1. BPSK Modulation and Demodulation

```
clc;
clear all;
close all;
binary_sequence = [1 0 1 1 0 0 1 0 0 1];
number_bits = length(binary_sequence);
%Define Parameters
Eb = 2;
Tb = 1;
Ac = 1;
nc = 4;
fc = nc/Tb;
tf = 99;
t = 0:1/tf:1;
tn = 0:1/(tf+1):number_bits;
tt = tn(1,2:end);
%Carrier Signal
wc = 2*pi*fc;
xc = Ac*cos(wc*t);
%Polar NRZ  signal generation
```

```
NRZ = [];
for m = 1:number_bits
   if binary_sequence(m) == 1
      NRZ = [NRZ ones(1,length(t))];
   else
      NRZ = [NRZ -ones(1,length(t))];
   end
end
%BPSK Signal Generation
TX = [];
for n = 1:number_bits
   if binary_sequence(n) == 1
      TX = [TX sqrt(2*Eb/Tb)*cos(2*pi*fc*t)];
   else
      TX = [TX -sqrt(2*Eb/Tb)*cos(2*pi*fc*t)];
   end
end
%Manually AWGN noise add
SNR = 1;
Ps = mean(abs(TX).^2);
Pn = Ps/(10^(SNR/10));
```

```
noise = sqrt(Pn) * randn(1,length(TX));
RX = TX + noise;
%Coreherent Demodulation
LO = sqrt(2/Tb)*cos(2*pi*fc*t);
BINSEQDET = [];
CS = [];
for n = 1:number_bits
temp = RX([(n-1)*(tf+1)+1:n*(tf+1)]);
S = sum(temp.*LO);
CS = [CS S];
if (S>0)
   BINSEQDET = [BINSEQDET 1];
else
   BINSEQDET = [BINSEQDET 0];
end
end
bit_errors = sum(abs(BINSEQDET - binary_sequence));
disp(['Total Bit Errors: ', num2str(bit_errors)]);
figure(1)
plot(t,xc)
title('Carrier Signal')
```

```
xlabel('Time(s)');
ylabel('Amplitude');

figure(2)
subplot(2,1,1)
plot(tt,NRZ)
title('Polar NRZ input binary sequence')
xlabel('Time(s)');
ylabel('Amplitude');

subplot(2,1,2)
plot(tt,TX(1,1:length(tt)));
title('BPSK modulated signal')
xlabel('Time(s)');
ylabel('Amplitude');

figure(3)
subplot(2,1,1)
plot(tt,RX(1,1:length(tt)));
title('Received BPSK Signal')
xlabel('Time(s)');
```

```
ylabel('Amplitude');

subplot(2,1,2)
stem(CS);
title('Output of the Coherent Correlation receiver')

figure(4)
subplot(2,1,1)
stem(binary_sequence,'Linewidth',2)
title('Input binary sequence');

subplot(2,1,2)
stem(BINSEQDET,'Linewidth',2)
title('Detected binary sequence');
```

2. Amplitude Modulation and Demodulation

```
clc;
clear;
close all;

% Parameters
```

```
fm = 10;

fc = 100;

am = 1;

ac = 1;

fs = 1000;

t = 0:1/fs:1;

k = 1;


% Modulating Signal (Message Signal)

mt = am * sin(2 * pi * fm * t);


% Carrier Signal

ct = ac * sin(2 * pi * fc * t);


% AM Modulation (Corrected Formula)

mod_signal = (ac + k * mt) .* ct;


% Envelope Detection for AM Demodulation (Approximated)

demod_signal = abs(hilbert(mod_signal));


% Plotting
```

```
figure;
subplot(2,2,1);
plot(t, mt);
title('Modulating Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,2,2);
plot(t, ct);
title('Carrier Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,2,3);
plot(t, mod_signal);
title('AM Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,2,4);
plot(t, demod_signal, 'r');
```

```
hold on;
plot(t, mt, 'b--');
title('Demodulated Signal (Envelope Detection)');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Demodulated', 'Original Message');
```

3. Frequency Modulation And Demodulation

```
clc; clear; close all;
% Parameters
fm = 10;
fc = 100;
am = 1;
ac = 1;
fs = 1000;
df = 50;
t = 0:1/fs:0.5;

% Modulating Signal (Message Signal)
mt = am * sin(2 * pi * fm * t);
```

```
% Carrier Signal
ct = ac * sin(2 * pi * fc * t);

% FM Modulated Signal
mod_index = df / fm;
mod_signal = ac * cos(2 * pi * fc * t + mod_index * sin(2 * pi * fm * t));

% FM Demodulation using Hilbert Transform
analytic_signal = hilbert(mod_signal);
inst_phase = unwrap(angle(analytic_signal));
demod_signal = diff(inst_phase) * (fs / (2 * pi * mod_index));
demod_signal = [demod_signal, demod_signal(end)];

% Plotting
figure;

subplot(4,1,1);
plot(t, mt);
title('Modulating Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

