .* carrier);
      demodulated_bits(i) = correlation > 0;
  end

  figure;
  subplot(6,1,1);
```

```
stairs([0:N]*Tb, [message message(end)], 'LineWidth', 2);
title('Original Message Bit Sequence'); ylim([-0.2 1.2]); grid on;

subplot(6,1,2);
plot(t, carrier, 'r', 'LineWidth', 2);
title('Carrier Signal'); grid on;

subplot(6,1,3);
stairs([0:N]*Tb, [mapped_signal mapped_signal(end)], 'LineWidth', 2);
title('Mapped Signal (1 → +1, 0 → -1)'); ylim([-1.2 3]); grid on;

subplot(6,1,4);
plot(t_total, bpsk_signal, 'b', 'LineWidth', 2);
title('BPSK Modulated Signal'); grid on;

subplot(6,1,5);
plot(t_total, received_signal, 'g', 'LineWidth', 2);
title('Received Signal after Matched Filter'); grid on;

subplot(6,1,6);
stairs([0:N]*Tb, [message message(end)], 'r', 'LineWidth', 2); hold on;
stairs([0:N]*Tb, [demodulated_bits demodulated_bits(end)], 'b--', 'LineWidth', 2);
title('Comparison: Original vs. Demodulated Bits'); ylim([-0.2 1.2]); grid on;
legend('Original', 'Demodulated');

disp('Original Message:');
disp(message);
disp('Demodulated Bits:');
disp(demodulated_bits);
ber = sum(demodulated_bits ~= message) / N;
disp(['Bit Error Rate (BER): ', num2str(ber)]);
```

EC22B1064



Inference:

In BPSK, binary symbols are represented by phase shifts, making it highly robust against noise. The presence of noise (AWGN) affects the demodulation process, leading to potential bit errors. Trade-off: BPSK offers better noise performance but requires phase-coherent detection.

Conclusion:

Binary Phase Shift Keying (BPSK) is an efficient modulation technique widely used in digital communication. BPSK provides better noise immunity than BFSK but requires coherent demodulation. The experiment successfully demonstrates BPSK modulation and demodulation, verifying the transmitted and received signals.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

□

EXPERIMENT – 9

DIFFERENTIAL PHASE SHIFT KEYING (DPSK)

April 1 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

To implement Binary Differential Phase Shift Keying (B-DPSK) on a randomly generated binary message sequence, demodulate the noisy modulated signal, and compare it with the original message. Additionally, generate plots for various stages of the modulation and demodulation process.

Theory:

Phase Shift Keying (PSK):

PSK is a digital modulation technique where the phase of the carrier signal is varied in accordance with the message signal.

Differential Phase Shift Keying (DPSK):

In DPSK, instead of transmitting the absolute phase of the carrier, the information is conveyed by the phase difference between consecutive bits.

- **No reference signal** is required for demodulation.
- Reduces the complexity of the receiver design.
- However, **error propagation** may occur during demodulation.

Binary DPSK (B-DPSK):

- Consecutive bits with the same value → No phase change.
- Opposite bits → Phase shift of π (180°).
- The first bit is treated as a reference.

Modulation and Demodulation Logic:

- A differential encoder encodes the input message sequence.
- At the receiver, the decoder compares the current phase with the previous phase to decode the original message.

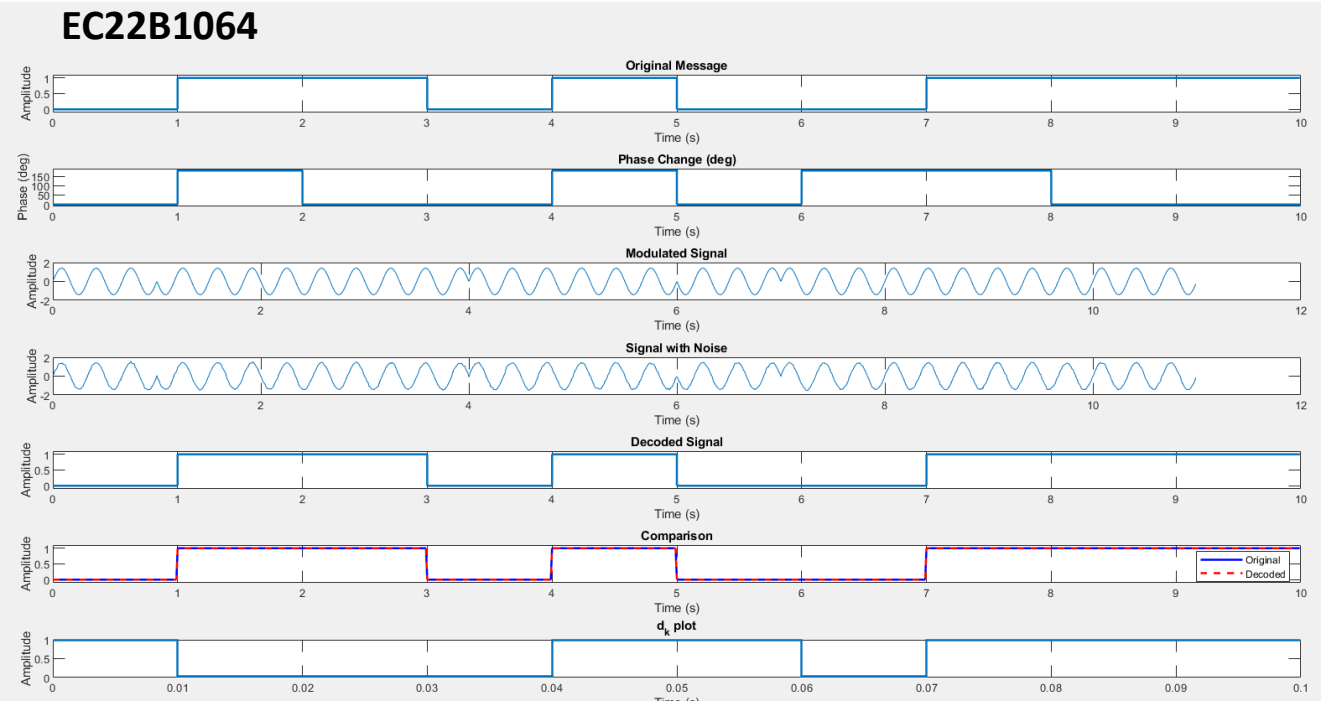
Q)Performing Binary Differential Phase Shift Keying(B-DPSK) on random message singal, demodulating the signal, performing the same operations on a noisy signal and showing the plots

