Old Code:

1. Amplitude Modulation and Demodulation

```
clc;
clear;
close;
fm = 10; % Frequency of modulating signal
fc = 100; % Frequency of carrier signal
am = 1; % Amplitude of modulating signal
ac = 1; % Amplitude of carrier signal
t = 0:0.001:1; % Time vector
k = am/ac; % Modulation index
wc1 = 2*pi*fm; % Angular frequency of modulating signal
wc2 = 2*pi*fc; % Angular frequency of carrier signal
mt = am*sin(wc1*t); % Modulating signal
ct = ac*sin(wc2*t); % Carrier signal
mod_signal = (1+k*mt).*ct; % Amplitude Modulated signal
demod_signal = (ac+mt); % Demodulated signal (simplified)

figure(1)
subplot(221)
plot(t,mt)
title('Modulating Signal')
subplot(222)
plot(t,ct)
title('Carrier Signal')
subplot(223)
plot(t,mod_signal,t,ac+mt,t,-ac-mt)
title('Modulated Signal ')
subplot(224)
plot(t,mt,t,demod_signal)
title('Demodulated Signal')
```

Explanation:
Modulating Signal (mt): A sine wave with frequency fm = 10 Hz.

Carrier Signal (ct): A sine wave with frequency fc = 100 Hz.

Modulated Signal: Generated using the formula (1 + k * mt) .* ct, where k is the modulation index.

Demodulated Signal: Reconstructed by adding the modulating signal to the carrier amplitude.

Plots:

Modulating signal, carrier signal, modulated signal, and demodulated signal.

2. Frequency Modulation

```
clc;
clear;
close;
fm = 10; % Frequency of modulating signal
fc = 100; % Frequency of carrier signal
am = 1; % Amplitude of modulating signal
ac = 1; % Amplitude of carrier signal
t = 0:0.001:0.5; % Time vector
df = 50; % Frequency deviation
wc1 = 2*pi*fm; % Angular frequency of modulating signal
wc2 = 2*pi*fc; % Angular frequency of carrier signal
mt = am*sin(wc1*t); % Modulating signal
ct = ac*sin(wc2*t); % Carrier signal
mod_signal = ac*cos(wc2*t+(2*pi*df/fm)*sin(wc1*t)); % Frequency Modulated
signal

figure(1)
subplot(311)
plot(t,mt)
title('Modulating Signal')
subplot(312)
plot(t,ct)
title('Carrier Signal')
subplot(313)
plot(t,mod_signal)
title('Modulated Signal')
```

Explanation:
Modulating Signal (mt): A sine wave with frequency fm = 10 Hz.

Carrier Signal (ct): A sine wave with frequency fc = 100 Hz.

Modulated Signal: Generated using the formula ac * cos(wc2 * t + (2 * pi * df / fm) * sin(wc1 * t)), where df is the frequency deviation.

Plots:

Modulating signal, carrier signal, and frequency-modulated signal.

3. Phase Modulation

```
clc;
clear;
close;
t = 0:0.001:0.5; % Time vector
fm = 5; % Frequency of modulating signal
fc = 20; % Frequency of carrier signal
A = 5; % Amplitude of carrier signal
mt = square(2*pi*fm*t); % Modulating signal (square wave)
ct = A*sin(2*pi*fc*t); % Carrier signal
psk_signal = ct.*mt; % Phase Shift Keying (PSK) signal

figure(1)
subplot(311)
plot(t,ct)
title('Carrier Signal')
grid on
subplot(312)
plot(t,mt,'r')
title('Message Signal')
grid on
subplot(313)
plot(t,psk_signal)
title('PSK Signal')
grid on
```

Explanation:
Modulating Signal (mt): A square wave with frequency fm = 5 Hz.

Carrier Signal (ct): A sine wave with frequency fc = 20 Hz.

PSK Signal: Generated by multiplying the carrier signal with the modulating signal.

Plots:

Carrier signal, message signal, and PSK signal.