```
subplot(4,1,2);
plot(t, ct);
title('Carrier Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4,1,3);
plot(t, mod_signal);
title('FM Modulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4,1,4);
plot(t, demod_signal, 'r');
hold on;
plot(t, mt, 'b--');
title('FM Demodulated Signal');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Demodulated', 'Original Message');
```

4. BER vs SNR text input

```
clc;

clear;

close all;


M = 8; % 8-PSK Modulation

bps = log2(M);


% Input text

text = 'Information and Communication Engineering';

symbols = double(text);


% Convert text to binary representation

symbolToBitMapping = de2bi(symbols, 8, 'left-msb');

totNoBits = numel(symbolToBitMapping);

inputReshapedBits = reshape(symbolToBitMapping, 1, totNoBits);


% Padding to make data length a multiple of bps

remainder = mod(totNoBits, bps);
```

```
if remainder == 0
   userPaddedData = inputReshapedBits;
else
   paddingBits = zeros(1, bps - remainder);
   userPaddedData = [inputReshapedBits paddingBits];
end

% Modulation
reshapedUserPaddedData = reshape(userPaddedData, [], bps);
bitToSymbolMapping = bi2de(reshapedUserPaddedData, 'left-msb');
modulated_symbol = pskmod(bitToSymbolMapping, M, 0); %
Removed 'gray' for MATLAB 2014

% SNR vs BER Analysis
SNR_range = 0:15;
BER = zeros(size(SNR_range));

for idx = 1:length(SNR_range)
   snr = SNR_range(idx);

   % Add noise
```

```
    noisySymbols = awgn(modulated_symbol, snr);

    % Demodulation
    demodulatedSymbol = pskdemod(noisySymbols, M, 0);

    % Convert symbols back to bits
    demodulatedSymbolToBitMapping = de2bi(demodulatedSymbol,
bps, 'left-msb');
    reshapedDemodulatedBits =
reshape(demodulatedSymbolToBitMapping.', 1, []);

    % Remove padding
    demodulatedBitsWithoutPadding =
reshapedDemodulatedBits(1:totNoBits);

    % Calculate BER
    [~, ber] = biterr(inputReshapedBits,
demodulatedBitsWithoutPadding);
    BER(idx) = ber;

    % Convert bits back to text
    if mod(length(demodulatedBitsWithoutPadding), 8) == 0
```

```
        txtBits = reshape(demodulatedBitsWithoutPadding, [], 8);
        txtBitsDecimal = bi2de(txtBits, 'left-msb');
        msg = char(txtBitsDecimal)';
    else
        msg = 'Error in text conversion';
    end
end

% Plot BER vs SNR
figure;
semilogy(SNR_range, BER, 'b-o', 'LineWidth', 2);
xlabel('SNR (dB)');
ylabel('BER');
title('SNR vs BER for 8-PSK over AWGN');
grid on;
```

5. BER vs SNR random value input

```
clc;
clear;
close all;

M = 8;
bps = log2(M);

% Generate a random binary data sequence
numBits = 10000;
randomBits = randi([0 1], 1, numBits);

% Padding to ensure data length is a multiple of bps
remainder = mod(numBits, bps);
if remainder ~= 0
    paddingBits = zeros(1, bps - remainder); % Add zero padding
    randomBits = [randomBits paddingBits];
end

% Convert bits to symbols
```

```
reshapedBits = reshape(randomBits, [], bps);
bitToSymbolMapping = bi2de(reshapedBits, 'left-msb');

% 8-PSK Modulation
modulatedSymbols = pskmod(bitToSymbolMapping, M, 0);

% SNR vs BER Analysis
SNR_range = 0:15;
BER = zeros(size(SNR_range));

for idx = 1:length(SNR_range)
    snr = SNR_range(idx);

    % Add AWGN noise
    noisySymbols = awgn(modulatedSymbols, snr);

    % Demodulation
    demodulatedSymbols = pskdemod(noisySymbols, M, 0);

    % Convert symbols back to bits
```

```
    demodulatedBitsMatrix = de2bi(demodulatedSymbols, bps, 'left-msb');
    receivedBits = reshape(demodulatedBitsMatrix.', 1, []); % Convert back to 1D array

    % Remove padding
    receivedBits = receivedBits(1:numBits);

    % Compute BER
    [~, ber] = biterr(randomBits(1:numBits), receivedBits);
    BER(idx) = ber;
end

% Plot BER vs SNR
figure;
semilogy(SNR_range, BER, 'b-o', 'LineWidth', 2);
xlabel('SNR (dB)');
ylabel('BER');
title('SNR vs BER for 8-PSK over AWGN with Random Data');
grid on;
```