```
N_bits = 10;
Fs = 100;
T = 1;

t_mod = 0:1/Fs:(N_bits+1)*T-1/Fs;
t_msg = 0:1/Fs:N_bits*T-1/Fs;

mk = [0 1 1 0 1 0 0 1 1 1];
d = zeros(1, N_bits+1);
d(1) = 1;

for k = 1:N_bits
    if mk(k) == 0
        d(k+1) = ~d(k);
    else
        d(k+1) = d(k);
    end
end

signal = ones(size(d));
signal(d==0) = -1;


phase = zeros(size(d));
phase(1) = 0;
for k = 2:length(d)
    if d(k) ~= d(k-1)
        phase(k) = 180;    % 180° change when bit flips
    else
        phase(k) = 0;      % No change when bit stays same
    end
end


 fc = 3;
 norm = sqrt(2/T);
 carrier = norm * sin(2*pi*fc*t_mod);
 mod_signal = repelem(signal, Fs*T) .* carrier;


snr = 25;
noisy_signal = awgn(mod_signal,snr, 'measured');

demod = noisy_signal .* carrier;
integrated = sum(reshape(demod, Fs*T, []))/(Fs*T);
received = ones(size(integrated));
received(integrated < 0) = 0;
```

```
decoded = zeros(1, length(received)-1);
for k = 1:length(decoded)
    if received(k+1) == received(k)
        decoded(k) = 1;
    else
        decoded(k) = 0;
    end
end

msg_plot = repelem(mk, Fs*T);
dec_plot = repelem(decoded, Fs*T);
phase_plot = repelem(phase, Fs*T);

figure
subplot(7,1,1)
stairs(t_msg, msg_plot,'LineWidth', 1.5)
title('Original Message')
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.1 1.1])

subplot(7,1,2)
stairs(t_mod, phase_plot,'LineWidth', 1.5)
title('Phase Change (deg)')
xlabel('Time (s)')
ylabel('Phase (deg)')
ylim([-10 190])
xlim([0 10])

subplot(7,1,3)
plot(t_mod, mod_signal)
title('Modulated Signal')
xlabel('Time (s)')
ylabel('Amplitude')

subplot(7,1,4)
plot(t_mod, noisy_signal)
title('Signal with Noise')
xlabel('Time (s)')
ylabel('Amplitude')

subplot(7,1,5)
stairs(t_msg, dec_plot(1:N_bits*Fs*T), 'LineWidth', 1.5)
title('Decoded Signal')
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.1 1.1])

subplot(7,1,6)
plot(t_msg, msg_plot, 'b', t_msg, dec_plot(1:N_bits*Fs*T), 'r--', 'LineWidth', 1.5)
legend('Original', 'Decoded')
title('Comparison')
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.1 1.1])
```

