Communication Systems Lab(23CCE383)

Department of Electronics and Communication Engineering

Amrita VishwaVidyapeetham, Amaravati Campus

Verified by **Approved by**

General lab guidelines and safety instructions during lab session

- Carry out the experiments in such a way that nobody will be injured or hurt.
- Carry out the experiments in such a way that the equipment will not be damaged or destroyed.
- Follow all written and verbal instructions carefully. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teaching assistant.
- Never work alone! You should be accompanied by your laboratory partner and/or the instructors/teaching assistants all the time.
- Perform only those experiments you find in the instructions or authorized by the instructors.
- Unauthorized experiments are prohibited.
- The workplace has to be tidy before, during and after the experiment.
- Read the handout and procedures before starting the experiments.
- Intentional misconduct will lead to exclusion from the lab.
- Never hurry. Haste causes many accidents.
- Always see that power is connected to your equipment through a circuit breaker.
- Connect the power source last. Disconnect the power source first.
- Never make wiring changes on live circuits.
- No food or drinks are allowed in the lab.

S. No.	Name of the Experiments	Date	Signature
1	Sampling and reconstruction of an analog signal by designing pulse amplitude modulator and demodulator circuits.		
2	Application of sampling by designing time division multiplexer and demultiplexer circuits.		
3	Amplitude modulator which can be used to transmit digital information via carrier and be able to reconstruct the message signal		
4	Phase modulator which can be used to transmit the digital information via carrier and be able to reconstruct the message signal.		
5	Pulse code modulator and Delta modulator		
6	Geometric representation of the given signal using Gram-Schmidt Orthogonalization procedure implemented in MATLAB.		
7	ASK (OOK) and BPSK modulator and demodulator and BER performance comparison		
8	M-PSK and QAM modulator and demodulator and BER performance comparison		
9	To study the effects of ISI by generating an Eye pattern		
10	Specifications, characterization of Hardware platforms like NooRadio, SDR, etc.		
11	Establishment of wireless communication link using a pair of hardware platform		

Experiment 1

Sampling and reconstruction of an analog signal by designing pulse amplitude modulator and demodulator circuits.

Aim: To study the process of sampling and reconstruction of an analog signal using Pulse Amplitude Modulation (PAM) and demodulation circuits using Matlab.

Apparatus:

1. PAM Modulation Kit and Demodulation Kit
2. CRO
3. Connecting probes
4. Matlab

Algorithm for PAM Modulation and Demodulation:

1. Signal Generation & Sampling

Parameters

- Message frequency (fm = 100 Hz)
- Simulation sampling rate (fs = 10 kHz, for smooth analog representation)
- PAM sampling rate (fs_sample = 800 Hz, must satisfy Nyquist: fs_sample > 2*fm)

Signals

1. Analog Message Signal:
    - x = sin(2*pi*fm*t); % Original 100 Hz sine wave
2. Sampled Signal:
    - n = 0:Ts:0.05;   % Sampling instants (Ts = 1/fs_sample)
      x_sampled = sin(2*pi*fm*n); % Discrete-time samples

2. PAM Modulation (Sample-and-Hold)

1. Initialize PAM Signal:

   - pam_signal = zeros(size(t)); % Same length as analog time vector

2. Generate PAM Waveform:

   - For each sampling interval [n(i), n(i+1)]:

     - Hold the sampled value x_sampled(i):

     - idx = t >= n(i) & t < n(i+1);
       pam_signal(idx) = x_sampled(i);

   - Output: Staircase-like PAM signal (sample-and-hold).

3. Demodulation (Signal Reconstruction)

Ideal Sinc Interpolation

1. Reconstruct Signal:

   - x_reconstructed = zeros(size(t));
     for i = 1:length(n)
        x_reconstructed = x_reconstructed + x_sampled(i) * sinc(fs_sample*(t - n(i)));
     end
   - Key Idea: Each sample contributes a sinc function centered at its sampling instant.

   - Nyquist Criterion: Perfect reconstruction is possible if fs_sample > 2*fm.

4. Visualization

Subplot 1: Original Analog Signal

- Pure 100 Hz sine wave.

Subplot 2: Sampled Signal

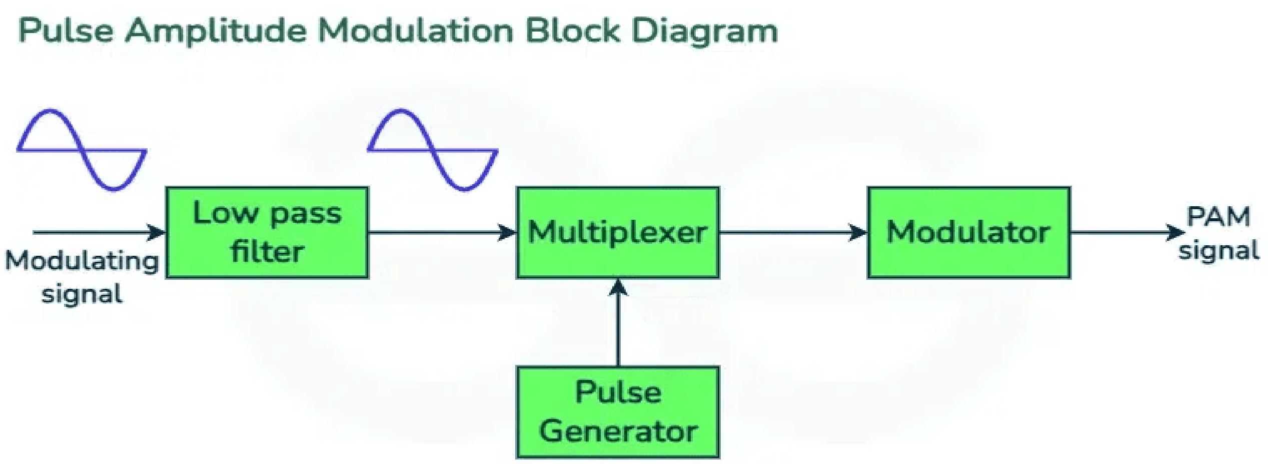
- Discrete samples (red stems) at fs_sample = 800 Hz.

Subplot 3: PAM Modulated Signal

- Sample-and-hold waveform (staircase).

Subplot 4: Reconstructed Signal

- Green: Reconstructed signal via sinc interpolation.
- Blue Dashed: Original signal for comparison.



PROCEDURE:-

1. Connections must be given as per the diagram.
2. Low frequency message signal is given as one input to PAM modulator.
3. Carrier pulse signal is given as another input to PAM modulator.
4. The pulse amplitude modulated waveform obtained is viewed in CRO.
5. Readings are taken for message,carrier and pulse amplitude modulated wave.
6. The modulated wave is given as input to demodulator
7. The demodulated output is noted in CRO.

Procedure (Using MATLAB):

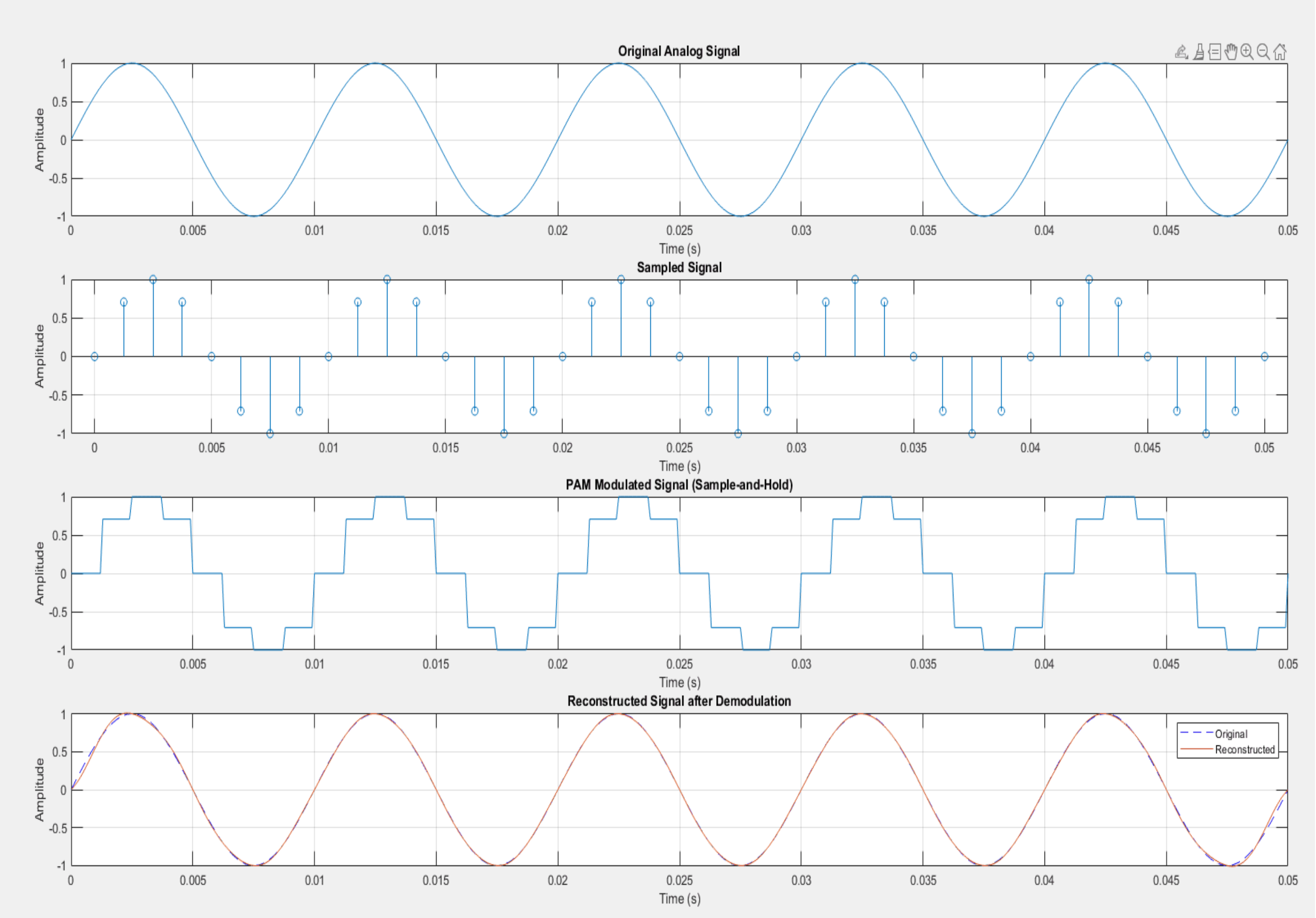
1. Generate an analog sine wave signal in MATLAB.
2. Generate sampling pulses to sample the analog signal (simulate PAM).
3. Construct a PAM signal by multiplying the analog signal with pulse train.
4. Use a low-pass Butterworth filter to reconstruct the original analog signal.
5. Plot the original, PAM, and reconstructed signals for comparison.

Code:

```
clc;
clear;
close all;

%% 1. Signal Parameters
fm = 100; % Message frequency (Hz)
fs = 10000; % Simulation sampling rate (for continuous-time modeling)
t = 0:1/fs:0.05; % Time vector
x = sin(2*pi*fm*t); % Original analog message signal

%% 2. Sampling Parameters
fs_sample = 800; % Sampling frequency (must be > 2*fm for Nyquist)
Ts = 1/fs_sample; % Sampling period
n = 0:Ts:0.05; % Sampling instants
x_sampled = sin(2*pi*fm*n); % Sampled signal values

%% 3. PAM Modulation (Sample & Hold)
pam_signal = zeros(size(t));
for i = 1:length(n)-1
 idx = t >= n(i) & t < n(i+1);
 pam_signal(idx) = x_sampled(i);
end
pam_signal(t >= n(end)) = x_sampled(end);

%% 4. Demodulation (Low-Pass Filtering for Reconstruction)
% Ideal sinc interpolation
x_reconstructed = zeros(size(t));
for i = 1:length(n)
 x_reconstructed = x_reconstructed + x_sampled(i) * sinc(fs_sample * (t - n(i)));
end

%% 5. Plotting
figure;
subplot(4,1,1);
plot(t, x);
title('Original Analog Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(4,1,2);
stem(n, x_sampled); hold on;
title('Sampled Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(4,1,3);
plot(t, pam_signal);
```

```
title('PAM Modulated Signal (Sample-and-Hold)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(4,1,4);
plot(t, x,'b--'); hold on;
plot(t, x_reconstructed);
title('Reconstructed Signal after Demodulation');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Original','Reconstructed');
grid on;
```

MODEL GRAPH:



RESULT:

- The demodulated signal should resemble the original message signal (m(t)), with some distortion due to filtering effects.

Viva Questions:

1. What is the Nyquist sampling theorem, and why is it important?
2. How does Pulse Amplitude Modulation (PAM) work in signal sampling?
3. What are the key differences between natural sampling and flat-top sampling?
4. How do you reconstruct an analog signal from its sampled version?
5. What is aliasing, and how can it be prevented?

Experiment 2
Application of Sampling by Designing Time Division Multiplexer and Demultiplexer Circuits

AIM: To study the process of sampling by Designing Time Division Modulation (TDM) and demodulation circuits using Matlab.

APPARATUS REQUIRED:

1. TDM – multiplexing Kit
2. TDM – demultiplexing Kit
3. Matlab

Algorithm for Time-Division Multiplexing (TDM) Simulation:

1. Initialization & Signal Generation

1. Sampling Parameters:
   - Sampling rate (fs = 10,000 Hz).
   - Time base (t = 0:1/fs:0.01) → 10 ms duration.
   - TDM switching frequency (fs_sample = 1000 Hz → switching every 1 ms).
2. Input Signals:
   - x1 = sin(2*pi*500*t) → 500 Hz sine wave.
   - x2 = sawtooth(2*pi*250*t) → 250 Hz sawtooth wave.

2. TDM Multiplexing

1. Initialize TDM Signal & Slot Flag:
   - tdm_signal = zeros(size(t)) → Stores the multiplexed signal.
   - slot_flag = zeros(size(t)) → Tracks which signal is being transmitted (0 for x1, 1 for x2).
2. Time-Slot Assignment:
   - Loop through each time sample:
     - Calculate the current time slot:
     - slot = mod(floor(t(i)/ts_sample), 2);
       - ts_sample = 1/fs_sample = 1 ms → Switching interval.
       - mod(..., 2) alternates between 0 and 1 every 1 ms.
     - Assign x1 or x2 to tdm_signal based on the slot:
     - if slot == 0
       tdm_signal(i) = x1(i); *% Transmit x1 in even slots*
       else
       tdm_signal(i) = x2(i); *% Transmit x2 in odd slots*
       end

3. TDM Demultiplexing

1. Separate Signals Using Slot Flag:
   - Initialize x1_demux and x2_demux to extract the original signals.
   - Loop through tdm_signal:
     - If slot_flag(i) == 0, assign to x1_demux.
     - Else, assign to x2_demux.

4. Signal Reconstruction (Low-Pass Filtering)

1. Design Butterworth LPF:
   - [b, a] = butter(6, 600/(fs/2)) → 6th-order filter with cutoff at 600 Hz.
   - The cutoff (600 Hz) is chosen to pass x1 (500 Hz) and x2 (250 Hz) while rejecting high-frequency artifacts.
2. Apply Zero-Phase Filtering (filtfilt):
   - x1_rec = filtfilt(b, a, x1_demux) → Reconstruct x1.
   - x2_rec = filtfilt(b, a, x2_demux) → Reconstruct x2.

5. Plotting the Results

The code generates a 4×2 subplot showing:

1. Original Signals (x1, x2).
2. TDM Multiplexed Signal (tdm_signal).
3. Slot Flag (slot_flag).
4. Demultiplexed Signals (x1_demux, x2_demux).
5. Reconstructed Signals (x1_rec, x2_rec).

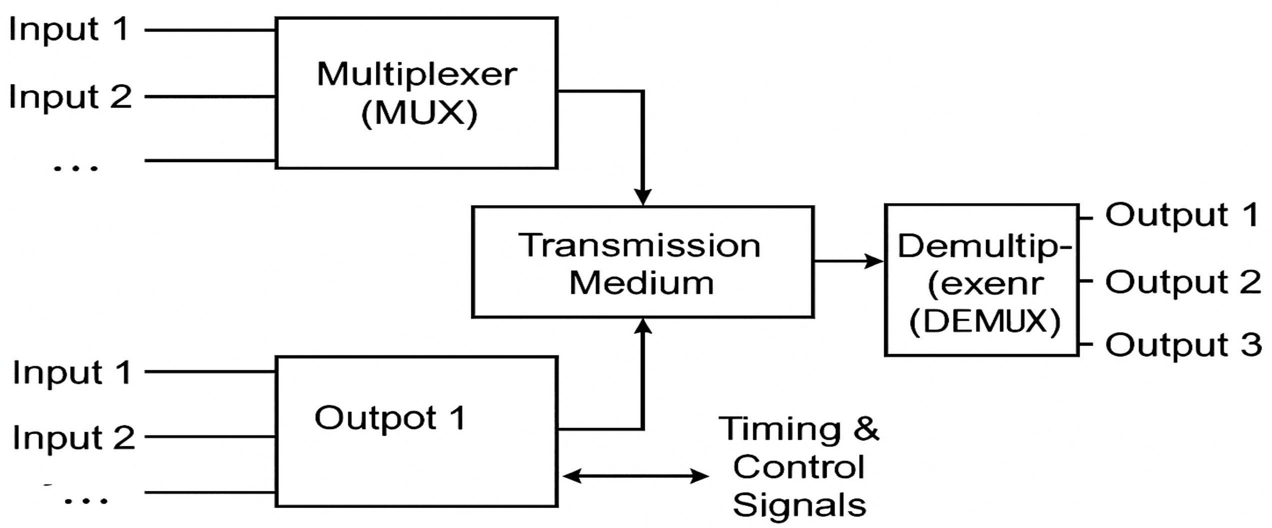


Fig: TDM Block Diagram

Merits:

a. It can easily accommodate both analog and digital sources.

b. TDM has immune to non-linearity’s in the channel.

Demerits:

It is highly sensitive to amplitude, phase variations in the channel.