4. BPSK Modulation and Demodulation

```
clc;
clear;
close all;
binary_sequence = randi([0 1],1,10); % Random binary sequence
N = length(binary_sequence); % Length of binary sequence
T = 1; % Bit duration
fs = 1000; % Sampling frequency
fc = 5; % Carrier frequency
SNR_dB = 3; % Signal-to-Noise Ratio in dB
t = 0:1/fs:(N*T)-1/fs; % Time vector
polar_sequence = 2 * binary_sequence - 1; % Convert binary to polar form
polar_NRZ_signal = zeros(1,N*fs); % Initialize polar NRZ signal
for i = 1 : N
    polar_NRZ_signal((i-1)*fs+1:i*fs) = polar_sequence(i); % Generate polar
NRZ signal
end
carrier_signal = cos(2 * pi * fc * t); % Carrier signal
BPSK_signal = polar_NRZ_signal .* carrier_signal; % BPSK Modulated signal
received_signal = awgn(BPSK_signal,SNR_dB,'measured'); % Add AWGN noise
demodulated_signal = received_signal .*carrier_signal; % Demodulated signal
integrated_signal = zeros(1,N); % Initialize integrated signal
for i = 1 : N
    integrated_signal(i) = mean(demodulated_signal((i-1)*fs+1:i*fs)); %
Integrate over bit duration
end
detected_sequence = zeros(1,N); % Initialize detected sequence
for i = 1 : N
    if integrated_signal(i) > 0
        detected_sequence(i) = 1; % Detect binary 1
    else
        detected_sequence(i) = 0; % Detect binary 0
    end
end
bit_error = sum(binary_sequence~=detected_sequence); % Calculate bit errors
BER = bit_error/N; % Calculate Bit Error Rate
disp('Transmitted Binary Sequence : ');
disp(binary_sequence);
disp('Detected Binary Sequence : ');
disp(detected_sequence);
disp(['Number of Bit Error : ',num2str(bit_error)]);
```

```
disp(['Bit Error Rate : ',num2str(BER)]);

figure(1);
subplot(411)
stairs(0:N-1,binary_sequence,'r','Linewidth',1.5)
title('Polar NRZ Input Binary Sequence');
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.5 1.5])
grid on
subplot(412)
plot(t,BPSK_signal,'c','Linewidth',1.5)
title('BPSK Modulated Signal');
xlabel('Time (s)')
ylabel('Amplitude')
grid on
subplot(413)
plot(t,received_signal,'g','Linewidth',1.5)
title('Received Signal In AWGN Channel');
xlabel('Time (s)')
ylabel('Amplitude')
grid on
subplot(414)
t_integrated = 0:T:(N-1)*T;
plot(t_integrated,integrated_signal,'b','Linewidth',1.5)
title('Output of the Coherent Correlation Receiver');
xlabel('Time (s)')
ylabel('Amplitude')
grid on

figure(2);
subplot(411)
plot(t,carrier_signal)
title('Carrier Signal');
xlabel('Time (s)')
ylabel('Amplitude');
grid on
subplot(412)
stairs(0:N-1,binary_sequence,'g','Linewidth',1.5)
title('Input Binary Sequence');
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.5 1.5])
```

```
grid on
subplot(413)
stairs(0:N-1,binary_sequence,'b','Linewidth',1.5)
title('Detected Binary Sequence');
xlabel('Time (s)')
ylabel('Amplitude')
ylim([-0.5 1.5])
grid on
subplot(414)
plot(polar_sequence,zeros(1,N),'bo','MarkerFaceColor','r')
title('Signal Space Diagram for BPSK');
xlabel('In Phase (I)')
ylabel('Quadrature (Q)')
xlim([-1.5 1.5])
ylim([-0.5 0.5])
grid on
```

Explanation:
Binary Sequence: A random binary sequence of length 10.

Polar NRZ Signal: Converts binary sequence to polar form (1 → 1, 0 → -1).

BPSK Signal: Generated by multiplying the polar NRZ signal with the carrier signal.

AWGN Channel: Adds noise to the BPSK signal.

Demodulation: Coherent detection using the carrier signal.

Bit Error Rate (BER): Calculated by comparing transmitted and detected sequences.

Plots:

Input binary sequence, BPSK modulated signal, received signal, and demodulated signal.