```
subplot(7,1,7)
stairs(t_msg(1:length(d)),d, 'LineWidth',1.5);
xlabel("Time (s)")
ylabel("Amplitude")
title("d_k plot")

disp(d);
```

EC22B1064



Inference:

Differential encoding in B-DPSK allows demodulation without a reference phase, which simplifies receiver circuitry. Noise can still impact phase differences between bits, potentially causing bit errors due to error propagation, especially at very low SNRs. However, it provides a robust alternative to conventional PSK for systems where carrier phase synchronization is difficult.

Conclusion:

The experiment successfully demonstrates B-DPSK modulation and demodulation. The decoded output matches the original message with slight variations in the presence of noise. B-DPSK is particularly useful in scenarios where coherent detection is not feasible, offering simplified receiver design at the cost of occasional error propagation.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).

EXPERIMENT – 10

DESIGN OF MICROSTRIP LINE USING ANSYS HFSS

April 8 - 2025

B Siddharth Sekhar - EC22B1064

Aim:

Design and Simulation of Microstrip Line using ANSYSS HFSS

1. Generate plots for S11 and S21
2. Calculate the Power reflected and transmitted through the line.

Theory:

A MICROSTRIP LINE

is a type of planar transmission line used in RF and microwave circuits. It consists of:

- A conductive strip (signal line) on top,
- A dielectric substrate (like FR4),
- And a ground plane at the bottom.

It is commonly used for signal routing and RF component integration (e.g., antennas, filters, couplers).

CHARACTERISTIC IMPEDANCE (Z_0)

• The characteristic impedance (Z_o) depends on:

- Width (W) of the conductor
- Substrate height (h)
- Dielectric constant (ε_r)

• For FR4 (ε_r = 4.4, h = 1.6 mm), to achieve Z_o = 50 Ω, the conductor width W ≈ 3.06 mm.

• Z_o is independent of length (L).

S-PARAMETERS (SCATTERING PARAMETERS)

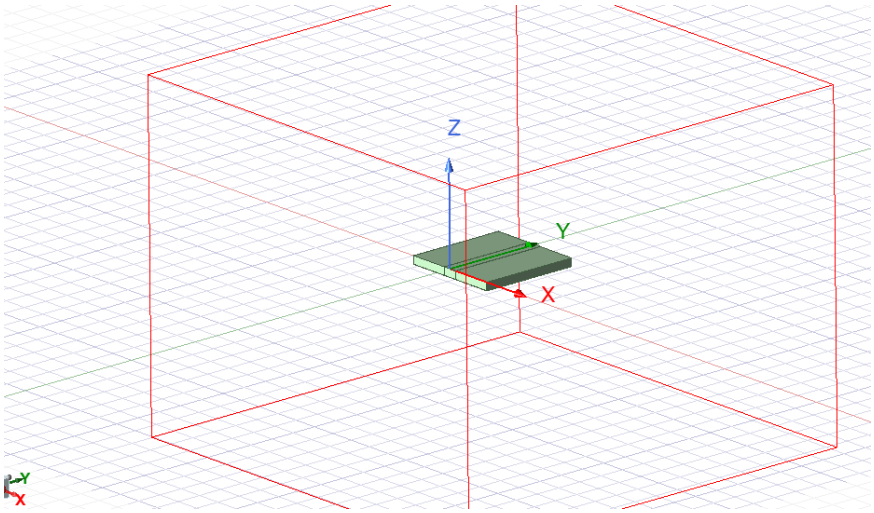
- S-parameters describe how RF signals behave in a network:
- S11 (Return Loss): Amount of signal reflected back from port 1.

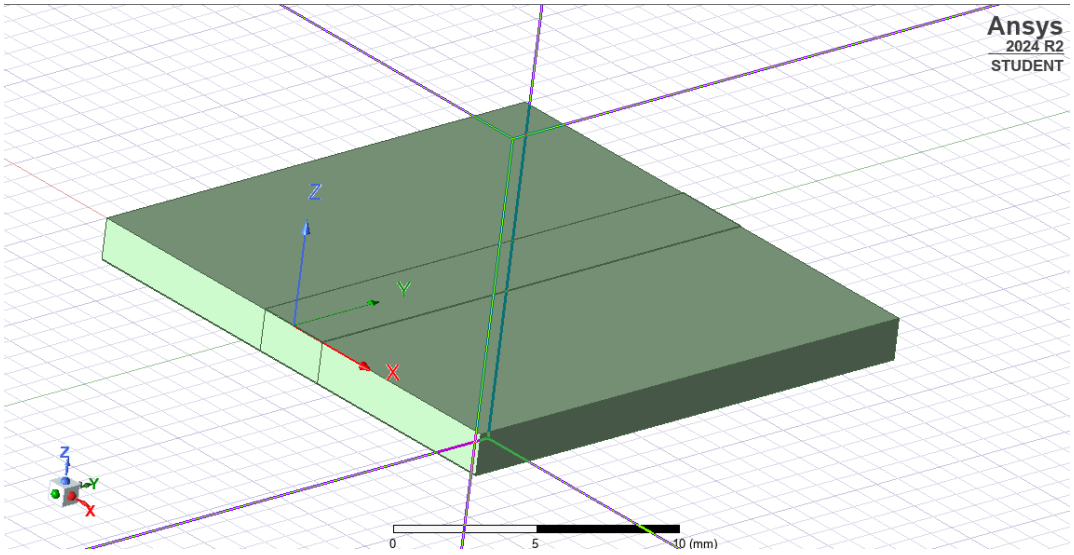
  Ideally S11 < -10 dB (less reflection, better matching).
- S21 (Insertion Loss / Transmission Gain): Amount of signal transmitted from port 1 to port 2 Ideally close to 0 dB (minimum loss).

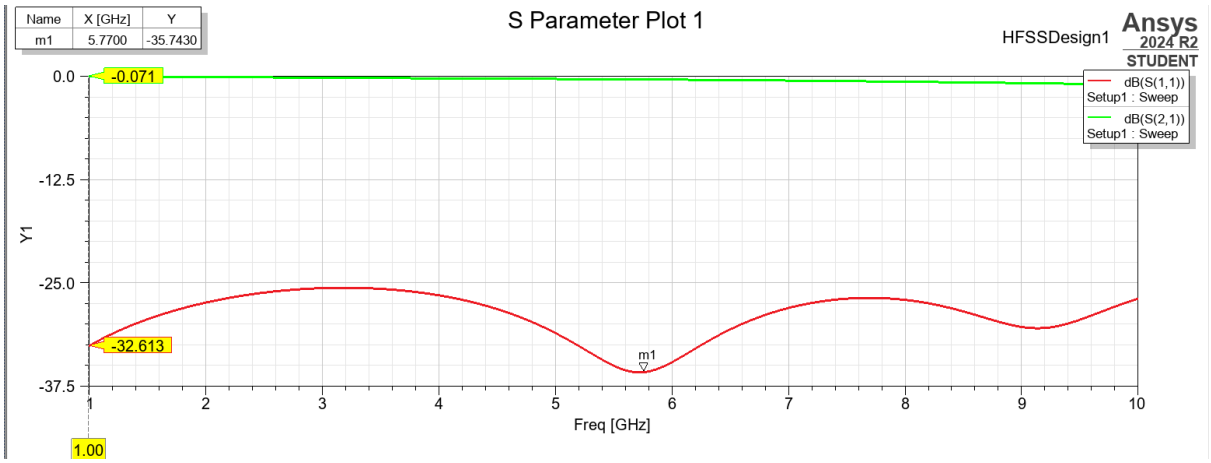
CHOOSEN DESIGN PARAMETERS

W-Width of the top conductor	3.06mm
L – Length of the top conductor	25mm
h – Height of the substrate	1.6mm
εr – Relative permittivity of substrate	εr = 4.4
Thickness (Ground,conductor)	0.035mm
substrate	25mmx25mm

Simulation Results and Observations :







From the Plot the following dB values:

- S11 @ 3 GHz = -26.75 dB
- S21 @ 3 GHz = -0.22 dB

Let's convert these to linear scale to calculate power.

Calculations:

$$P_{reflected} = |S_{11}|^2 . P_{in}$$

$$P_{transmitted} = |S_{11}|^2 . P_{in}$$

S11 = –26.75 dB :

$$|S_{11}| = 10^{(-26.75)/(20)} = 0.0425$$

$$P_{reflected} = |0.0425|^2 . 100 = 0.18\,W$$

S21 = –0.22 dB :

$$|S_{21}| = 10^{(-0.22)/(20)} = 0.9746$$

$$P_{transmitted} = |0.9746|^2 . 100 = 94.48\,W$$

Inference:

From the simulation results, the microstrip line designed with a characteristic impedance of 50 ohms

shows excellent performance at 3 GHz. The S-parameter plot indicates:

• S11 = -26.75 dB, meaning very low signal reflection (good impedance matching).

• S21 = -0.22 dB, indicating minimal transmission loss.

This confirms that the line is well-matched and eeiciently transmits power with negligible reflection.

Conclusion:

The microstrip line was successfully designed and simulated using ANSYS HFSS. The analysis of S11 and S21 parameters shows that the structure achieves proper impedance matching and low signal loss at the operating frequency. This validates the effectiveness of the design and suitability of the microstrip line for high-frequency applications.

References:

[1] Simon Haykins, Communication systems, 2nd ed. (New York John Wiley and Sons, 2005).