Code:

```
clc; clear;
fs = 10000; % Sampling rate
t = 0:1/fs:0.01; % Time base (10 ms)
fs_sample = 1000; % TDM switching frequency
ts_sample = 1/fs_sample;
x1 = sin(2*pi*500*t); % Signal 1: sine
x2 = sawtooth(2*pi*250*t); % Signal 2: sawtooth
tdm_signal = zeros(size(t));
slot_flag = zeros(size(t));
for i = 1:length(t)
 slot = mod(floor(t(i)/ts_sample), 2);
 slot_flag(i) = slot;
 if slot == 0
 tdm_signal(i) = x1(i);
 else
 tdm_signal(i) = x2(i);
 end
end
x1_demux = zeros(size(t));
x2_demux = zeros(size(t));
for i = 1:length(t)
 if slot_flag(i) == 0
 x1_demux(i) = tdm_signal(i);
 else
 x2_demux(i) = tdm_signal(i);
 end
end
[b, a] = butter(6, 600/(fs/2)); % 600 Hz cutoff
x1_rec = filtfilt(b, a, x1_demux);
x2_rec = filtfilt(b, a, x2_demux);
figure;
subplot(4,2,1);
plot(t, x1); title('Original Signal 1'); xlabel('Time'); ylabel('Amplitude'); grid on;
```

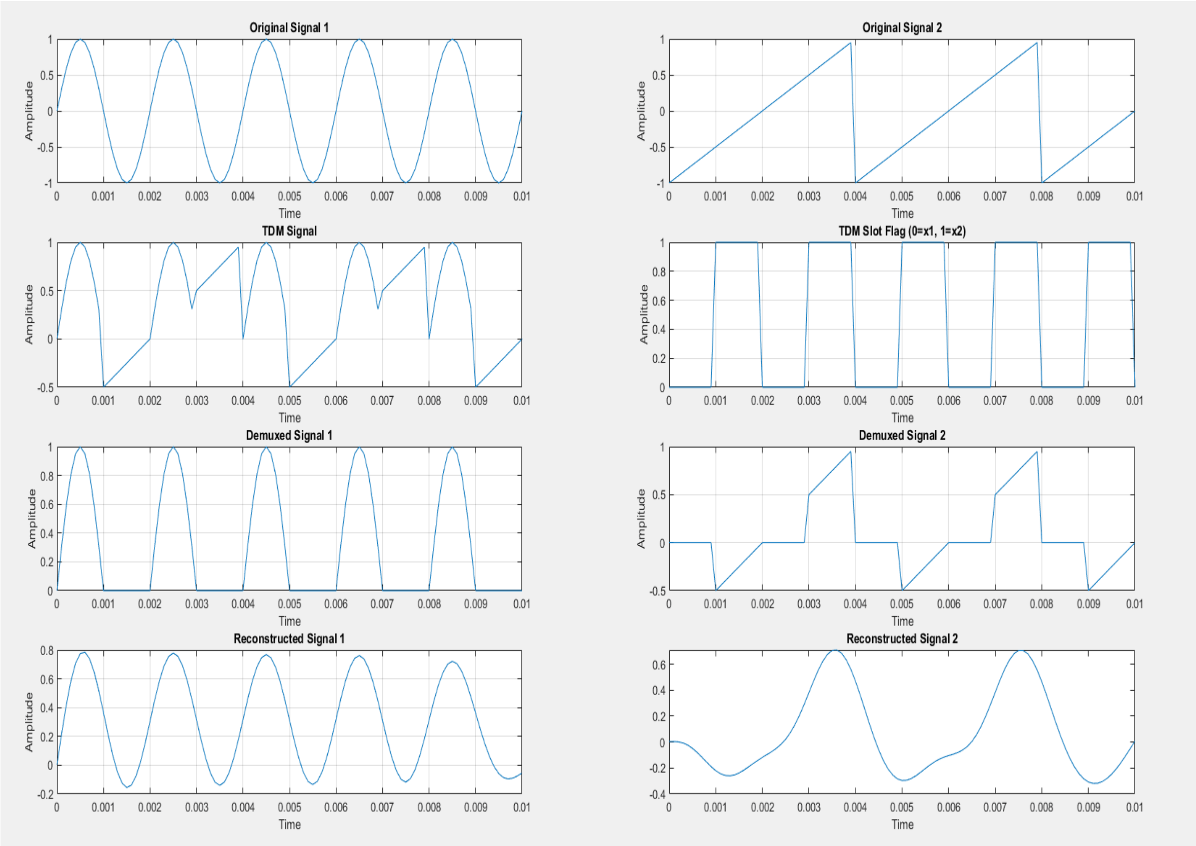
```
subplot(4,2,2);
plot(t, x2); title('Original Signal 2'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,3);
plot(t, tdm_signal); title('TDM Signal'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,4);
plot(t, slot_flag); title('TDM Slot Flag (0=x1, 1=x2)'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,5);
plot(t, x1_demux); title('Demuxed Signal 1'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,6);
plot(t, x2_demux); title('Demuxed Signal 2'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,7);
plot(t, x1_rec); title('Reconstructed Signal 1'); xlabel('Time'); ylabel('Amplitude'); grid on;
subplot(4,2,8);
plot(t, x2_rec); title('Reconstructed Signal 2'); xlabel('Time'); ylabel('Amplitude'); grid on;
```

Procedure:

This experiment includes:

1. Generating two analog signals (sine waves)
2. Sampling them alternately (time division multiplexing)
3. Demultiplexing the TDM signal using synchronized clocks
4. Reconstructing original signals using low-pass filters

Model Graph:



<u>Result:</u>

Thus for given message signal Time Division multiplexing and demultiplexing is obtained. It is plotted in graph.

<u>Viva Questions:</u>

1. Explain the working principle of TDM with an example.
2. What are the advantages of TDM over Frequency Division Multiplexing (FDM)?
3. How does synchronization play a role in TDM systems?
4. What are guard bands, and why are they used in TDM?
5. How would you design a basic TDM circuit using analog switches?

Experiment 3

Transmission of Digital Information Using Analog Amplitude Modulation

AIM: To implement an amplitude modulation and demodulation system that transmits digital information using a sinusoidal carrier and reconstructs the original binary data at the receiver.

Apparatus Required

- MATLAB (or GNU Radio)
- PC with MATLAB installed

Algortihm for Amplitude Modulation:

1. Initialization & Signal Generation

1. Define Parameters:
   - Sampling frequency (Fs = 10,000 Hz).
   - Carrier frequency (fc = 500 Hz).
   - Message frequency (fm = 50 Hz).
   - Carrier amplitude (Ac = 1.0).
   - Message amplitude (Am = 0.5).
   - Time duration (duration = 0.1 sec).
   - Generate Time Vector (t):
     - t = 0:1/Fs:duration; % Time base (0 to 0.1 sec with 10,000 samples)
   - Generate Message Signal (m(t)):
     - m = Am * cos(2*pi*fm*t); % 50 Hz cosine wave
   - Generate Carrier Signal (c(t)):
     - carrier = Ac * cos(2*pi*fc*t); % 500 Hz cosine wave

2. AM Modulation

1. Modulate the Signal (s(t)):
   - Standard AM equation:
   - $s(t)=(Ac+m(t))\cdot\cos(2\pi fct)$
   - MATLAB implementation:
   - s = (Ac + m) .* cos(2*pi*fc*t);
2. Plot the AM Signal:
   - The envelope (upper and lower bounds) is given by:
   - Envelope=$Ac+m(t)$
   - MATLAB visualization:
   - plot(t, s); hold on;
     plot(t, Ac + m, 'k', t, -Ac - m, 'k');

3. AM Demodulation (Envelope Detection)

1. Extract the Envelope Using Hilbert Transform:
   - The Hilbert transform computes the analytic signal:
   - analytic_signal=$s(t)+j\cdot s\hat{}(t)$
   - where $s\hat{}(t)$ is the Hilbert transform of $s(t)$.
   - The envelope is the magnitude of the analytic signal:
   - envelope= ∣ analytic_signal ∣
   - MATLAB implementation:
   - analytic_signal = hilbert(s);
     envelope = abs(analytic_signal);
2. Remove DC Offset (Carrier Amplitude):
   - The envelope contains a DC component (Ac), which is subtracted:
   - reconstructed=envelope−Ac
   - reconstructed = envelope - Ac;
3. Optional: Low-Pass Filtering (Smoothing)
   - A 6th-order Butterworth LPF removes high-frequency noise:
   -
   - [b, a] = butter(6, (fm * 2) / (Fs / 2)); *% Cutoff at 100 Hz (2×fm)*
     reconstructed = filtfilt(b, a, reconstructed);

4. Plotting the Results

The code generates a 5×1 subplot showing:

1. Message Signal (m(t)) → Original 50 Hz cosine wave.
2. Carrier Signal (c(t)) → 500 Hz cosine wave.
3. AM Modulated Signal (s(t)) with its envelope.
4. Demodulated Signal (Envelope) before DC removal.
5. Reconstructed Message Signal after DC removal and filtering.

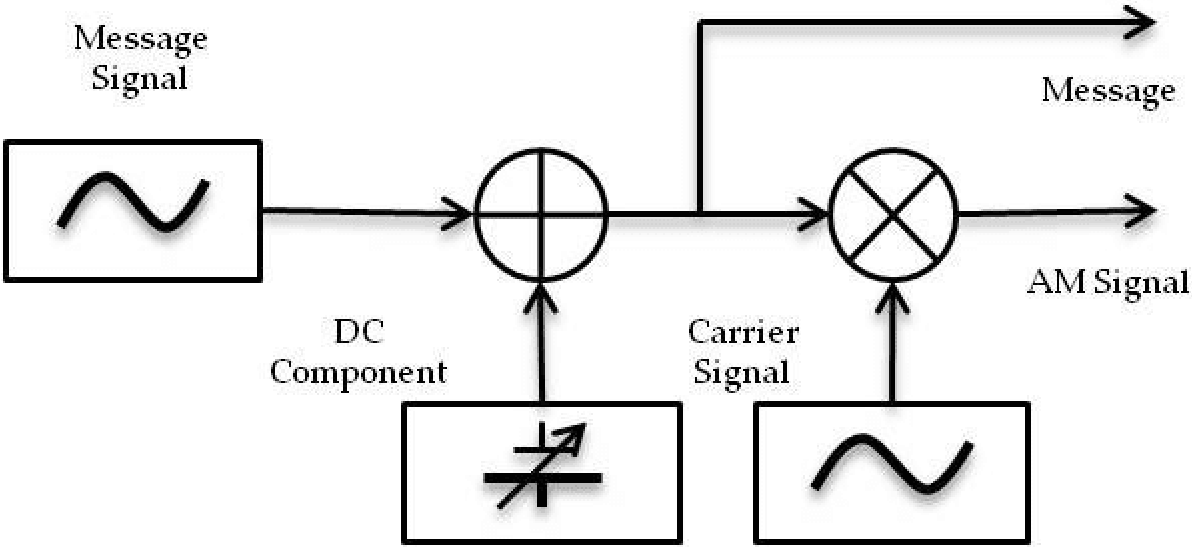


Fig: Amplitude Modulation

Code:

```
clc; clear; close all;
Fs = 10000; % Sampling frequency (Hz)
fc = 500; % Carrier frequency (Hz)
fm = 50; % Message frequency (Hz)
Ac = 1.0; % Carrier amplitude
Am = 0.5; % Message amplitude
duration = 0.1; % Time duration (seconds)
t = 0:1/Fs:duration;
m = Am * cos(2*pi*fm*t); % Message signal
carrier = Ac * cos(2*pi*fc*t); % Carrier signal
s = (Ac + m) .* cos(2*pi*fc*t); % AM modulated signal
analytic_signal = hilbert(s);
envelope = abs(analytic_signal); % Envelope = demodulated signal
% Remove DC (carrier amplitude)
reconstructed = envelope - Ac;
% Optional: Low-pass filter to smooth the envelope
[b, a] = butter(6, (fm * 2) / (Fs / 2)); % 6th-order Butterworth
reconstructed = filtfilt(b, a, reconstructed);
figure('Name','AM Modulation & Demodulation','NumberTitle','off');
% 1. Message Signal
subplot(5,1,1);
plot(t, m);
title('Message Signal (Cosine)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

```
% 2. Carrier Signal
subplot(5,1,2);
plot(t, carrier);
title('Carrier Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% 3. AM Modulated Signal
subplot(5,1,3);
plot(t, s); hold on;
plot(t, Ac + m,'k', t, -Ac - m','k'); % Envelope
title('AM Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
legend('AM Signal', 'Envelope');
grid on;
% 4. Demodulated Signal (Envelope)
subplot(5,1,4);
plot(t, envelope);
title('Demodulated Signal (Envelope)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% 5. Reconstructed Message Signal
subplot(5,1,5);
plot(t, reconstructed);
title('Reconstructed Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

Procedure:

Modulation:

- Generate a square wave (digital binary data) using the function generator.
- Generate a high-frequency sine wave (carrier).
- Use an analog multiplier or transistor switch to modulate the carrier amplitude based on the digital data.
- Observe the AM waveform on the oscilloscope.

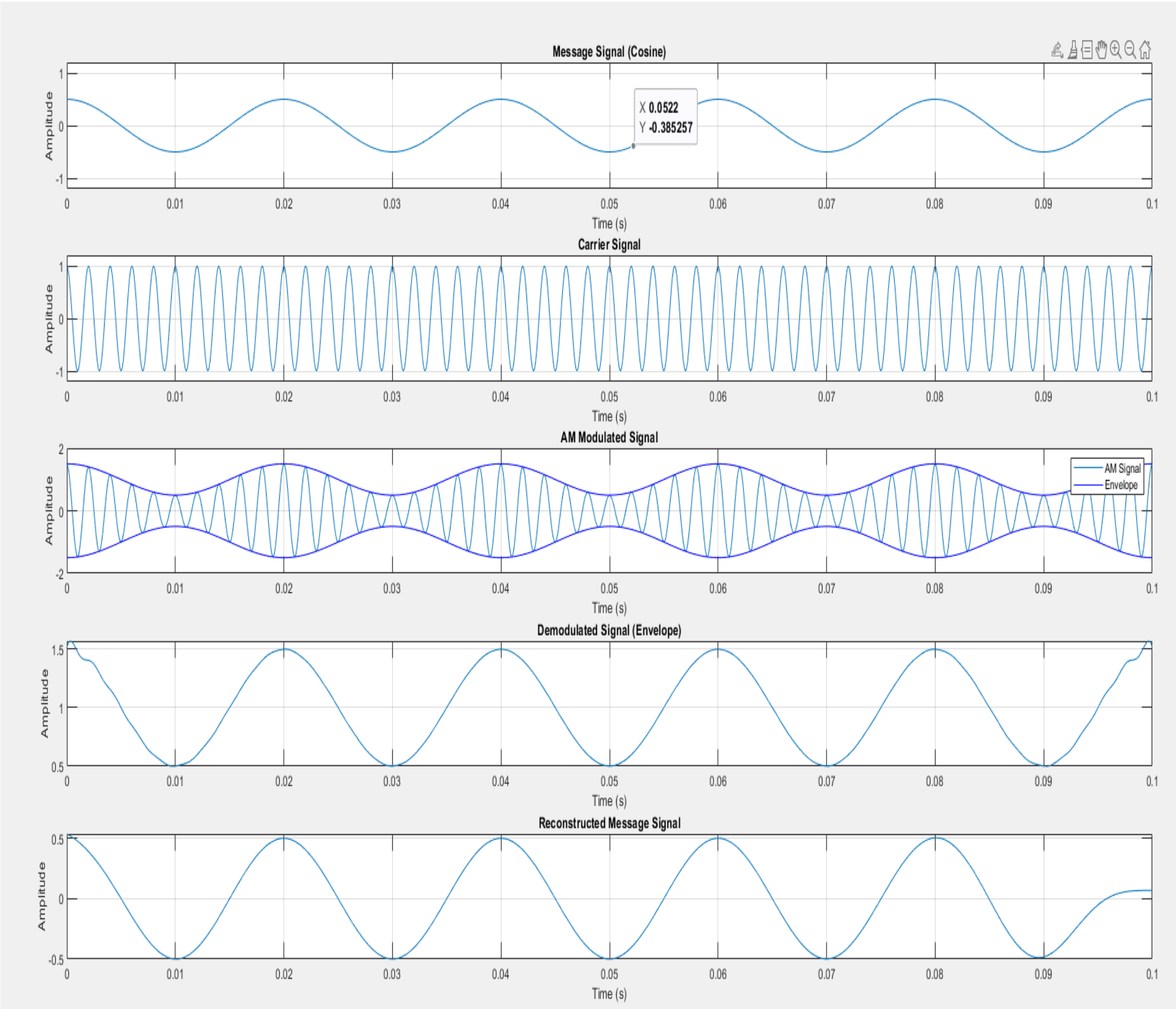
Transmission:

- Transmit the modulated signal over a wired connection or channel.

Demodulation:

- Multiply the received signal by a locally generated carrier (coherent detection).
- Pass the result through a low-pass filter to extract the baseband digital signal.
- Use a comparator for thresholding circuit to clean up the reconstructed digital data.

<u>Model Graph:</u>



<u>Result:</u>Thus for given message signal Amplitude modulation and demodulation is obtained. It is plotted in graph.

Viva Question:

1. How can AM be used to transmit digital data?
2. What is Amplitude Shift Keying (ASK), and how does it differ from analog AM?
3. What are the challenges in demodulating an AM-modulated digital signal?
4. How does envelope detection work in AM demodulation?
5. What is the impact of noise on AM-based digital communication?

Experiment 4

Phase Modulation for Digital Data Transmission

Aim: To implement a phase modulation and demodulation system in MATLAB for transmitting binary digital data using a carrier signal and reconstruct the original message at the receiver.