5. BER vs SNR

```
clc;
clear;
close all;
M = 8; % Modulation order
bps = log2(M); % Bits per symbol
txt = 'Information and Communication Engineering'; % Input text
```

```
symbols = double(txt); % Convert text to ASCII values
symbolToBitMapping = de2bi(symbols,8,'left-msb'); % Convert ASCII to binary
totalNumberOfBits = numel(symbolToBitMapping); % Total number of bits
inputReshapedBits = reshape(symbolToBitMapping,1,totalNumberOfBits); %
Reshape to 1D array
remainder = rem(totalNumberOfBits,bps); % Check for padding
if (remainder) == 0
    userPaddedData = inputReshapedBits; % No padding needed
else
    paddingBits = zeros(1,bps - remainder); % Add padding
    userPaddedData = [inputReshapedBits paddingBits];
end
reshapedUserData = reshape(userPaddedData,numel(userPaddedData)/bps,bps); %
Reshape for symbol mapping
bitToSymbolMapping = bi2de(reshapedUserData,'left-msb'); % Convert bits to
symbols
modulatedSymbol = pskmod(bitToSymbolMapping,M); % Modulate using PSK
SNR = []; % Initialize SNR array
BER = []; % Initialize BER array
for snr = 0:15
    SNR = [SNR snr]; % Store SNR value
    noisySymbols = awgn(modulatedSymbol,snr,'measured'); % Add AWGN noise
    demodulatedSymbol = pskdemod(noisySymbols,M); % Demodulate
    demodulatedSymbolToBitMapping = de2bi(demodulatedSymbol,'left-msb'); %
Convert symbols to bits
    reshapedDemodulatedBits =
reshape(demodulatedSymbolToBitMapping,1,numel(demodulatedSymbolToBitMapping
)); % Reshape to 1D array
    demodulatedBitsWithoutPadding =
reshapedDemodulatedBits(1:totalNumberOfBits); % Remove padding
    [noe, ber] = biterr(inputReshapedBits,demodulatedBitsWithoutPadding); %
Calculate BER
    BER = [BER ber]; % Store BER value
end
figure(1)
semilogy(SNR,BER,'--'); % Plot BER vs SNR
xlabel('SNR');
ylabel('BER');
title('SNR vs BER');
```

Explanation:

Input Text: Converted to binary and mapped to symbols.

Modulation: 8-PSK modulation is used.

AWGN Channel: Adds noise to the modulated signal.

Demodulation: PSK demodulation is performed.

BER Calculation: Bit Error Rate is calculated for different SNR values.

Plot: BER vs SNR is plotted on a logarithmic scale.

Cleaned Code:

1. Amplitude Modulation and Demodulation (Cleaned)

```
clc; clear; close;
fm = 10; fc = 100; am = 1; ac = 1;
t = 0:0.001:1;
k = am/ac;
mt = am*sin(2*pi*fm*t);
ct = ac*sin(2*pi*fc*t);
mod_signal = (1+k*mt).*ct;
demod_signal = (ac+mt);

figure(1)
subplot(221); plot(t,mt); title('Modulating Signal');
subplot(222); plot(t,ct); title('Carrier Signal');
subplot(223); plot(t,mod_signal); title('Modulated Signal');
subplot(224); plot(t,mt,t,demod_signal); title('Demodulated Signal');
```

2. Frequency Modulation (Cleaned)

```
clc; clear; close;
fm = 10; fc = 100; am = 1; ac = 1; df = 50;
t = 0:0.001:0.5;
mt = am*sin(2*pi*fm*t);
ct = ac*sin(2*pi*fc*t);
mod_signal = ac*cos(2*pi*fc*t + (2*pi*df/fm)*sin(2*pi*fm*t));

figure(1)
subplot(311); plot(t,mt); title('Modulating Signal');
subplot(312); plot(t,ct); title('Carrier Signal');
subplot(313); plot(t,mod_signal); title('Modulated Signal');
```

3. Phase Modulation (Cleaned)

```
clc; clear; close;
t = 0:0.001:0.5; fm = 5; fc = 20; A = 5;
mt = square(2*pi*fm*t);
ct = A*sin(2*pi*fc*t);
```

```
psk_signal = ct.*mt;

figure(1)
subplot(311); plot(t,ct); title('Carrier Signal'); grid on;
subplot(312); plot(t,mt,'r'); title('Message Signal'); grid on;
subplot(313); plot(t,psk_signal); title('PSK Signal'); grid on;
```