Apparatus:

- MATLAB software
- Computer with standard processing power
- Pulse Modulation & Demodulation Kit

Algorithm for Phase Modulation:

1. Initialization & Signal Generation

1. Define Parameters:
    - Sampling frequency (Fs = 5000 Hz).
    - Message frequency (fm = 5 Hz).
    - Carrier frequency (fc = 20 Hz).
    - Time vector (t = 0:1/Fs:1).
2. Generate Message Signal (message):
    - A square wave (for distinct phase transitions):
    - message = square(2*pi*fm*t);
        - Alternates between +1 and -1 at 5 Hz.
3. Generate Carrier Signal (carrier):
    - A sine wave at 20 Hz:
    - carrier = sin(2*pi*fc*t);

2. Phase Modulation (PM)

1. Modulate the Signal (pm_signal):
    - The PM signal is generated by varying the carrier's phase with the message:
    - $s(t)=\cos(2\pi fct+\Delta\phi\cdot m(t))$
    - where $\Delta\phi=\pi/2$ (phase deviation).
    - MATLAB implementation:
    - pm_signal = cos(2*pi*fc*t + (pi/2)*message);

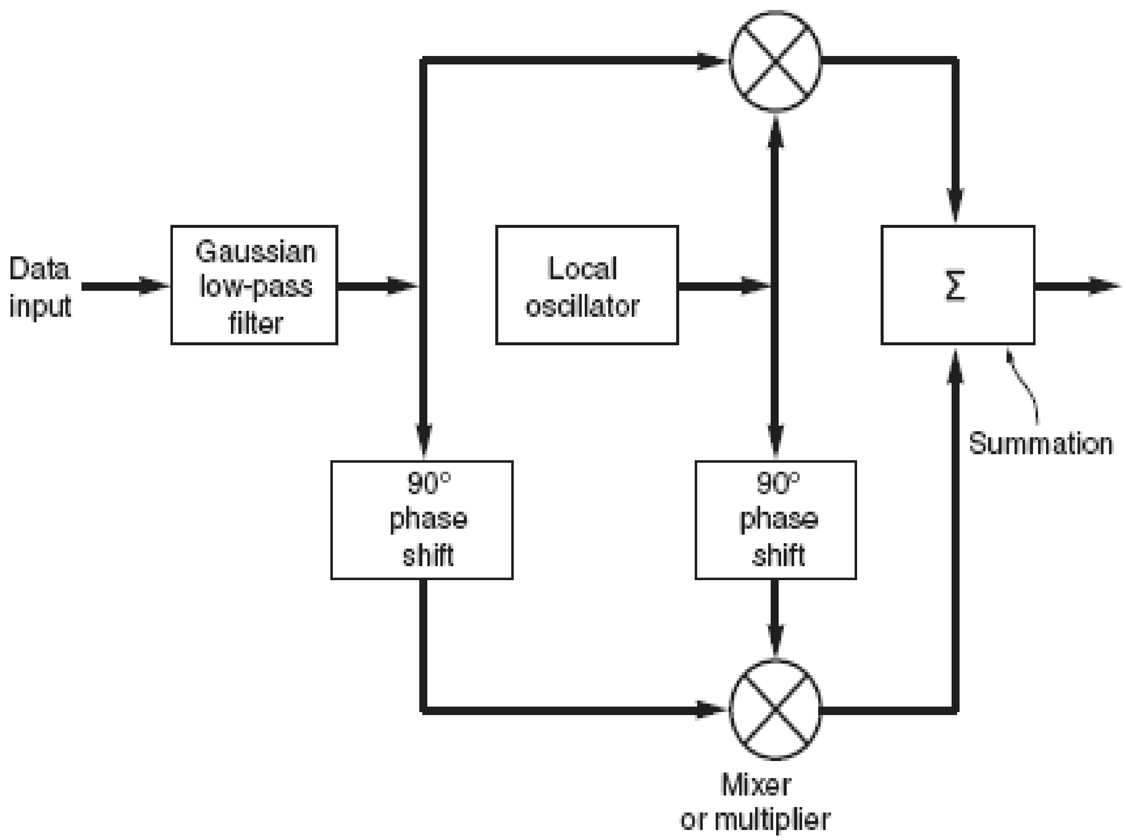
3. PM Demodulation (Hilbert Transform Method)

1. Compute Analytic Signal:
   - The Hilbert transform extracts the instantaneous phase:
   - analytic_signal=$s(t)+j\cdot s$^(t)
   - where s^(t) is the Hilbert transform of $s(t)$.
   - MATLAB:
   - analytic_signal = hilbert(pm_signal);
2. Extract Instantaneous Phase:
   - The phase is obtained using angle() and unwrap():
   - $\phi(t)$=unwrap(∠analytic_signal)
   - instantaneous_phase = unwrap(angle(analytic_signal));
3. Demodulate the Signal:
   - Subtract the carrier's linear phase term and scale by $\pi/2$:
   - demodulated$(t)=\pi/2\phi(t)-2\pi fct$
   - demodulated = (instantaneous_phase - 2*pi*fc*t)/(pi/2);
4. Low-Pass Filtering (Butterworth LPF):
   - A 5th-order Butterworth filter smoothens the demodulated signal:
   - [b, a] = butter(5, fm*2/Fs); % Cutoff at 10 Hz (2×fm)
     reconstructed = filtfilt(b, a, demodulated);

4. Plotting the Results

The code generates a 5×1 subplot showing:

1. (a) Original Message Signal → Square wave at 5 Hz.
2. (b) Carrier Signal → Sine wave at 20 Hz.
3. (c) Phase Modulated Signal → PM waveform.
4. (d) Demodulated Signal (Before Filtering) → Noisy, but follows the message.
5. (e) Reconstructed Signal (After Filtering) → Smoothed and matches the original message.



Code:

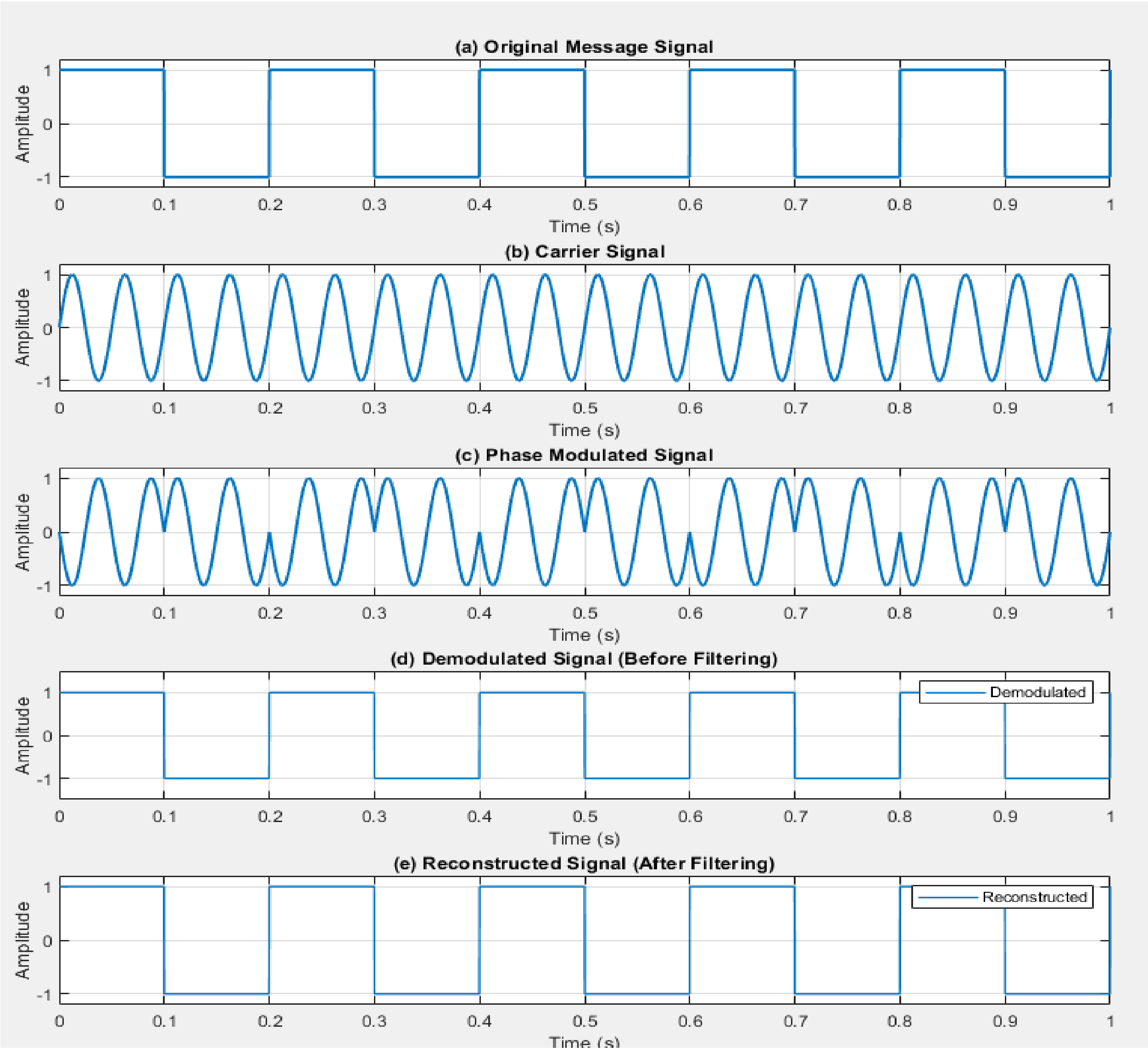
```
% Phase Modulation with Separate Demodulation & Reconstruction Plots
clear all;
close all;
clc;
% Time parameters
Fs = 5000; % Sampling frequency (Hz)
t = 0:1/Fs:1; % Time vector (1 second)
% Message signal (square wave for clear phase transitions)
fm = 5; % Message frequency (Hz)
message = square(2*pi*fm*t);
% Carrier signal (high frequency sine wave)
fc = 20; % Carrier frequency (Hz)
carrier = sin(2*pi*fc*t);
% Phase modulation with clear phase jumps
pm_signal = cos(2*pi*fc*t + (pi/2)*message);
analytic_signal = hilbert(pm_signal);
instantaneous_phase = unwrap(angle(analytic_signal));
demodulated = (instantaneous_phase - 2*pi*fc*t)/(pi/2);
[b,a] = butter(5, fm*2/Fs);
reconstructed = filtfilt(b, a, demodulated);
figure;
% Original Message Signal
subplot(5,1,1);
```

```
plot(t, message);
title('(a) Original Message Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% Carrier Signal
subplot(5,1,2);
plot(t, carrier);
title('(b) Carrier Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% Phase Modulated Signal
subplot(5,1,3);
plot(t, pm_signal);
title('(c) Phase Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
% Demodulated Signal (before filtering)
subplot(5,1,4);
plot(t, demodulated);
plot(t, message); % Original for comparison
title('(d) Demodulated Signal (Before Filtering)');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Demodulated', 'Original');
grid on;
% Reconstructed Signal (after filtering)
subplot(5,1,5);
plot(t, reconstructed);
plot(t, message); % Original for comparison
title('(e) Reconstructed Signal (After Filtering)');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Reconstructed', 'Original');
grid on;
```

Procedure:

- **Initialize Parameters:**
  - Set time, sampling frequency, and create a time vector.
- **Generate Message Signal:**
  - Use a square wave to simulate digital data (±1).
- **Generate Carrier Signal:**
  - Use a cosine wave with high frequency.
- **Perform Phase Modulation:**
  - Modify the phase of the carrier using the digital message.
- **Demodulation:**
  - Extract the phase using the Hilbert transform.
  - Take the derivative to estimate the original message.
- **Reconstruction:**
  - Apply a low-pass filter to recover the message signal.
- **Plot the Results:**
  - Plot message, carrier, modulated, and reconstructed signals.

Model Graph:



Result: Thus for given message signal Phase modulation and demodulation is obtained. It is plotted in graph.

Viva Questions:

1. How does Phase Shift Keying (PSK) differ from analog phase modulation?
2. Explain the difference between BPSK and QPSK.
3. What is a constellation diagram, and how is it useful in digital modulation?
4. How does coherent detection work in PSK demodulation?
5. What is phase ambiguity, and how can it be resolved in PSK systems?

EXPERIMENT 5(a)
Pulse Code Modulation

Aim: To implement Pulse Code Modulation (PCM) in MATLAB.

Apparatus

- MATLAB Software (any version)
- Computer System
- Signal Generator (MATLAB simulated)

ALgorithm for Pulse Code:

1. Initialization & Input Parameters

1. User Inputs:
   - n: Number of bits for PCM (e.g., 3 to 8).
   - samples: Number of samples per period (e.g., 20 to 100).
2. Generate Analog Signal (s(t)):
   - A sine wave with amplitude Am = 8:
   - s(t)=8·sin(x),x=0:samples2π:4π

2. Sampling

1. Plot Original Analog Signal:
   - plot(x, s);
     title('Analog Signal');
2. Plot Sampled Signal (Discrete-Time):
   - stem(x, s);
     title('Sampled Signal');

3. Quantization

1. Determine Quantization Levels:
   - Number of levels:
   - L=2n
   - Step size (quantization interval):
   - Δ=LVmax−Vmin,Vmax=8,Vmin=−8
   - Partition boundaries:
   - part=[−8+Δ,−8+2Δ,…,8−Δ]
   - Quantization codes (midpoints):
   - code=[−8+2Δ,−8+23Δ,…,8−2Δ]

2. Quantize the Sampled Signal:
    - MATLAB’s quantiz function maps samples to quantization levels:
    - [ind, q] = quantiz(s, part, code);
    - ind = Quantization indices.
    - q = Quantized signal values.
3. Plot Quantized Signal:
    - stem(x, q);
      title('Quantized Signal');

4. Encoding (Binary Representation)

1. Convert Quantization Indices to n-bit Binary:
    - Each index is converted to an n-bit binary word (MSB first):
    - code1 = de2bi(ind, n, 'left-msb');
2. Flatten Binary Matrix into a Bitstream:
    - coded = reshape(code1', 1, []);
3. Plot Encoded PCM Signal:
    - stairs(coded);
      title('Encoded Signal (Binary PCM)');

5. Decoding (Reconstruction)

1. Reshape Bitstream into n-bit Words:
    - qunt = reshape(coded, n, [])';
2. Convert Binary Back to Decimal Indices:
    - index = bi2de(qunt, 'left-msb');
3. Reconstruct Quantized Signal:
    - Map indices back to quantized amplitudes:
    - $q$reconstructed=Vmin+2Δ+Δ⋅index
    - q_reconstructed = vmin + (del/2) + del*index';
4. Plot Reconstructed Signal:
    - plot(x(1:length(q_reconstructed)), q_reconstructed);
      title('Demodulated Signal');

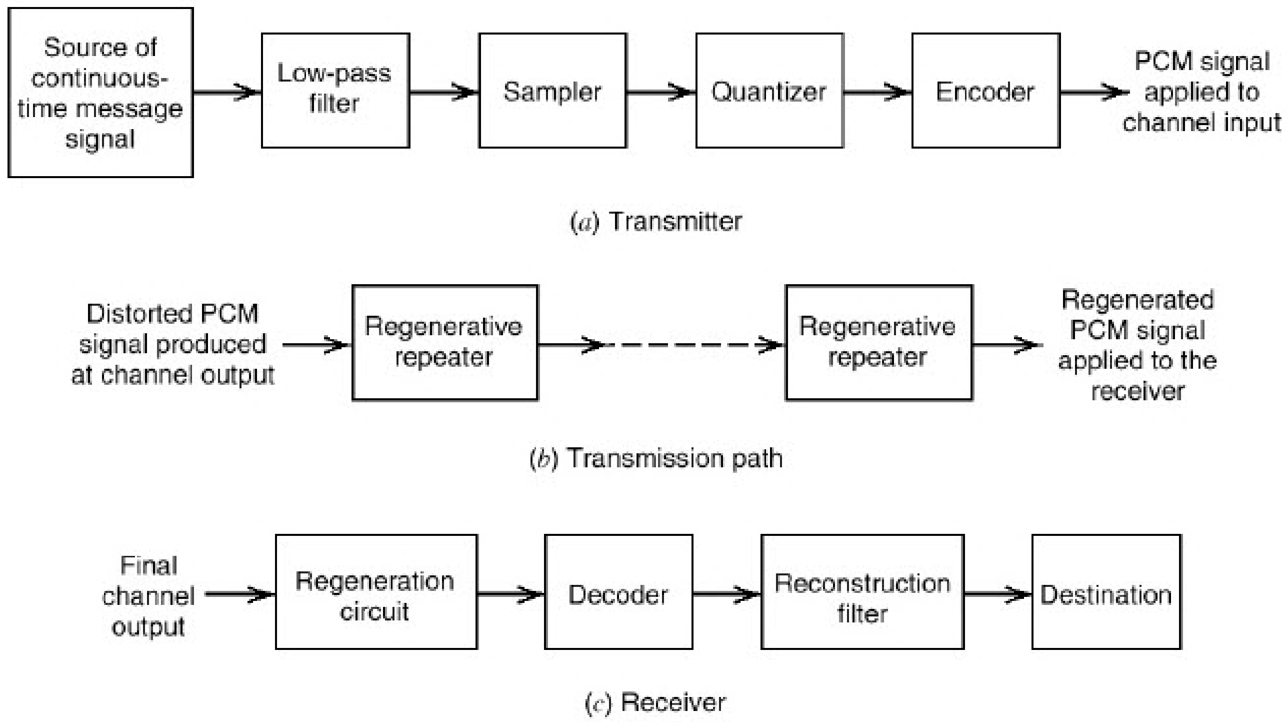


Fig: PCM

Code:

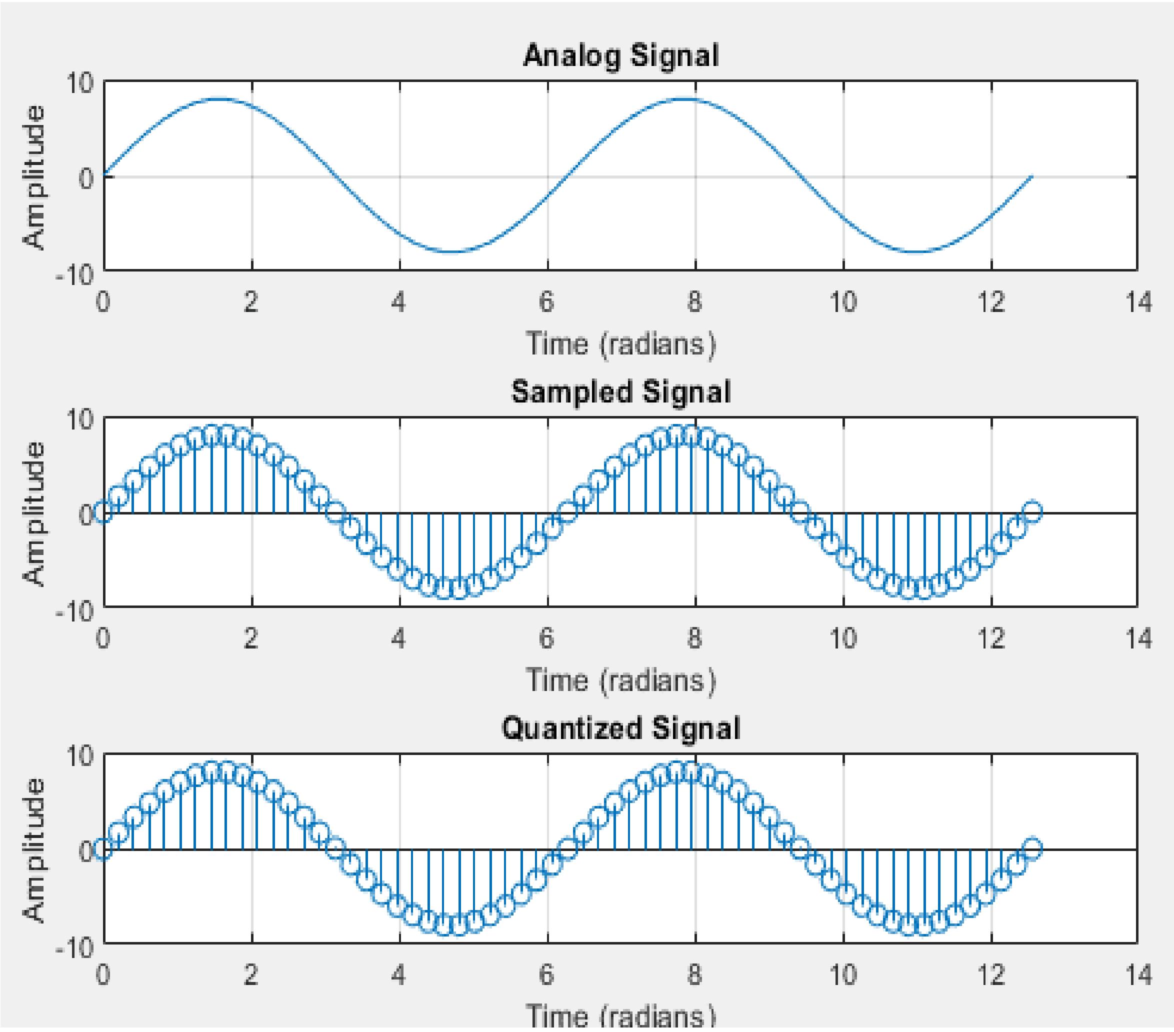
```
clc;
close all;
clear all;
% Input parameters
n = input('Enter n value for n-bit PCM system (e.g., 3-8): ');
samples = input('Enter number of samples in a period (e.g., 20-100): ');
% Signal generation
Am = 8;
x = 0:(2*pi/samples):4*pi;
s = Am*sin(x);
% Plot original and sampled signals
figure(1);
subplot(3,1,1);
plot(x, s);
grid on;
title('Analog Signal');
ylabel('Amplitude');
xlabel('Time (radians)');
subplot(3,1,2);
stem(x, s);
grid on;
title('Sampled Signal');
```

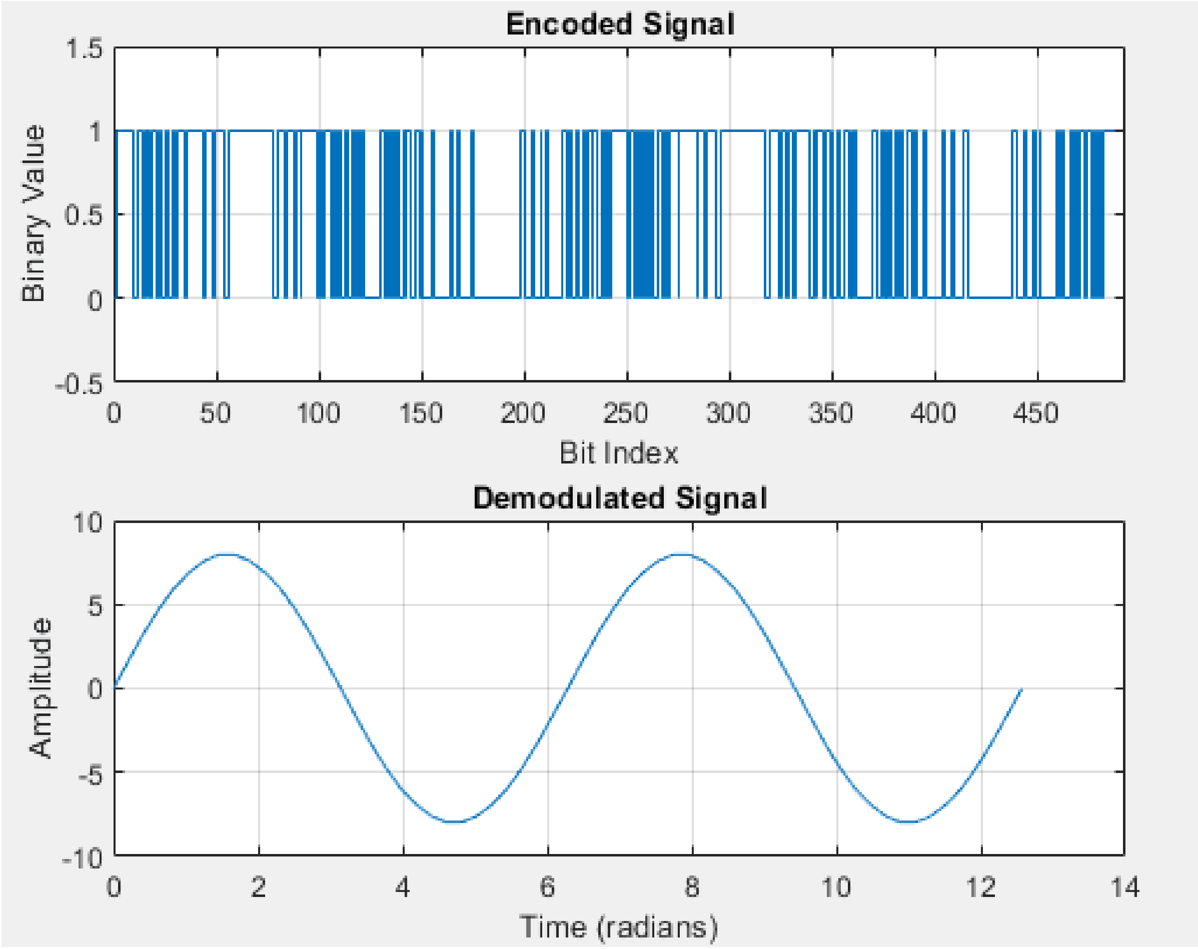
```
ylabel('Amplitude');
xlabel('Time (radians)');
% Quantization
L = 2^n;
vmax = Am;
vmin = -vmax;
del = (vmax-vmin)/L;
part = vmin+del:del:vmax-del;
code = vmin+(del/2):del:vmax-(del/2);
[ind, q] = quantiz(s, part, code);
subplot(3,1,3);
stem(x, q);
grid on;
title('Quantized Signal');
ylabel('Amplitude');
xlabel('Time (radians)');
% Encoding
code1 = de2bi(ind, n, 'left-msb');
coded = reshape(code1', 1, []); % Flatten the binary matrix
% Plot encoded signal
figure(2);
subplot(2,1,1);
stairs(coded);
grid on;
axis([0 length(coded)+5 -0.5 1.5]);
title('Encoded Signal');
ylabel('Binary Value');
xlabel('Bit Index');
% Decoding
qunt = reshape(coded, n, [])'; % Reshape back to n-bit words
index = bi2de(qunt, 'left-msb');
q_reconstructed = vmin + (del/2) + del*index';
% Plot reconstructed signal
subplot(2,1,2);
plot(x(1:length(q_reconstructed)), q_reconstructed);
grid on;
title('Demodulated Signal');
ylabel('Amplitude');
xlabel('Time (radians)');
```

Procedure:

1. Generate an analog sinusoidal signal.
2. Sample the signal at a frequency greater than twice the signal frequency.
3. Quantize the sampled signal using uniform quantization.
4. Convert quantized values to binary.
5. Plot the original, quantized and sampling signals.

Model Graph:





Result: The original signal was sampled and quantized.

Viva Questions:

1. What are the key steps in PCM encoding?
2. How does quantization affect PCM signal quality?

Experiment 5(b)

Delta Modulation

AIM: To implement Delta Modulation in MATLAB.

APPARATUS:

- MATLAB Software
- Computer System
- Signal Generator (in code)

Algorithm for Delta Modulation:

1. Initialization & Signal Generation

1. Define Parameters:
   - Signal amplitude (a = 2).
   - Time vector (t = 0:2*pi/50:2*pi).
   - Original signal (x = a*sin(t)).
   - Step size (delta = 0.25).
2. Initialize Arrays:
   - xq: Stores the quantized (staircase) signal.
   - d: Stores the binary DM output (1 for increase, 0 for decrease).

2. Delta Modulation (Encoding)

1. Algorithm:
   - Compare the input signal x(i) with the previous quantized value xq(i).
   - If x(i) > xq(i), set d(i) = 1 and increase xq(i+1) by delta.
   - Else, set d(i) = 0 and decrease xq(i+1) by delta.

```
for i = 1:length(t)-1
    if x(i) > xq(i)
        d(i) = 1;
        xq(i+1) = xq(i) + delta;
    else
        d(i) = 0;
        xq(i+1) = xq(i) - delta;
    end
end
```

2. Output:
   - d: Binary DM signal (0s and 1s).
   - xq: Staircase approximation of x(t).

■

3. Delta Demodulation (Decoding)

1. Algorithm:
   - Start with demod(1) = 0.
   - For each bit in d(i):
     - If d(i) = 1, increase demod(i+1) by delta.
     - Else, decrease demod(i+1) by delta.

```
for i = 1:length(d)
   if d(i) == 1
      demod(i+1) = demod(i) + delta;
   else
      demod(i+1) = demod(i) - delta;
   end
end
```

2. Output:
   - demod: Reconstructed signal (similar to xq but shifted by 1 sample).

4. Plotting the Results

1. Subplot 1: Original, Modulated, and Demodulated Signals
   - Original signal (x): Smooth sine wave.
   - Staircase approximation (xq): Quantized DM signal.
   - Demodulated signal (demod(2:end)): Reconstructed signal (almost identical to xq).
2. Subplot 2: Binary DM Output
   - Binary DM signal (d): 0s and 1s representing slope changes.

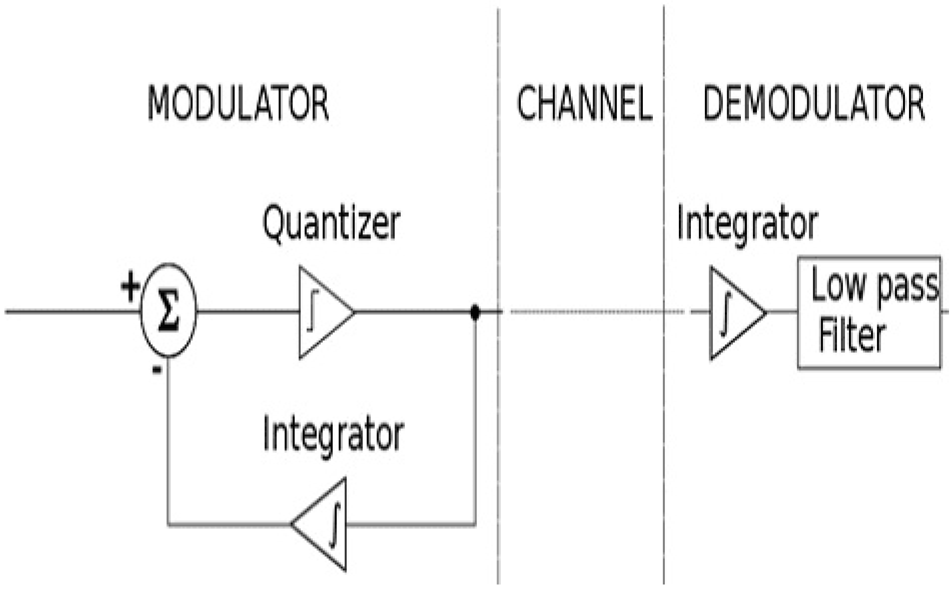


Fig: Delta Modulation

Code:

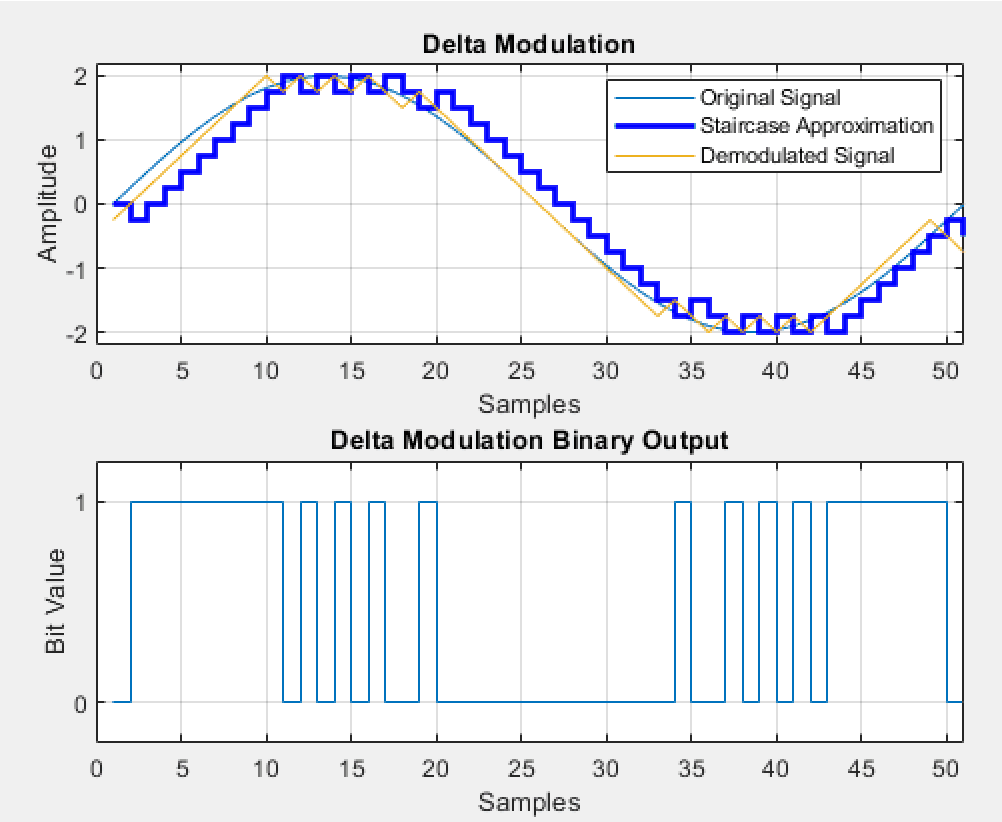
```
clc;
close all;
clear all;
% Parameters
a = 2; % Amplitude of signal
t = 0:2*pi/50:2*pi; % Time vector
x = a*sin(t); % Original signal
delta = 0.25; % Step size
% Delta Modulation Encoding
xq = zeros(1, length(t)); % Quantized signal
d = zeros(1, length(t)); % Binary output
xq(1) = 0; % Initial value
for i = 1:length(t)-1
 if x(i) > xq(i)
 d(i) = 1;
 xq(i+1) = xq(i) + delta;
 else
 d(i) = 0;
 xq(i+1) = xq(i) - delta;
 end
end
% Delta Demodulation
demod = zeros(1, length(t)+1);
demod(1) = 0;
for i = 1:length(d)
 if d(i) == 1
 demod(i+1) = demod(i) + delta;
 else
 demod(i+1) = demod(i) - delta;
 end
end
% Plotting
figure;
% Original and Modulated Signals
subplot(2,1,1);
plot(x);
hold on;
stairs(xq);
plot(demod(2:end)); % Skip initial zero
```

```
hold off;
xlabel('Samples');
ylabel('Amplitude');
title('Delta Modulation', 'FontSize', 10);
legend('Original Signal', 'Staircase Approximation', 'Demodulated Signal');
grid on;
axis([0 51 -(a+0.2) a+0.2]);
% Binary DM Signal
subplot(2,1,2);
stairs(d);
xlabel('Samples');
ylabel('Bit Value');
title('Delta Modulation Binary Output', 'FontSize', 10);
axis([0 51 -0.2 1.2]);
yticks([0 1]);
grid on;
```

Procedure:

1. Generate a sinusoidal signal.
2. Set step size (delta) for modulation.
3. For each sample:
    - Compare current input with predicted output.
    - Adjust predicted output by ±delta.
4. Plot the reconstructed signal and bit stream.

Model Graph:



Result:

The Delta Modulated signal was successfully reconstructed. The accuracy depends on the step size. Increasing delta improves tracking but reduces precision.

Viva Questions:

1. What is the main advantage of Delta Modulation over PCM?
2. What is slope overload distortion in Delta Modulation?

Experiment 6

Geometric Signal Representation using Gram-Schmidt Orthogonalization

Aim: To implement the Gram-Schmidt Orthogonalization procedure in MATLAB to geometrically represent given signals in a vector space.

Apparatus:

- Matlab
- Pc or Computer

Algorithm:

Signal Definition

1. Input Signals:
2. signal1 = [1 1 0 0]; *% Binary signal 1*

   signal2 = [0 1 1 0]; *% Binary signal 2*

   signal3 = [1 0 0 1]; *% Binary signal 3*
3. Combine into Matrix:
4. signals = [signal1; signal2; signal3];

2. Gram-Schmidt Orthogonalization

Key Steps:

1. Initialize Basis Vectors:
2. basis = zeros(num_signals, len);
3. Orthogonalization Process:
   - For each signal v = signals(i,:):
     - Subtract its projection onto all previous basis vectors:
     - v = v - dot(v,basis(j,:))/dot(basis(j,:),basis(j,:)) * basis(j,:);
     - Normalize the residual to get the new basis vector:
     - basis(i,:) = v/norm(v);
4. Coefficient Calculation:
   - Compute projection coefficients of each signal onto the basis:

- coefficients(k,m) = dot(signals(k,:), basis(m,:));

Output:

- basis: Orthonormal vectors spanning the signal space.
- coefficients: Coordinates of original signals in this basis.

3. Visualization

A. Time-Domain Plots

1. Original Signals:
2. stem(signals(i,:), 'Color', colors(i));
3. Basis Vectors:
4. stem(basis(i,:), 'Marker', 's');

B. Geometric Representation

1. 2D Case:
    - Plot basis vectors as arrows from origin.
    - Reconstruct signals using coefficients:
    - quiver(0,0, signal_vec(1), signal_vec(2));
2. **3D Case:**
    - Similar to 2D but with quiver3.