4. BPSK Modulation and Demodulation (Cleaned)

```
clc; clear; close all;
binary_sequence = randi([0 1],1,10);
N = length(binary_sequence); T = 1; fs = 1000; fc = 5; SNR_dB = 3;
t = 0:1/fs:(N*T)-1/fs;
polar_sequence = 2 * binary_sequence - 1;
polar_NRZ_signal = repelem(polar_sequence, fs);
carrier_signal = cos(2 * pi * fc * t);
BPSK_signal = polar_NRZ_signal .* carrier_signal;
received_signal = awgn(BPSK_signal, SNR_dB, 'measured');
demodulated_signal = received_signal .* carrier_signal;
integrated_signal = reshape(mean(reshape(demodulated_signal, fs, N)), 1,
N);
detected_sequence = integrated_signal > 0;
bit_error = sum(binary_sequence ~= detected_sequence);
BER = bit_error/N;

disp('Transmitted Binary Sequence:'); disp(binary_sequence);
disp('Detected Binary Sequence:'); disp(detected_sequence);
disp(['Bit Errors: ', num2str(bit_error), ', BER: ', num2str(BER)]);

figure(1)
subplot(411); stairs(0:N-1, binary_sequence, 'r', 'Linewidth', 1.5);
title('Polar NRZ Input Binary Sequence'); ylim([-0.5 1.5]); grid on;
subplot(412); plot(t, BPSK_signal, 'c', 'Linewidth', 1.5);
title('BPSK Modulated Signal'); grid on;
subplot(413); plot(t, received_signal, 'g', 'Linewidth', 1.5);
title('Received Signal in AWGN Channel'); grid on;
subplot(414); plot(0:T:(N-1)*T, integrated_signal, 'b', 'Linewidth', 1.5);
title('Output of Coherent Correlation Receiver'); grid on;
```

5. BER vs SNR (Cleaned)

```
clc; clear; close all;
M = 8; bps = log2(M);
txt = 'Information and Communication Engineering';
symbols = double(txt);
symbolToBitMapping = de2bi(symbols, 8, 'left-msb');
totalNumberOfBits = numel(symbolToBitMapping);
inputReshapedBits = reshape(symbolToBitMapping, 1, totalNumberOfBits);
paddingBits = zeros(1, mod(-totalNumberOfBits, bps));
userPaddedData = [inputReshapedBits paddingBits];
reshapedUserData = reshape(userPaddedData, [], bps);
bitToSymbolMapping = bi2de(reshapedUserData, 'left-msb');
modulatedSymbol = pskmod(bitToSymbolMapping, M);
SNR = 0:15; BER = zeros(1, length(SNR));

for snr = SNR
    noisySymbols = awgn(modulatedSymbol, snr, 'measured');
    demodulatedSymbol = pskdemod(noisySymbols, M);
    demodulatedBits = de2bi(demodulatedSymbol, 'left-msb');
    reshapedDemodulatedBits = reshape(demodulatedBits, 1, []);
    demodulatedBitsWithoutPadding =
reshapedDemodulatedBits(1:totalNumberOfBits);
    [~, BER(snr+1)] = biterr(inputReshapedBits,
demodulatedBitsWithoutPadding);
end

figure(1)
semilogy(SNR, BER, '--');
xlabel('SNR'); ylabel('BER'); title('SNR vs BER');
```

New New Code:

1. BPSK Modulation and Demodulation
This section demonstrates Binary Phase Shift Keying (BPSK) modulation and demodulation, including noise addition and coherent detection.

matlab
Copy

```matlab
clc;
clear all;
close all;

% Binary sequence
binary_sequence = [1 0 1 1 0 0 1 0 0 1];
number_bits = length(binary_sequence);

% Parameters
Eb = 2; % Energy per bit
Tb = 1; % Bit duration
Ac = 1; % Carrier amplitude
nc = 4; % Number of carrier cycles per bit
fc = nc / Tb; % Carrier frequency
tf = 99; % Time factor
t = 0:1/tf:1; % Time vector for one bit
tn = 0:1/(tf+1):number_bits; % Time vector for all bits
tt = tn(1, 2:end); % Adjusted time vector

% 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 add AWGN noise
SNR = 1; % Signal-to-Noise Ratio
Ps = mean(abs(TX).^2); % Signal power
Pn = Ps / (10^(SNR / 10)); % Noise power
noise = sqrt(Pn) * randn(1, length(TX)); % Generate noise
RX = TX + noise; % Add noise to the signal

% Coherent demodulation
LO = sqrt(2 / Tb) * cos(2 * pi * fc * t); % Local oscillator
BINSEQDET = [];
CS = [];
for n = 1:number_bits
    temp = RX((n - 1) * (tf + 1) + 1:n * (tf + 1));
    S = sum(temp .* LO); % Correlation
    CS = [CS S];
    if S > 0
        BINSEQDET = [BINSEQDET 1];
    else
        BINSEQDET = [BINSEQDET 0];
    end
end

% Calculate bit errors
bit_errors = sum(abs(BINSEQDET - binary_sequence));
disp(['Total Bit Errors: ', num2str(bit_errors)]);

% Plotting
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

This section demonstrates Amplitude Modulation (AM) and demodulation using envelope detection.

matlab

Copy