4. Orthonormality Verification

Check that the basis is orthonormal:

```
orth_check = basis * basis';  % Should be identity matrix
```

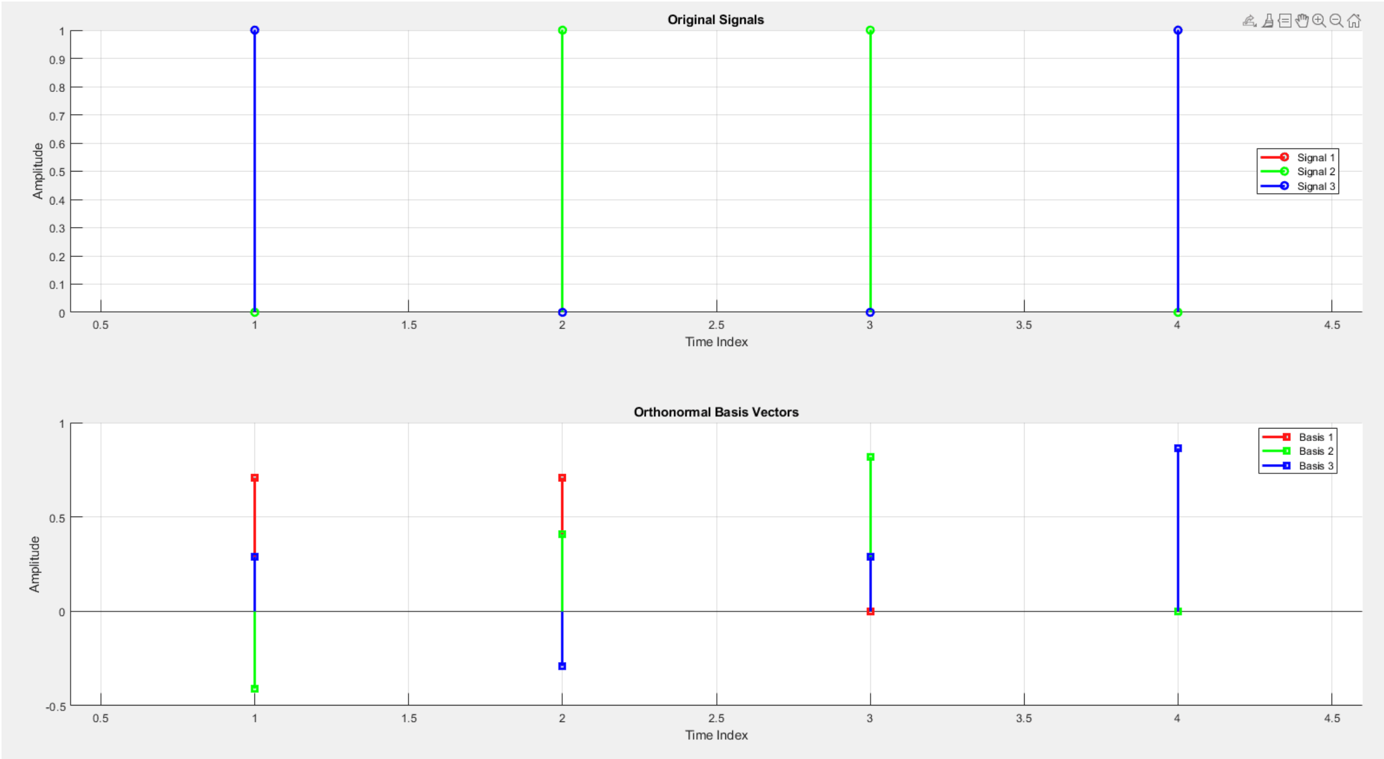
Code:

```
%% Geometric Representation of Signals using Gram-Schmidt Orthogonalization
% This script demonstrates how to represent signals geometrically using
% Gram-Schmidt orthogonalization procedure
clc;
clear;
close all;
%% Step 1: Define the signals to be analyzed
% Each row represents a different signal
signal1 = [1 1 0 0]; % Signal 1
signal2 = [0 1 1 0]; % Signal 2
signal3 = [1 0 0 1]; % Signal 3
signals = [signal1; signal2; signal3]; % Combine signals into a matrix
num_signals = size(signals, 1); % Number of signals
disp('Original Signals:');
disp(signals);
%% Step 2: Implement Gram-Schmidt Orthogonalization Procedure
[basis, coefficients] = gramSchmidt(signals);
disp('Orthonormal Basis Vectors:');
disp(basis);
disp('Signal Coefficients in Basis Representation:');
disp(coefficients);
%% Step 3: Plot the original signals and basis vectors
figure('Name', 'Signal and Basis Comparison', 'Position', [100 100 800 600]);
% Plot original signals
subplot(2,1,1);
hold on;
colors = ['r', 'g', 'b', 'm', 'c']; % Color codes for plotting
for i = 1:num_signals
 stem(signals(i,:), 'LineWidth', 2, 'Marker', 'o', ...
 'DisplayName', ['Signal ', num2str(i)], 'Color', colors(i));
end
title('Original Signals');
xlabel('Time Index');
ylabel('Amplitude');
legend('Location', 'best');
grid on;
hold off;
% Plot basis vectors
subplot(2,1,2);
hold on;
for i = 1:size(basis,1)
 stem(basis(i,:), 'LineWidth', 2, 'Marker', 's', ...
 'DisplayName', ['Basis ', num2str(i)], 'Color', colors(i));
end
title('Orthonormal Basis Vectors');
xlabel('Time Index');
ylabel('Amplitude');
legend('Location', 'best');
grid on;
```

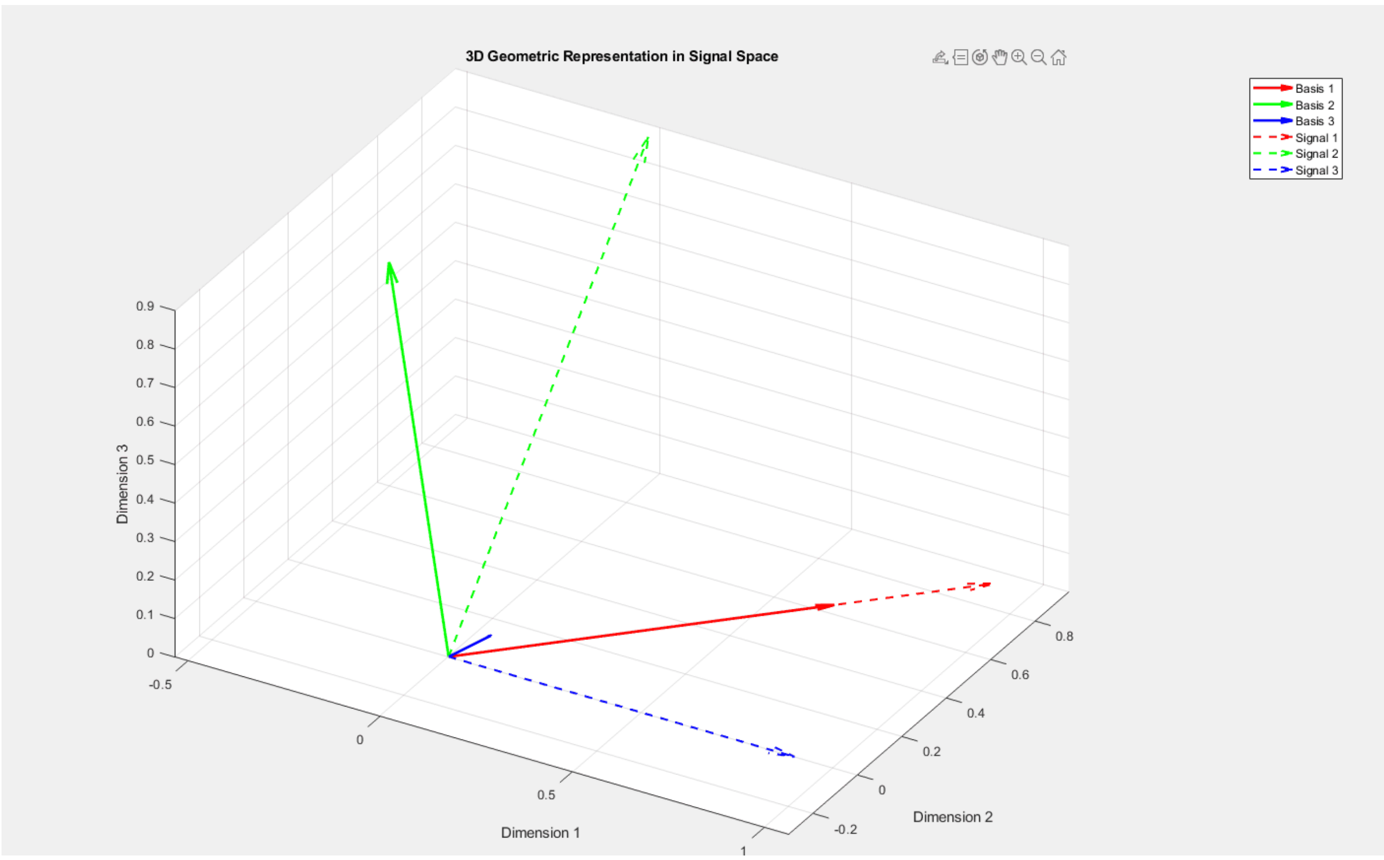
```
hold off;
%% Step 4: Geometric Representation (for 2D or 3D cases)
signal_space_dim = size(basis,1);
if signal_space_dim == 2
 % 2D Visualization
 figure('Name', '2D Geometric Representation');
 hold on;

 % Plot basis vectors
 quiver(0, 0, basis(1,1), basis(1,2), 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
 quiver(0, 0, basis(2,1), basis(2,2), 'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);

 % Plot original signals in this basis
 for i = 1:num_signals
 signal_vec = coefficients(i,1)*basis(1,:) + coefficients(i,2)*basis(2,:);
 quiver(0, 0, signal_vec(1), signal_vec(2), '--', 'LineWidth', 1.5, ...
 'Color', colors(i), 'DisplayName', ['Signal ', num2str(i)]);
 end

 xlabel('Dimension 1');
 ylabel('Dimension 2');
 title('2D Geometric Representation in Signal Space');
 legend('Basis 1', 'Basis 2', 'Signal 1', 'Signal 2', 'Signal 3');
 grid on;
 axis equal;
 hold off;

elseif signal_space_dim == 3
 % 3D Visualization
 figure('Name', '3D Geometric Representation');
 hold on;

 % Plot basis vectors
 quiver3(0, 0, 0, basis(1,1), basis(1,2), basis(1,3), 'r', 'LineWidth', 2);
 quiver3(0, 0, 0, basis(2,1), basis(2,2), basis(2,3), 'g', 'LineWidth', 2);
 quiver3(0, 0, 0, basis(3,1), basis(3,2), basis(3,3), 'b', 'LineWidth', 2);

 % Plot original signals in this basis
 for i = 1:num_signals
 signal_vec = coefficients(i,1)*basis(1,:) + coefficients(i,2)*basis(2,:) + coefficients(i,3)*basis(3,:);
 quiver3(0, 0, 0, signal_vec(1), signal_vec(2), signal_vec(3), '--', ...
 'LineWidth', 1.5, 'Color', colors(i), 'DisplayName', ['Signal ', num2str(i)]);
 end

 xlabel('Dimension 1');
 ylabel('Dimension 2');
 zlabel('Dimension 3');
 title('3D Geometric Representation in Signal Space');
 legend('Basis 1', 'Basis 2', 'Basis 3', 'Signal 1', 'Signal 2', 'Signal 3');
 grid on;
 axis equal;
 view(30, 30); % Set viewing angle
```

```
 hold off;
else
 disp(['Signal space has ', num2str(signal_space_dim), ' dimensions - cannot plot directly']);
end
%% Step 5: Verify Orthonormality of Basis Vectors
% Check that all basis vectors are orthogonal and normalized
orth_check = basis * basis';
disp('Orthonormality Check (should be identity matrix):');
disp(orth_check);
%% Gram-Schmidt Orthogonalization Function
function [basis, coefficients] = gramSchmidt(signals)
 [num_signals, len] = size(signals);
 basis = zeros(num_signals, len);
 coefficients = zeros(num_signals, num_signals);

 for i = 1:num_signals
 % Start with the original signal
 v = signals(i,:);

 % Subtract projections onto all previous basis vectors
 for j = 1:i-1
 projection = dot(v, basis(j,:)) / dot(basis(j,:), basis(j,:));
 v = v - projection * basis(j,:);
 end

 % Normalize if not zero vector
 norm_v = norm(v);
 if norm_v > eps % eps is MATLAB's floating-point relative accuracy
 basis(i,:) = v / norm_v;
 else
 basis(i,:) = v;
 end

 % Compute coefficients for all signals up to current basis vector
 for k = 1:num_signals
 for m = 1:i
 coefficients(k,m) = dot(signals(k,:), basis(m,:));
 end
 end
 end

 % Remove zero rows from basis (if any)
 basis = basis(any(basis,2),:);
end
```

Model Graph:



Original Signals
Amplitude
Time Index
Signal 1
Signal 2
Signal 3
Orthonormal Basis Vectors
Amplitude
Time Index
Basis 1
Basis 2
Basis 3



3D Geometric Representation in Signal Space
Basis 1
Basis 2
Basis 3
Signal 1
Signal 2
Signal 3
Dimension 1
Dimension 2
Dimension 3

Result:

1. Console Output:
    - The orthonormal basis vectors
    - The coefficients representing each original signal in terms of the basis
2. Figures:
    - Plot comparing original signals and derived basis vectors
    - 3D geometric representation (for 3-signal case) showing the signal space

Viva Questions:

1. What is the purpose of the Gram-Schmidt orthogonalization process?
2. How does orthogonalization help in signal representation?
3. What is the difference between orthogonal and orthonormal vectors?
4. Why do we need basis vectors in signal processing?
5. What is the geometric interpretation of signals in vector space?

Experiment 7

ASK (OOK) and BPSK Modulation & Demodulation with BER Performance Comparison

Aim: To implement and compare the performance of ASK (OOK) and BPSK modulation techniques in terms of Bit Error Rate (BER) using MATLAB simulation.

Apparatus:

- MATLAB Software
- Computer System

Theory:

1. Initialization & Parameters

1. System Parameters:
   - Sampling frequency (Fs = 1000 Hz).
   - Bit duration (Tb = 1 sec).
   - Number of bits for visualization (N_vis = 6).
   - Number of bits for BER analysis (N_ber = 1e5).
   - Carrier frequencies (Fc_bpsk = 5 Hz, Fc_ask = 10 Hz).
   - SNR range (EbN0_dB = 0:1:10 dB).
2. Time Vector:
   - t = 0:1/Fs:Tb-1/Fs; % Time axis for one bit

2. Waveform Generation (Visualization)

1. Input Data:
   - data_vis = [1 0 1 0 1 0]; % Alternating bit pattern
2. BPSK Modulation:
   - For each bit:
     - Bit '1': +sin(2πFc_bpsk t)
     - Bit '0': -sin(2πFc_bpsk t)
   - carrier = sin(2*pi*Fc_bpsk*t);
     s = (2*data_vis(i)-1) * carrier; % Maps {0,1} to {-1,+1}
3. ASK Modulation:
   - For each bit:

- Bit '1': cos(2πFc_ask t)
- Bit '0': 0 (no signal)

- if data_vis(i) == 1
    s = cos(2*pi*Fc_ask*t);
  else
    s = zeros(1, length(t));
  end

3. BER Analysis

1. Generate Random Bits:
   - bits = randi([0 1], 1, N_ber);
2. Loop Over SNR Values:
   - Convert EbN0_dB to linear scale: EbN0 = 10^(EbN0_dB(i)/10).
3. ASK BER Calculation:
   - Transmit: ask_rx = bits + noise_ask
   - Demodulate: Threshold at 0.5
   - ask_demod = ask_rx > 0.5;
     BER_ask(i) = sum(ask_demod ~= bits)/N_ber;
   - Theoretical BER:
   - $Pe\text{ASK}=Q(Eb/N0)$
   - BER_ask_theory(i) = qfunc(sqrt(EbN0));
4. BPSK BER Calculation:
   - Transmit: Map {0,1} to {-1,+1} → bpsk_mod = 2*bits - 1
   - Demodulate: Threshold at 0
   - bpsk_demod = bpsk_rx > 0;
     BER_bpsk(i) = sum(bpsk_demod ~= bits)/N_ber;
   - Theoretical BER:
   - $Pe\text{BPSK}=Q(2Eb/N0)$
   - BER_bpsk_theory(i) = qfunc(sqrt(2*EbN0));

4. Plotting Results

1. Subplot 1: Original binary data (stairs plot).
2. Subplot 2: BPSK modulated signal (time-domain).
3. Subplot 3: ASK modulated signal (time-domain).
4. Subplot 4: BPSK demodulated data (compared to original).
5. Subplot 5: ASK demodulated data (compared to original).
6. Subplot 6: BER performance (simulated vs. theoretical).

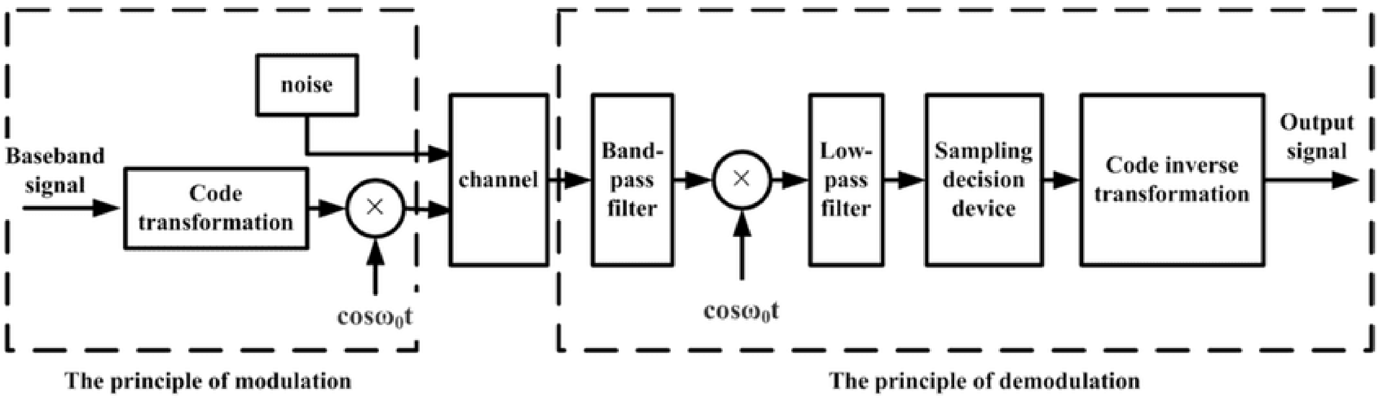


Fig: ASK Modulation and Demodulation

BER in BPSK (Binary Phase Shift Keying)

In BPSK, each bit is represented by one of two phase values of the carrier:

- Bit 1 → 0∘ phase
- Bit 0 → 180∘ phase (i.e., the signal is inverted)

The theoretical BER for BPSK in an AWGN channel is:

BERBPSK=21⋅erfc(N0Eb)

BPSK performs better than ASK in noisy environments due to its phase-based encoding, offering lower BER at the same SNR.