```
clc;
clear;
close all;

% Parameters
fm = 10; % Message frequency
fc = 100; % Carrier frequency
am = 1; % Message amplitude
ac = 1; % Carrier amplitude
```

```
fs = 1000; % Sampling frequency
t = 0:1/fs:1; % Time vector
k = 1; % Modulation index

% Modulating signal (message signal)
mt = am * sin(2 * pi * fm * t);

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

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

% Envelope detection for AM demodulation
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

This section demonstrates Frequency Modulation (FM) and demodulation using the Hilbert transform.

matlab
Copy

```matlab
clc;
clear;
close all;

% Parameters
fm = 10; % Message frequency
fc = 100; % Carrier frequency
am = 1; % Message amplitude
ac = 1; % Carrier amplitude
fs = 1000; % Sampling frequency
df = 50; % Frequency deviation
t = 0:1/fs:0.5; % Time vector

% Modulating signal (message signal)
mt = am * sin(2 * pi * fm * t);

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

% FM modulation
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)

This section calculates the Bit Error Rate (BER) for 8-PSK modulation using a text input.

matlab

Copy

```
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
symbolToBitMapping = de2bi(symbols, 8, 'left-msb');
totNoBits = numel(symbolToBitMapping);
inputReshapedBits = reshape(symbolToBitMapping, 1, totNoBits);

% Padding
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);

% SNR vs BER analysis
SNR_range = 0:15;
BER = zeros(size(SNR_range));
for idx = 1:length(SNR_range)
    snr = SNR_range(idx);
    noisySymbols = awgn(modulated_symbol, snr); % Add noise
    demodulatedSymbol = pskdemod(noisySymbols, M, 0); % Demodulation
    demodulatedSymbolToBitMapping = de2bi(demodulatedSymbol, bps,
'left-msb');
    reshapedDemodulatedBits = reshape(demodulatedSymbolToBitMapping.', 1,
[]);
    demodulatedBitsWithoutPadding = reshapedDemodulatedBits(1:totNoBits);
    [~, ber] = biterr(inputReshapedBits, demodulatedBitsWithoutPadding); %
Calculate BER
    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');
grid on;
```

5. BER vs SNR (Random Input)

This section calculates the Bit Error Rate (BER) for 8-PSK modulation using random binary data.

matlab

Copy

```
clc;
clear;
close all;
```

```
M = 8; % 8-PSK modulation
bps = log2(M);

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

% Padding
remainder = mod(numBits, bps);
if remainder ~= 0
    paddingBits = zeros(1, bps - remainder);
    randomBits = [randomBits paddingBits];
end

% Convert bits to symbols
reshapedBits = reshape(randomBits, [], bps);
bitToSymbolMapping = bi2de(reshapedBits, 'left-msb');
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);
    noisySymbols = awgn(modulatedSymbols, snr); % Add noise
    demodulatedSymbols = pskdemod(noisySymbols, M, 0); % Demodulation
    demodulatedBitsMatrix = de2bi(demodulatedSymbols, bps, 'left-msb');
    receivedBits = reshape(demodulatedBitsMatrix.', 1, []);
    receivedBits = receivedBits(1:numBits); % Remove padding
    [~, ber] = biterr(randomBits(1:numBits), receivedBits); % Calculate BER
    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;
```

Summary

The code demonstrates BPSK, AM, and FM modulation and demodulation.

It also calculates BER vs SNR for 8-PSK modulation using both text and random binary inputs.

Each section is self-contained and can be run independently.