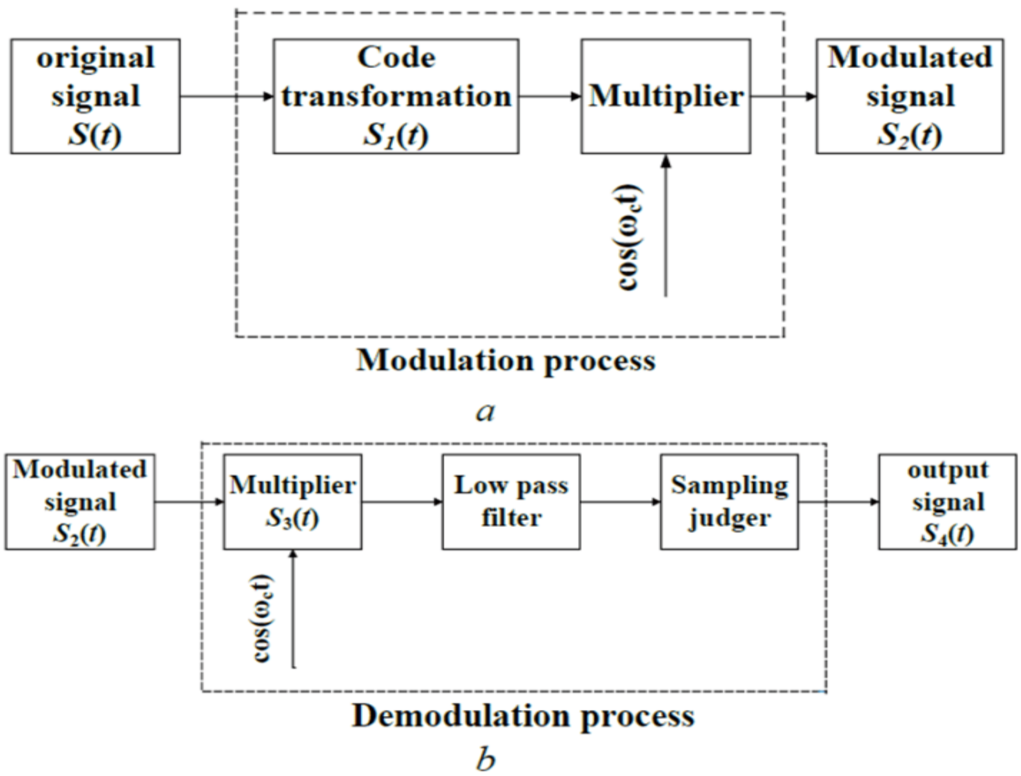


Fig: BPSK Modulation and Demodulation

○

Code:

```
clc;
clear;
close all;
%% Simulation Parameters
Fs = 1000; % Sampling frequency (Hz)
Tb = 1; % Bit duration (s)
N_vis = 6; % Number of bits for visualization
N_ber = 1e5; % Number of bits for BER analysis
t = 0:1/Fs:Tb-1/Fs; % Time vector for one bit
EbN0_dB = 0:1:10; % SNR range in dB
Fc_bpsk = 5; % Carrier frequency for BPSK
Fc_ask = 10; % Carrier frequency for ASK
%% Waveform Generation (for visualization)
data_vis = [1 0 1 0 1 0]; % Alternating pattern for clear transitions
% BPSK Modulation for visualization
bpsk_modulated = [];
time_axis = [];
for i = 1:N_vis
 carrier = sin(2*pi*Fc_bpsk*t);
 s = (2*data_vis(i)-1) * carrier;
 bpsk_modulated = [bpsk_modulated s];
 time_axis = [time_axis t + (i-1)*Tb];
end
% ASK Modulation for visualization
ask_modulated = [];
for i = 1:N_vis
 if data_vis(i) == 1
 s = cos(2*pi*Fc_ask*t);
 else
 s = zeros(1, length(t));
 end
 ask_modulated = [ask_modulated s];
end
%% BER Analysis
bits = randi([0 1], 1, N_ber);
% Initialize BER storage
BER_ask = zeros(size(EbN0_dB));
BER_bpsk = zeros(size(EbN0_dB));
BER_ask_theory = zeros(size(EbN0_dB));
BER_bpsk_theory = zeros(size(EbN0_dB));
```

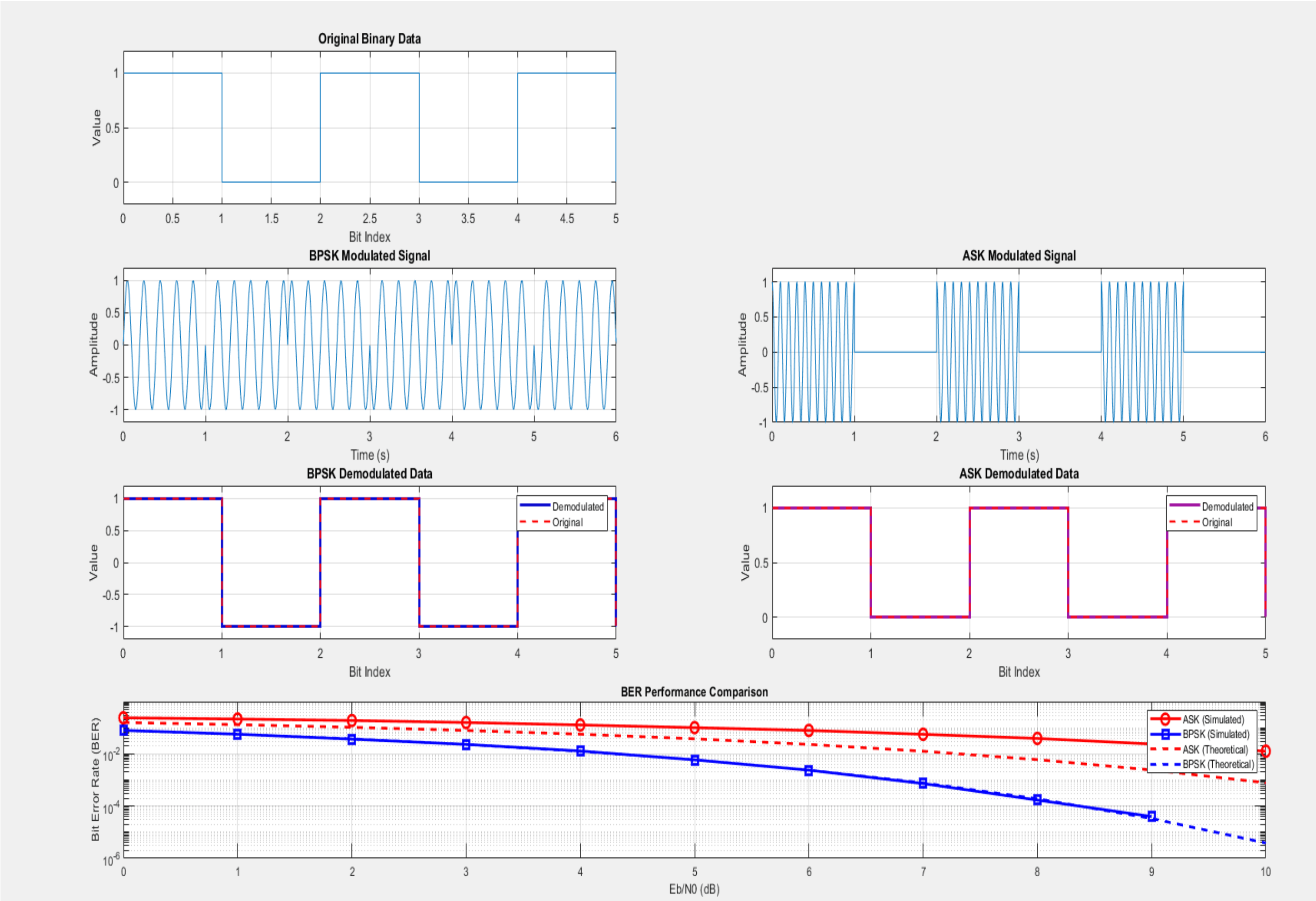
```
for i = 1:length(EbN0_dB)
 % SNR to noise variance conversion
 EbN0 = 10^(EbN0_dB(i)/10);

 %% ASK BER Calculation
 noise_power_ask = 1/(2*EbN0);
 noise_ask = sqrt(noise_power_ask)*randn(1, N_ber);
 ask_rx = bits + noise_ask;
 ask_demod = ask_rx > 0.5;
 BER_ask(i) = sum(ask_demod ~= bits)/N_ber;
 BER_ask_theory(i) = qfunc(sqrt(EbN0));

 %% BPSK BER Calculation
 bpsk_mod = 2*bits - 1;
 noise_power_bpsk = 1/(2*EbN0);
 noise_bpsk = sqrt(noise_power_bpsk)*randn(1, N_ber);
 bpsk_rx = bpsk_mod + noise_bpsk;
 bpsk_demod = bpsk_rx > 0;
 BER_bpsk(i) = sum(bpsk_demod ~= bits)/N_ber;
 BER_bpsk_theory(i) = qfunc(sqrt(2*EbN0));
end
figure;
% 1. Original Binary Data
subplot(4,2,1);
stairs(0:N_vis-1, data_vis);
ylim([-0.2 1.2]);
title('Original Binary Data');
xlabel('Bit Index');
ylabel('Value');
grid on;
% 2. BPSK Modulated Signal
subplot(4,2,3);
plot(time_axis, bpsk_modulated);
hold on;
title('BPSK Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
ylim([-1.2 1.2]);
grid on;
% 3. ASK Modulated Signal
subplot(4,2,4);
plot(time_axis, ask_modulated);
hold on;
```

```
title('ASK Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
ylim([-0.2 1.2]);
grid on;
% 4. BPSK Demodulated Data
subplot(4,2,5);
stairs(0:N_vis-1, (2*data_vis-1), 'LineWidth', 2, 'Color', [0 0 0.8]);
hold on;
stairs(0:N_vis-1, (2*data_vis-1), 'r--', 'LineWidth', 1.5);
ylim([-1.2 1.2]);
title('BPSK Demodulated Data');
xlabel('Bit Index');
ylabel('Value');
legend('Demodulated', 'Original');
grid on;
set(gca, 'XTick', 0:N_vis-1);
% 5. ASK Demodulated Data
subplot(4,2,6);
stairs(0:N_vis-1, data_vis, 'LineWidth', 2, 'Color', [0.6 0 0.6]);
hold on;
stairs(0:N_vis-1, data_vis, 'r--', 'LineWidth', 1.5);
ylim([-0.2 1.2]);
title('ASK Demodulated Data');
xlabel('Bit Index');
ylabel('Value');
legend('Demodulated', 'Original');
grid on;
set(gca, 'XTick', 0:N_vis-1);
% 6. BER Performance Plot
subplot(4,2,[7 8]);
semilogy(EbN0_dB, BER_ask, 'ro-', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
semilogy(EbN0_dB, BER_bpsk, 'bs-', 'LineWidth', 2, 'MarkerSize', 8);
semilogy(EbN0_dB, BER_ask_theory, 'r--', 'LineWidth', 2);
semilogy(EbN0_dB, BER_bpsk_theory, 'b--', 'LineWidth', 2);
grid on;
xlabel('Eb/N0 (dB)');
ylabel('Bit Error Rate (BER)');
title('BER Performance Comparison');
legend('ASK (Simulated)', 'BPSK (Simulated)', 'ASK (Theoretical)', 'BPSK (Theoretical)');
axis([0 10 1e-6 1]);
```

Procedure:

- Generate random binary data of size N.
- Modulate the signal using both:
  - ASK (OOK)
  - BPSK
- Add white Gaussian noise to simulate a real communication channel.
- Demodulate the received signal and recover the original binary data.
- Compare transmitted and received data to compute the BER.

Model Graph:



Result:

- ASK and BPSK modulated and demodulated signal are generated.

Viva Questions:

1. What is On-Off Keying (OOK), and where is it used?
2. How does BPSK improve noise immunity compared to ASK?
3. Derive the expression for BER in BPSK.
4. What is the role of matched filters in digital demodulation?
5. How would you experimentally compare BER performance between ASK and BPSK?

Experiment 8
M-PSK and QAM Modulation and BER Comparison

AIM: - To implement and compare the performance of M-ary Phase Shift Keying (M-PSK) and Quadrature Amplitude Modulation (QAM) in terms of Bit Error Rate (BER).

APPARATUS:-
- MATLAB Software
- Computer System

Theory:-

1. M-PSK (M-ary Phase Shift Keying)

Modulator:

- M-PSK encodes information in the **phase** of a carrier signal.
- Each symbol represents log2M bits, where M is the number of phases.
- The carrier signal:
- s(t)=Acos(2πfct+θm),θm=M2πm,m=0,1,…,M−1
- Example:
  - BPSK (M=2): θ=0,π
  - QPSK (M=4): θ=0,π/2,π,3π/2

Demodulator:

- **Coherent detection** is typically used.
- The received signal is correlated with reference signals of each possible phase.
- The phase closest to the received one is selected.

BER (for AWGN channel):

- Theoretical BER (approximate for large M):
- Pb≈log2M2Q(2log2M⋅N0Eb⋅sin(Mπ))
- As **M increases**, BER worsens at fixed Eb/N0 due to reduced angular separation.

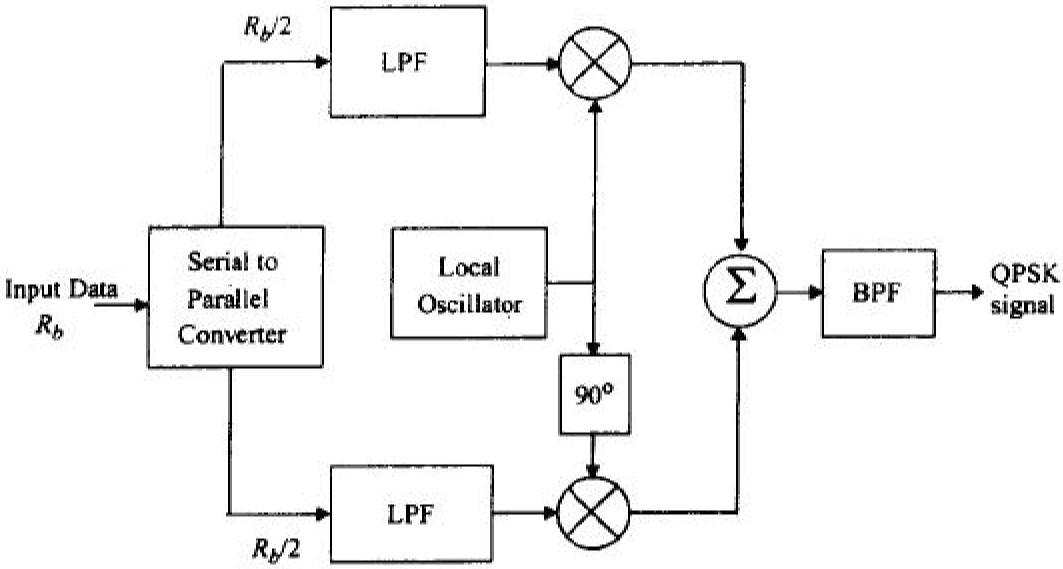


Fig: MPSK Modulation and Demodulation

2. QAM (Quadrature Amplitude Modulation)

Modulator:

- Combines both **amplitude** and **phase** variation.
- QAM symbols are arranged in a 2D grid (constellation).
- Example: 16-QAM has 16 points (4 amplitude levels on I and Q axes).
- Transmit signal:
- s(t)=I⋅cos(2πfct)−Q⋅sin(2πfct)
- where I and Q are from the symbol mapping.

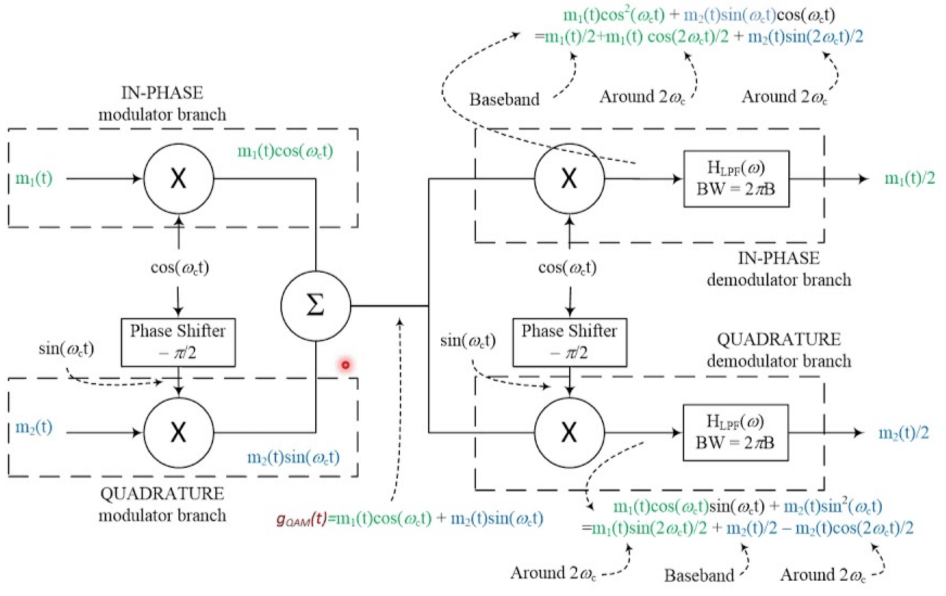
Demodulator:

- **Coherent detection**.
- Correlates received signal with carrier to get I and Q components.
- Decides on the nearest constellation point.

BER (for square M-QAM over AWGN):

- Approximate BER:
- Pb≈log2M4(1−M1)Q(M−13log2M⋅N0Eb)
- QAM offers better **spectral efficiency** than PSK at the cost of **higher noise sensitivity**.

QAM Modulation and Demodulation



QAM Modulator/Demodulator

3

Algorithm:

1. Initialization & Parameters

1. System Parameters:

- Number of bits (N = 1e5).
- SNR range (EbN0_dB = 0:2:20 dB).
- Modulation schemes (modSchemes = {'BPSK', 'QPSK', '8-PSK', '16-QAM', '64-QAM'}).
- Constellation sizes (M_list = [2, 4, 8, 16, 64]).

2. Preallocate BER Matrix:

- ber = zeros(length(M_list), length(EbN0_dB));

2. BER Calculation Loop

A. Loop Over Modulation Schemes

- For each modulation scheme (k = 1:length(M_list)):

1. Extract Parameters:

- Constellation size (M = M_list(k)).
- Bits per symbol (bitsPerSymbol = log2(M)).

2. Generate Random Bits:

- bits = randi([0 1], N, 1);
- Padding: Ensure bit length is a multiple of bitsPerSymbol.

3. Convert Bits to Symbols:

- symbols = bi2de(reshape(bits, [], bitsPerSymbol), 'left-msb');

4. Modulation:

- For M-PSK:

- tx = pskmod(symbols, M, pi/M, 'gray'); % Gray-coded PSK
- For M-QAM:
- tx = qammod(symbols, M, 'gray'); % Gray-coded QAM

B. Loop Over SNR Values

For each EbN0_dB(i):

1. Convert Eb/N0 to SNR per Symbol:
   - SNR=Eb/N0+10log10(bitsPerSymbol)
   - SNR = EbN0 + 10*log10(bitsPerSymbol);
2. AWGN Channel:
   - rx = awgn(tx, SNR, 'measured');
3. Demodulation:
   - For M-PSK:
   - rxSymbols = pskdemod(rx, M, pi/M, 'gray');
   - For M-QAM:
   - rxSymbols = qamdemod(rx, M, 'gray');
4. Convert Symbols Back to Bits:
   - rxBits = de2bi(rxSymbols, bitsPerSymbol, 'left-msb');
     rxBits = rxBits(:);
5. BER Calculation:
   - BER=Total bitsNumber of bit errors
   - ber(k, i) = sum(bits ~= rxBits) / length(bits);

3. Plotting Results

1. Semilog Plot of BER vs. Eb/N0:
   - semilogy(EbN0_dB, ber(1,:), '-o', ... % BPSK
              EbN0_dB, ber(2,:), '-s', ... % QPSK
              EbN0_dB, ber(3,:), '-^', ... % 8-PSK
              EbN0_dB, ber(4,:), '-d', ... % 16-QAM
              EbN0_dB, ber(5,:), '-x', ... % 64-QAM
              'LineWidth', 1.5);
2. Labels & Legend:
   - legend(modSchemes, 'Location', 'southwest');
     xlabel('Eb/N0 (dB)');
     ylabel('Bit Error Rate (BER)');
     title('BER Performance: M-PSK and M-QAM');

Code:

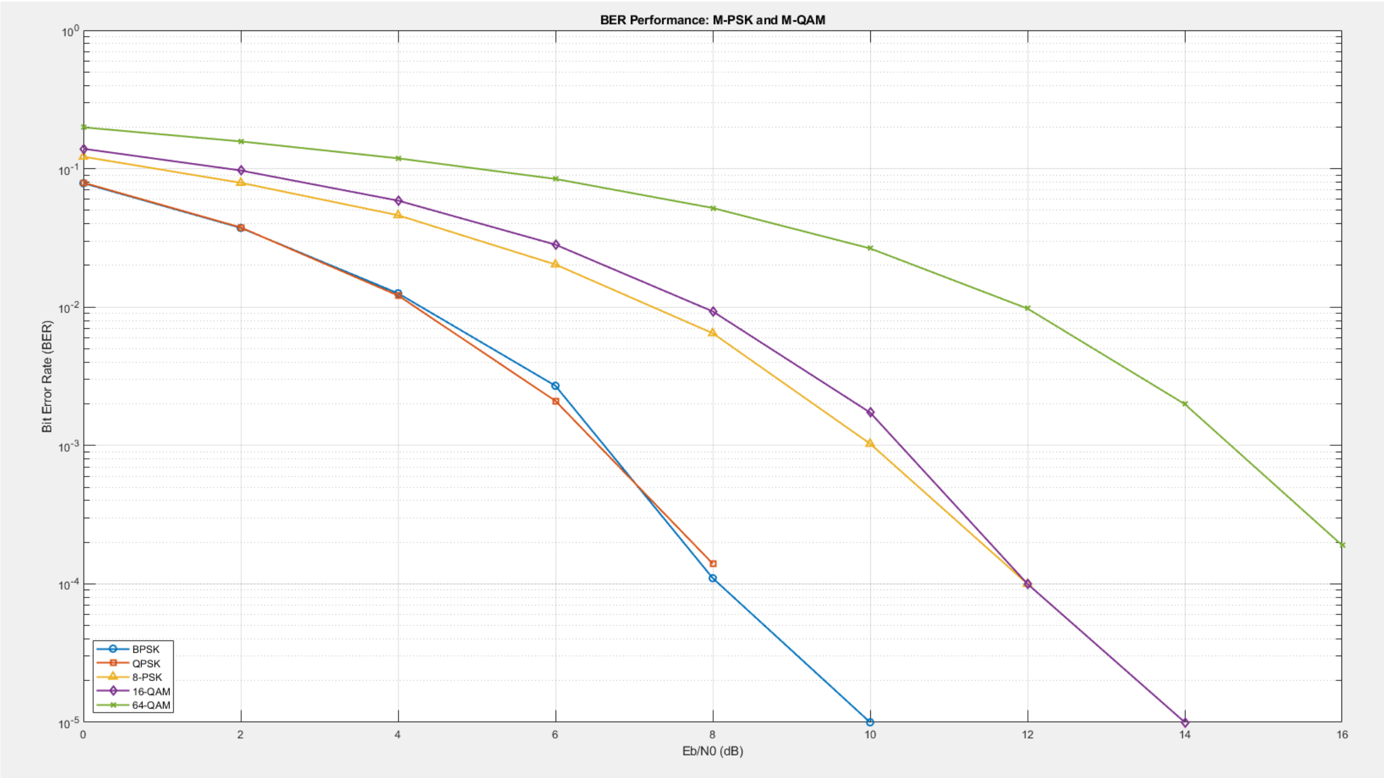
```
clc; clear; close all;
% Parameters
N = 1e5; % Number of bits
EbN0_dB = 0:2:20; % Eb/N0 range in dB
modSchemes = {'BPSK', 'QPSK', '8-PSK', '16-QAM', '64-QAM'};
M_list = [2, 4, 8, 16, 64];
% Preallocate BER matrix
ber = zeros(length(M_list), length(EbN0_dB));
% Loop over modulation schemes
for k = 1:length(M_list)
 M = M_list(k);
 bitsPerSymbol = log2(M);

 % Generate random bits
 bits = randi([0 1], N, 1);
 % Pad bits to make it multiple of bitsPerSymbol
 if mod(length(bits), bitsPerSymbol) ~= 0
 bits = [bits; zeros(bitsPerSymbol - mod(length(bits), bitsPerSymbol), 1)];
 end
 % Reshape and convert to symbols
 symbols = bi2de(reshape(bits, [], bitsPerSymbol), 'left-msb');

 % Modulation
 if contains(modSchemes{k}, 'PSK')
 tx = pskmod(symbols, M, pi/M, 'gray');
 else
 tx = qammod(symbols, M, 'gray');
 end
 % Loop over Eb/N0
 for i = 1:length(EbN0_dB)
 % Calculate SNR per symbol
 EbN0 = EbN0_dB(i);
 SNR = EbN0 + 10*log10(bitsPerSymbol);

 % AWGN channel
 rx = awgn(tx, SNR, 'measured');

 % Demodulation
 if contains(modSchemes{k}, 'PSK')
 rxSymbols = pskdemod(rx, M, pi/M, 'gray');
 else
 rxSymbols = qamdemod(rx, M, 'gray');
```

```
 end

 % Convert symbols back to bits
 rxBits = de2bi(rxSymbols, bitsPerSymbol, 'left-msb');
 rxBits = rxBits(:);

 % Truncate extra padded bits
 rxBits = rxBits(1:length(bits));

 % BER Calculation
 ber(k, i) = sum(bits ~= rxBits) / length(bits);
 end
end
% Plot
figure;
semilogy(EbN0_dB, ber(1,:), '-o', ...
 EbN0_dB, ber(2,:), '-s', ...
 EbN0_dB, ber(3,:), '-^', ...
 EbN0_dB, ber(4,:), '-d', ...
 EbN0_dB, ber(5,:), '-x', 'LineWidth', 1.5);
grid on;
legend(modSchemes, 'Location', 'southwest');
xlabel('Eb/N0 (dB)');
ylabel('Bit Error Rate (BER)');
title('BER Performance: M-PSK and M-QAM');
```

PROCEDURE:

- Open MATLAB and create a new script file.
- Define the number of bits, Eb/N0 range, and modulation types (BPSK, QPSK, 8-PSK, 16-QAM, 64-QAM).
- Generate random bits for transmission.
- Modulate the bits using the corresponding PSK or QAM modulation scheme.
- Add AWGN noise to the transmitted signal for different Eb/N0 values.
- Demodulate the received noisy signal.
- Convert the demodulated symbols back to bits.
- Calculate the Bit Error Rate (BER) by comparing transmitted and received bits.
- Repeat steps 4–8 for all modulation schemes.
- Plot BER vs. Eb/N0 for each modulation scheme.

Model Graph:



Result: The simulation was successfully carried out for BPSK, QPSK, 8-PSK, 16-QAM, and 64-QAM modulation schemes over an AWGN channel. The Bit Error Rate (BER) was computed for each scheme over a range of Eb/N0 from 0 dB to 20 dB.

Viva Questions:

1. What is the difference between QPSK and 8-PSK?
2. How does QAM achieve higher data rates compared to PSK?
3. What is Gray coding, and why is it used in QAM?
4. How does increasing M in M-PSK affect BER performance?
5. Compare the bandwidth efficiency of 16-QAM and 16-PSK.

<u>Experiment 9</u>
To Study the Effects of ISI by Generating an Eye Pattern

<u>Aim:</u>To study the effects of Intersymbol Interference (ISI) in a digital communication system by generating and analyzing the Eye Pattern using MATLAB.

<u>Apparatus:</u>
- MATLAB Software
- Computer System

<u>Algorithm:</u>

1. Initialization & Parameters

1. System Parameters:
   - Number of bits (numBits = 1000).
   - Samples per symbol (sps = 8).
   - Roll-off factor (0.5 for raised cosine filter).
   - Filter span (6 symbols).
2. Generate Random Bits:
   - data = randi([0 1], numBits, 1); % Binary data stream

2. BPSK Modulation

1. Map Bits to Symbols:
   - ○ 0 → -1
   - ○ 1 → +1
   - modData = 2*data - 1; % BPSK symbols: [-1, +1]

3. Upsampling

1. Increase Sampling Rate:
   - Insert sps-1 zeros between symbols to match the desired samples per symbol.
   - upsampled = upsample(modData, sps); % e.g., [1, 0, 0, 0, -1, 0, 0, 0, ...]

4. Pulse Shaping (Raised Cosine Filter)

1. Design Filter:
   - filt = rcosdesign(0.5, 6, sps); % Roll-off=0.5, span=6 symbols
   - Why Raised Cosine?
   - Minimizes ISI by ensuring zero crossings at symbol intervals.
2. Apply Filter:
   - txSignal = conv(upsampled, filt, 'same'); % Convolve and truncate
   - The 'same' flag keeps the output length equal to upsampled.

5. Eye Diagram Generation

1. Plot Eye Diagram:
   - eyediagram(txSignal, 2*sps); % 2 symbols per trace
     title('Eye Diagram Showing ISI Effect');
   - Parameters:
     - txSignal: Pulse-shaped signal.
     - 2*sps: Ensures each trace covers 2 symbol periods for clear visualization.

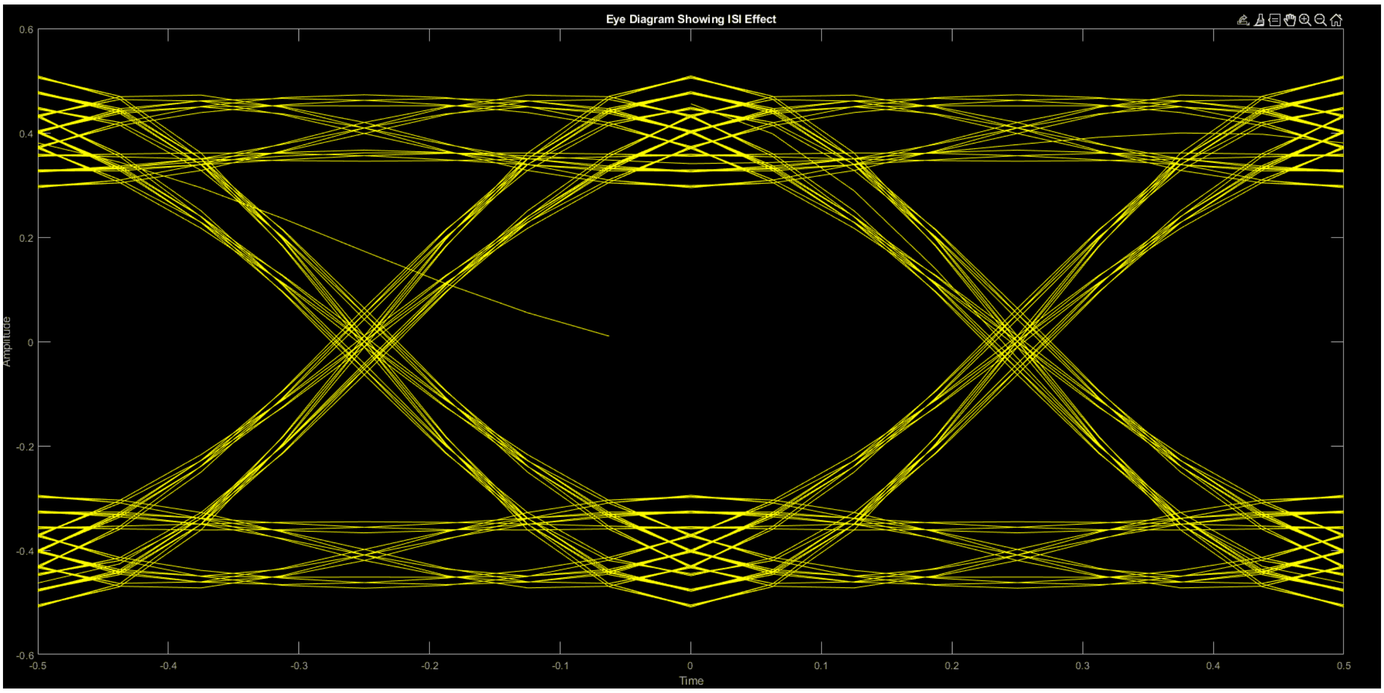
<u>Code:</u>

```
clc; clear; close all;

% Parameters
numBits = 1000;
sps = 8;  % Samples per symbol

% Generate random bits
data = randi([0 1], numBits, 1);

% BPSK modulation
modData = 2*data - 1; % Map 0->-1, 1->+1

% Upsample
upsampled = upsample(modData, sps);

% Pulse shaping filter (Raised Cosine)
filt = rcosdesign(0.5, 6, sps);  % Roll-off = 0.5, span = 6 symbols
txSignal = conv(upsampled, filt, 'same');

% Eye diagram
eyediagram(txSignal, 2*sps);
title('Eye Diagram Showing ISI Effect');
```

<u>Procedure:</u>

1. Open MATLAB and create a new script.
2. Generate a random binary bit stream.
3. Use pulse shaping (e.g., raised cosine filter) to simulate channel effects.
4. Pass the signal through a simulated channel (optional ISI can be introduced).
5. Use the eye diagram function to plot the Eye Pattern.
6. Analyze the eye opening, timing jitter, and possible ISI.

Eye Design:



Result:

- The Eye Pattern was successfully generated using MATLAB.
- The openness of the eye diagram indicates the level of ISI:
    - A wide open eye → minimal ISI
    - A partially or fully closed eye → severe ISI
- The shape of the eye allows assessment of timing jitter, amplitude distortion, and noise margin.

Viva Questions:

1. What causes ISI in digital communication systems?

2. How does an eye pattern help in analyzing signal quality?

3. What are the key parameters observed in an eye diagram?

4. How can ISI be mitigated in baseband transmission?

5. What is the role of raised cosine filtering in reducing ISI?

Experiment 10

Specifications and Characterization of Hardware Platforms like NooRadio, SDR

Aim: To study and document the technical specifications, features, and operational characteristics of wireless communication hardware platforms such as NooRadio, RTL-SDR, HackRF, and USRP (SDR devices).

Apparatus:

- MatlabSoftware
- PC or Computer for simulation

Theory:

- Software Defined Radios (SDRs) use software to perform signal processing tasks that were traditionally handled by hardware.
- Platforms like NooRadio and USRP offer real-time processing and flexible tuning across wide frequency ranges.
- Key features such as frequency range, sampling rate, ADC resolution, bandwidth, and host interface define the capability of each platform.

Algorithm:

1. FM Signal Transmission & Reception

A. Signal Generation & Modulation

1. Parameters:
    - Sampling rate (fs = 1 MHz).
    - Duration (duration = 0.1 sec).
    - FM deviation (fmDeviation = 75 kHz).
    - Audio tone (audioFreq = 1 kHz).
2. Generate Modulating Signal:
3. modSignal = sin(2*pi*audioFreq*t); *% 1 kHz sine wave*
4. FM Modulation:
5. fmModulator = comm.FMModulator('SampleRate', fs, 'FrequencyDeviation', fmDeviation);

   fmTxSignal = fmModulator(modSignal');

B. Channel Simulation (AWGN)

1. Add Noise:

2. SNR_dB = 30; *% Signal-to-noise ratio*
   fmRxSignal = awgn(fmTxSignal, SNR_dB, 'measured');

C. FM Demodulation

1. Recover Audio Signal:
2. fmDemodulator = comm.FMDemodulator('SampleRate', fs, 'FrequencyDeviation', fmDeviation);
   demodSignal = fmDemodulator(fmRxSignal);

D. Visualization (500-Sample Segments)

1. Plot:
   - Transmitted FM signal (clean).
   - Received FM signal (noisy).
   - Demodulated audio signal.

2. Spectrum Analysis (FFT)

1. Compute Power Spectral Density (PSD):
2. [Pxx, F] = pwelch(fmTxSignal, hann(nfft), nfft/2, nfft, fs, 'centered');
3. Plot Spectrum:
   - Shows frequency content of FM signal.
   - Centered around carrier frequency.

3. QPSK Digital Communication

A. Signal Generation & Modulation

1. Generate Random Bits:
2. dataBits = randi([0 1], numBits, 1);
3. QPSK Modulation:
4. qpskModulator = comm.QPSKModulator('BitInput', true);
   qpskTxSignal = qpskModulator(dataBits);

B. Channel Simulation (AWGN)

1. Add Noise:
2. qpskRxSignal = awgn(qpskTxSignal, SNR_dB, 'measured');

C. QPSK Demodulation & BER Calculation

1. Demodulate:
2. qpskDemodulator = comm.QPSKDemodulator('BitOutput', true);

   rxBits = qpskDemodulator(qpskRxSignal);
3. Compute BER:
4. [ber, numErrors] = biterr(dataBits, rxBits);

Code:

```
%% Clear workspace
clc; clear; close all;

%% 1. Simulate FM Signal Transmission & Reception
fs = 1e6;           % Sample rate (1 MHz)
duration = 0.1;      % Signal duration (seconds)
t = 0:1/fs:duration-1/fs;

% Generate FM-modulated signal
fmDeviation = 75e3;  % FM deviation (Hz)
audioFreq = 1e3;     % Simulated audio tone (1 kHz)
modSignal = sin(2*pi*audioFreq*t);
fmModulator = comm.FMModulator('SampleRate', fs, 'FrequencyDeviation', fmDeviation);
fmTxSignal = fmModulator(modSignal');

% Add noise (AWGN)
SNR_dB = 30;         % Signal-to-noise ratio (dB)
fmRxSignal = awgn(fmTxSignal, SNR_dB, 'measured');

% Demodulate FM
fmDemodulator = comm.FMDemodulator('SampleRate', fs, 'FrequencyDeviation', fmDeviation);
demodSignal = fmDemodulator(fmRxSignal);

%% 2. Plot Representative Segments (Not Entire Signal)
samples_to_show = 500; % Only show 500 samples for clarity
start_sample = 1000;   % Start at sample 1000 to skip transient

figure;
```

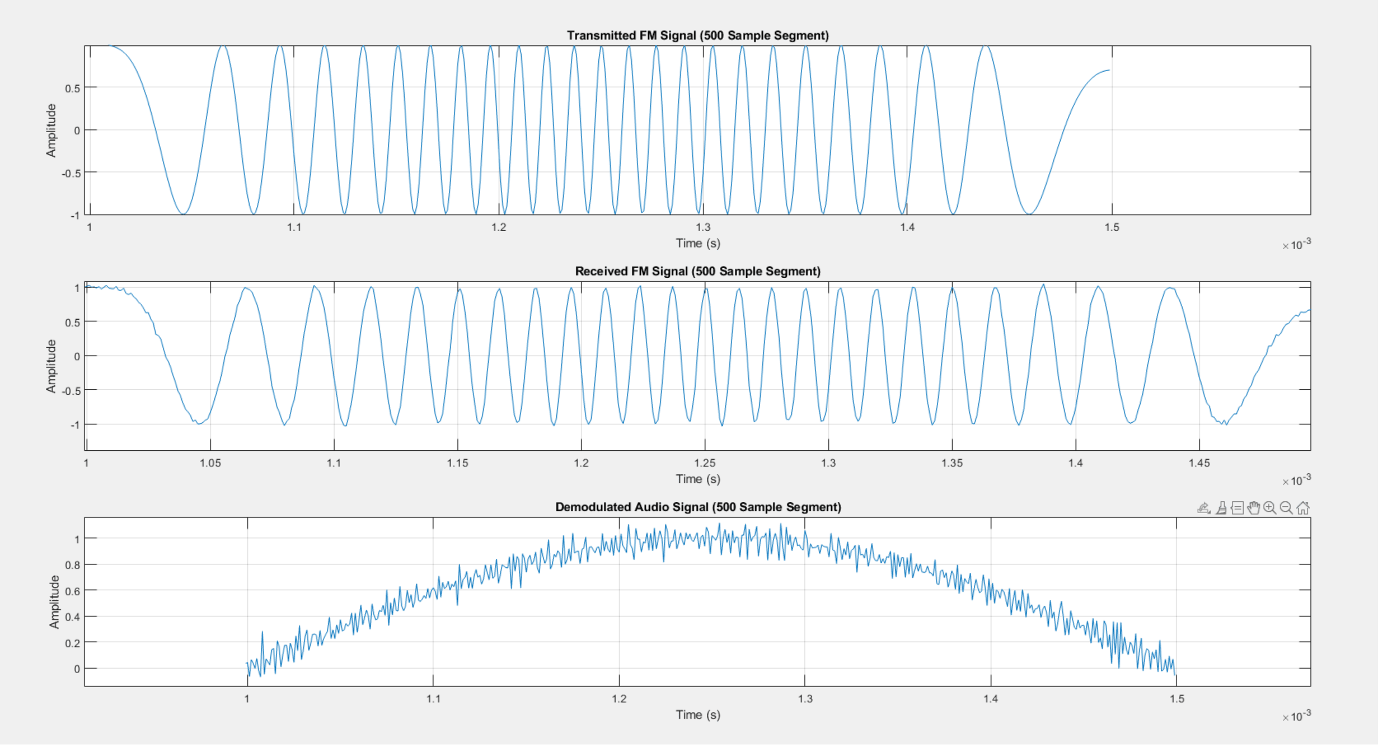
```
% Transmitted FM Signal (Clean)
subplot(3,1,1);
plot(t(start_sample:start_sample+samples_to_show), ...
    real(fmTxSignal(start_sample:start_sample+samples_to_show)));
title('Transmitted FM Signal (500 Sample Segment)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Received FM Signal (Noisy)
subplot(3,1,2);
plot(t(start_sample:start_sample+samples_to_show), ...
    real(fmRxSignal(start_sample:start_sample+samples_to_show)));
title('Received FM Signal (500 Sample Segment)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Demodulated Audio Signal
subplot(3,1,3);
plot(t(start_sample:start_sample+samples_to_show), ...
    demodSignal(start_sample:start_sample+samples_to_show));
title('Demodulated Audio Signal (500 Sample Segment)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

%% 3. Spectrum Analysis (FFT)
figure;
nfft = 1024;
[Pxx, F] = pwelch(fmTxSignal, hann(nfft), nfft/2, nfft, fs, 'centered');
plot(F/1e3, 10*log10(Pxx));
title('Spectrum of FM Signal');
xlabel('Frequency (kHz)');
ylabel('Power (dB)');
grid on;

%% 4. QPSK Simulation (Unchanged)
numBits = 1000;
dataBits = randi([0 1], numBits, 1);

% QPSK Modulation
qpskModulator = comm.QPSKModulator('BitInput', true);
qpskTxSignal = qpskModulator(dataBits);

% Add noise
qpskRxSignal = awgn(qpskTxSignal, SNR_dB, 'measured');

% QPSK Demodulation
qpskDemodulator = comm.QPSKDemodulator('BitOutput', true);
rxBits = qpskDemodulator(qpskRxSignal);
```

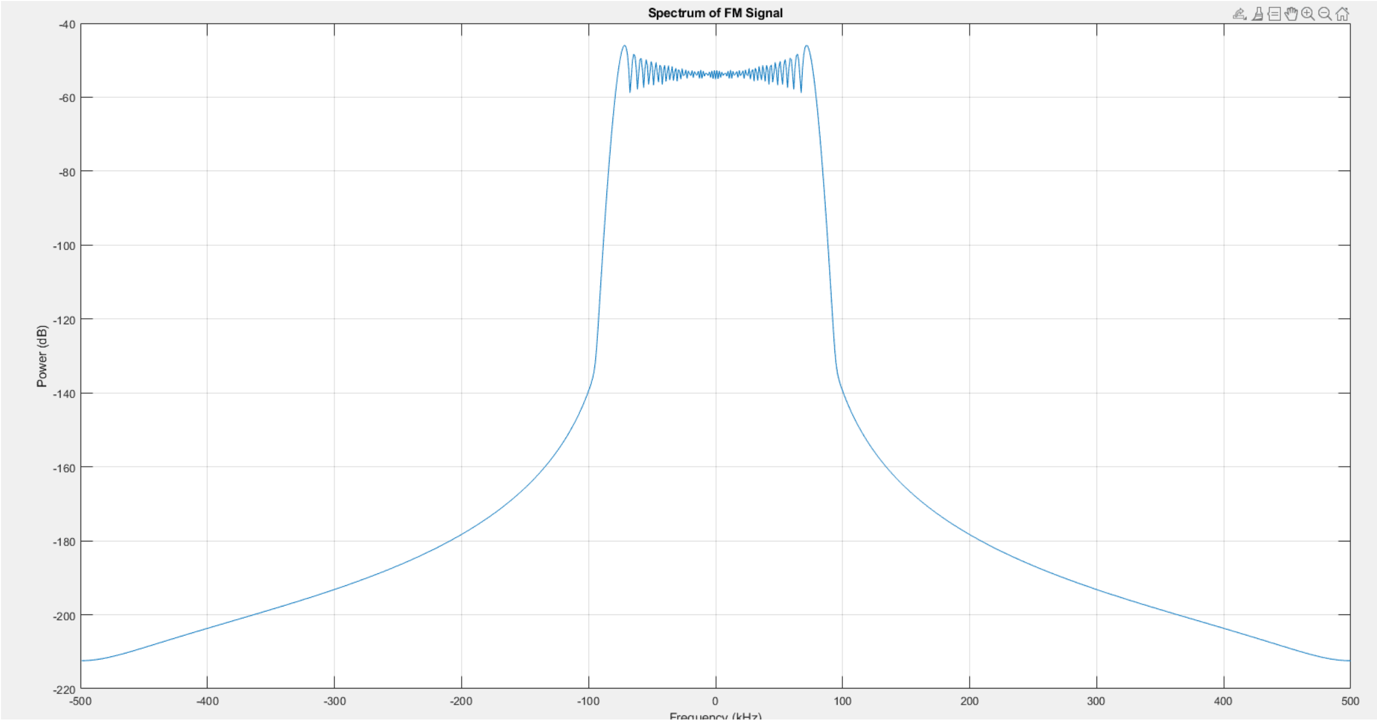
```
% Calculate BER
[ber, numErrors] = biterr(dataBits, rxBits);
disp(['Bit Error Rate (BER): ', num2str(ber)]);
disp(['Number of Errors: ', num2str(numErrors)]);
```

Procedure:

- Select 2–3 SDR platforms (e.g., RTL-SDR, NooRadio, HackRF, USRP).
- Visit their official websites or datasheets and collect key specifications:
  - Frequency range
  - ADC resolution
  - Bandwidth
  - Supported sample rate
  - Interface (USB, Ethernet, etc.)
  - Compatibility with software (MATLAB, GNU Radio)
- Tabulate and compare the specifications.
- Use MATLAB or GNU Radio (simulation only) to:
  - Generate a sine wave or QPSK signal.
  - Visualize the signal spectrum.
  - Apply noise and measure SNR or BER (using built-in MATLAB tools).
- Discuss which SDR would be suitable for different applications based on your findings (e.g., FM radio, Wi-Fi, 5G, etc.).

Model Graph:





Result:

- Transmitted FM waveform should show modulated signal.
- Received waveform shows signal with noise.
- Demodulated audio signal should resemble original 1 kHz tone.
- Spectrum plot should display FM bandwidth (~150 kHz).

Viva Questions:

1. What is the role of Frequency Deviation in FM?
2. How does noise affect demodulated output?
3. What is the significance of BER in digital communication?
4. How does MATLAB simulate RF environments?
5. How is spectrum analysis useful in SDR characterization?

Experiment- 11

Establishment of Wireless Communication Link Using a Pair of SDR Platforms

Aim: To establish a point-to-point wireless communication link using a pair of Software Defined Radio (SDR) platforms and analyze the transmission and reception of signals in real-time.

Apparatus:

- Matlab
- Pc or Computer for Simulation

Code:

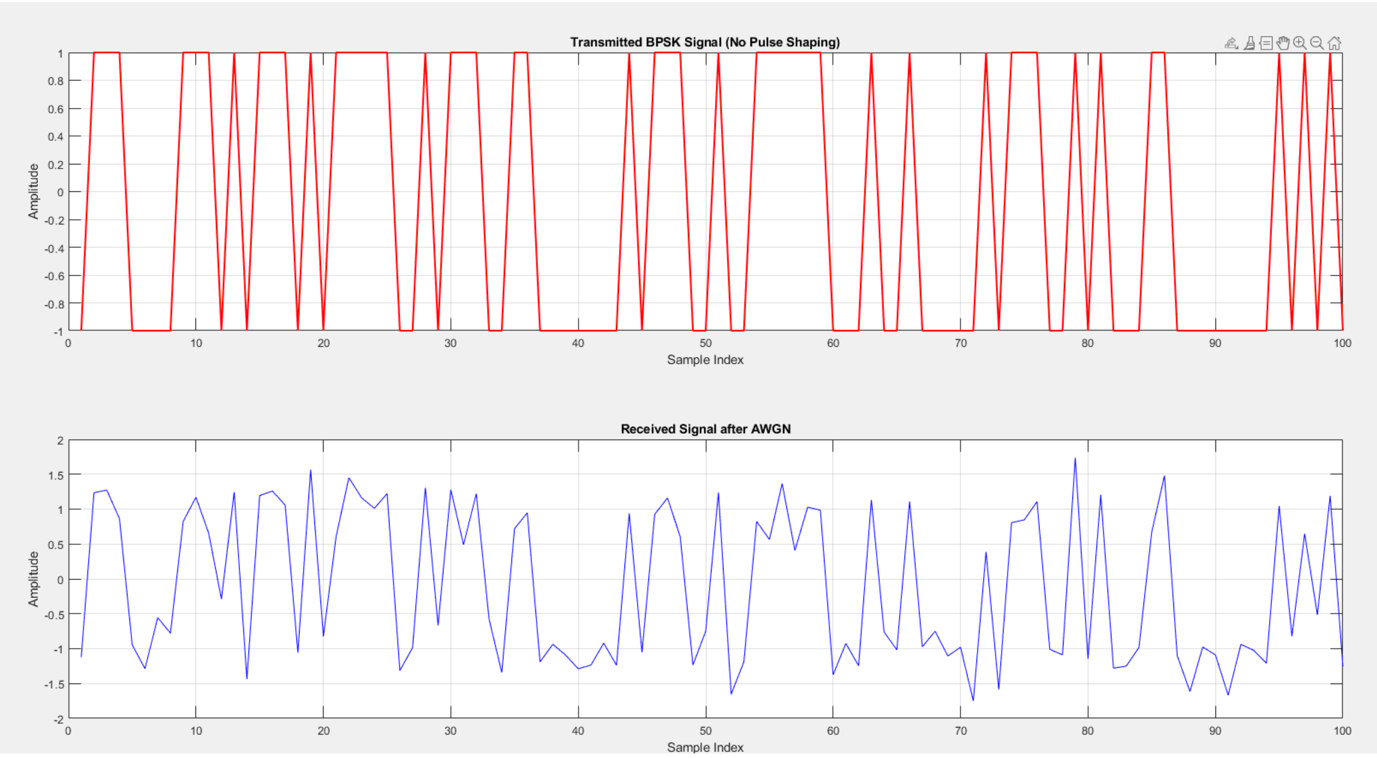
```
clc;
clear;
%% Parameters
numBits = 1000;
SNR_dB = 10;
snrVec = 0:2:30;
%% Transmitter
txBits = randi([0 1], numBits, 1);
txSymbols = 2 * txBits - 1; % BPSK: 0→-1, 1→+1
%% Channel (AWGN)
rxSignal = awgn(txSymbols, SNR_dB, 'measured');
%% Visualization
figure;
subplot(2,1,1);
plot(txSymbols(1:100), 'r', 'LineWidth', 1.5);
title('Transmitted BPSK Signal (No Pulse Shaping)');
xlabel('Sample Index'); ylabel('Amplitude'); grid on;
subplot(2,1,2);
plot(rxSignal(1:100), 'b');
title('Received Signal after AWGN');
xlabel('Sample Index'); ylabel('Amplitude'); grid on;
%% Receiver
rxBits = rxSignal > 0; % Threshold detection
rxBits = rxBits(1:length(txBits)); % Ensure length match
[errCount, BER] = biterr(txBits, rxBits);
disp(['Errors: ' num2str(errCount)]);
disp(['BER: ' num2str(BER)]);
```

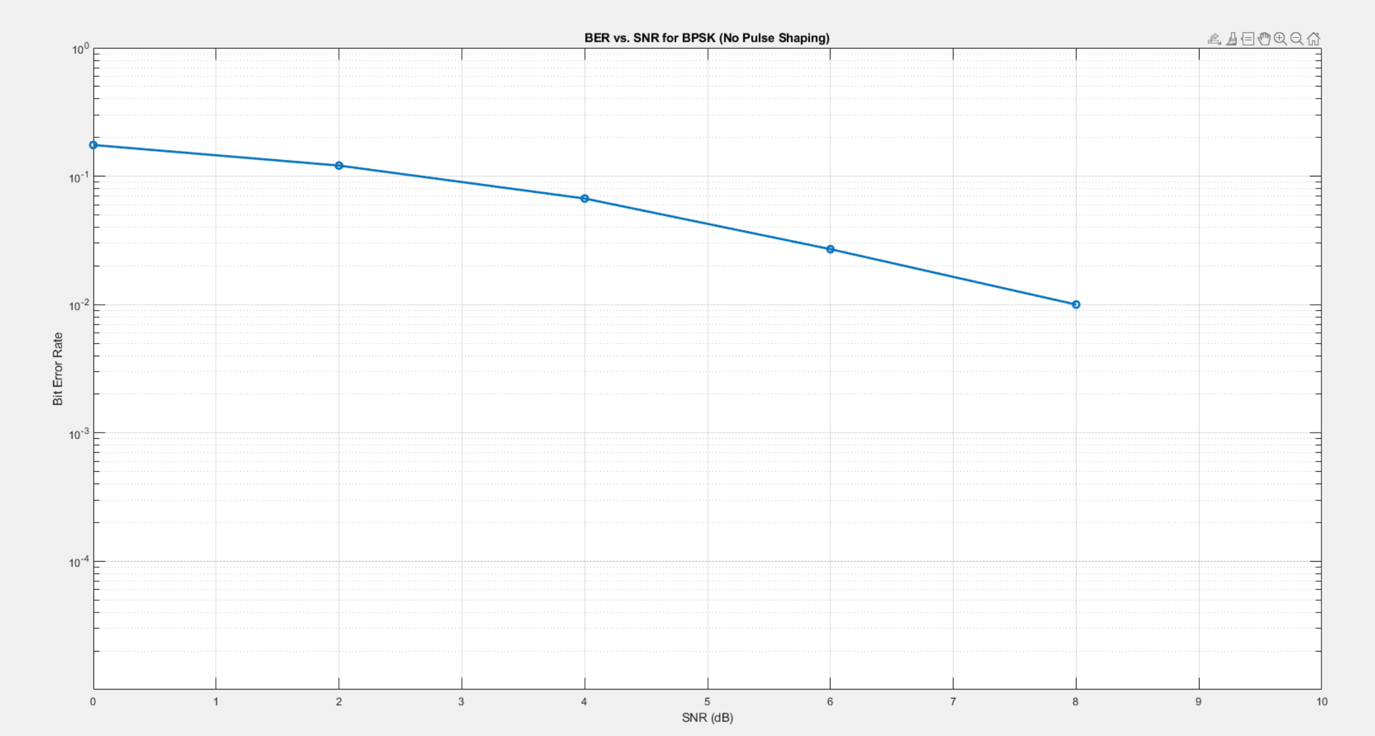
```
%% BER vs. SNR
berVec = zeros(size(snrVec));
for i = 1:length(snrVec)
 noisy = awgn(txSymbols, snrVec(i), 'measured');
 rxBits = noisy > 0;
 rxBits = rxBits(1:length(txBits));
 [~, berVec(i)] = biterr(txBits, rxBits);
end
figure;
semilogy(snrVec, berVec, '-o', 'LineWidth', 2);
xlabel('SNR (dB)'); ylabel('Bit Error Rate');
title('BER vs. SNR for BPSK (No Pulse Shaping)');
grid on; ylim([1e-5 1]);
```

Procedure:

- Create a binary data sequence of length N.
- Modulate the data using BPSK .Map bits {0,1} to symbols {-1,+1}.
- Transmit data through an AWGN channel Add Gaussian noise to the modulated signal at different SNR values.
- Demodulate the received signal and decide the received bits based on the sign of received samples.
- Calculate BER and compare the transmitted bits with received bits and compute the error rate.
- Plot BER vs SNR Repeat steps 3–5 for a range of SNR values and plot BER curve

Model Graph:





Result:

- Transmitted BPSK signal shows clear bit pattern.
- Received signal is corrupted with Gaussian noise.
- Detected bits are estimated using a threshold..

Viva Questions:

1. What is BPSK and how does it work?
2. Why do we use AWGN in channel modeling?
3. What is the difference between BER and SNR?
4. How does threshold detection work in BPSK?
5. What improvements can be made for real SDR transmission?