1.

```
clear all;
close all;
% Generate random binary message
msg = round(rand(1, 1000));
k = length(msg);
encoded = zeros(1, 2*k);  % Rate 1/2 encoded output
% Convolutional encoder with constraint length 3, G=[6 7] in octal
reg = [0 0];  % 2-bit memory
for i = 1:k
   input_bit = msg(i);
   reg = [input_bit reg];
   reg = reg(1:3);
   g1 = xor(reg(1), reg(2));                        % G1 = 6 = 110
   g2 = xor(xor(reg(1), reg(2)), reg(3));           % G2 = 7 = 111
   encoded(2*i-1) = g1;
   encoded(2*i) = g2;
end
% Convert binary to bipolar
encoded(encoded == 0) = -1;
length_user = length(encoded);
% Parameters
fc = 5000;
eb = 0.5;
bitrate = 1000;
tb = 1 / bitrate;
chiprate = 10000;
tc = 1 / chiprate;
samples_per_bit = round(tb / tc);
t = tc:tc:(tb * length_user);
% Baseband signal
basebandsig = repelem(encoded, samples_per_bit);
% Plot baseband signal
figure(1)
stairs(t(1:800), basebandsig(1:800), 'LineWidth', 1.5)
xlabel('Time (sec)')
ylabel('Binary value')
set(gca, 'ytick', [-1 1])
title('A segment of original binary sequence for a single user')
grid on
% BPSK Modulation
bpskmod = [];
for i = 1:length_user
   for j = tc:tc:tb
       bpskmod = [bpskmod sqrt(2*eb)*encoded(i)*cos(2*pi*fc*j)];
   end
end
% Frequency spectrum
number = length(t);
```

```
spectrum = abs(fft(bpskmod));
sampling_frequency = 2*fc;
sampling_interval = 1 / sampling_frequency;
for i = 1:number
   frequency(i) = (1.0 / (number * sampling_interval)) * i;
end
figure(2)
plot(frequency, spectrum)
title('Frequency Domain analysis of BPSK modulated signal for a single user')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
grid on
% PN sequence generation (length = 10 per bit)
seed = [1 -1 1 -1];
pn = [];
for i = 1:length_user
   for j = 1:10
       pn = [pn seed(4)];
       temp = xor(seed(4) == 1, seed(3) == 1);  % XOR logic
       temp = 1 - 2 * temp; % Convert logical XOR to bipolar (+1/-1)
       seed = [temp seed(1:3)];
   end
end
% Upsample PN sequence to match signal length
pnupsampled = repelem(pn, round(tc * chiprate));
% Transmit: spread BPSK signal
sigtx = bpskmod .* pnupsampled(1:length(bpskmod));
% Plot transmitted signal
figure(3)
plot(t(1:200), sigtx(1:200))
title('A segment of Transmitted DS-CDMA signal')
xlabel('Time (sec)')
ylabel('Amplitude')
grid on
% BER Simulation
snr_in_dBs = 0:1:10;
for m = 1:length(snr_in_dBs)
   ber(m) = 0;

   % Add AWGN
   sig_power = mean(sigtx.^2);                     % Signal power
   snr_linear = 10^(snr_in_dBs(m) / 10);           % SNR in linear scale
   noise_power = sig_power / snr_linear;           % Noise power
   noise = sqrt(noise_power) * randn(1, length(sigtx));  % Gaussian noise
   composite_signal = sigtx + noise;               % Add noise

   % De-spreading
   rx = composite_signal .* pnupsampled(1:length(composite_signal));
```

```
    % Demodulate carrier
    demodcar = [];
    for i = 1:length_user
        for j = tc:tc:tb
            demodcar = [demodcar sqrt(2*eb)*cos(2*pi*fc*j)];
        end
    end
    bpskdemod = rx .* demodcar;

    % Integrate and dump over each bit (10 samples per bit)
    len_dmod = length(bpskdemod);
    sum_bits = zeros(1, len_dmod / samples_per_bit);
    for i = 1:length(sum_bits)
        range = (i-1)*samples_per_bit + 1 : i*samples_per_bit;
        sum_bits(i) = sum(bpskdemod(range));
    end

    % Hard decision decoding
    rxbits = double(sum_bits > 0);

    % Simple Viterbi-like decoder (hard decision) -- just keeps first half
    % since we can't use vitdec without toolbox
    decoded = rxbits(1:2:end);  % crude way for comparison
    % Clip to match length
    min_len = min(length(decoded), length(msg));
    errors = sum(decoded(1:min_len) ~= msg(1:min_len));  % Bit errors
    rat = errors / min_len;                              % BER
    ber(m) = rat;
end
% BER Plot
figure(4)
plot(snr_in_dBs, ber, 'o-');
xlabel('Signal to Noise Ratio (dB)')
ylabel('BER')
legend('BER simulation for a single user')
title('Coded BER simulation under AWGN channel (Toolbox-Free)')
grid on
```

2.

```
clear all;
close all;
% Message bits
msg = round(rand(1,1000));
% Manual convolutional encoder (constraint length 3, generators [6 7])
msg = [msg 0 0]; % Padding for termination
user = [];
reg = [0 0];
for i = 1:length(msg)
    reg = [msg(i) reg(1:2)];
```

```
   g1 = xor(reg(1), reg(3));
   g1 = xor(g1, reg(2)); % corresponds to octal 7
   g2 = xor(reg(1), reg(2)); % corresponds to octal 6
   user = [user g1 g2];
end
% Convert to bipolar
user(user==0) = -1;
length_user = length(user);
% Signal parameters
fc = 5000;
eb = 0.5;
bitrate = 1000;
tb = 1/bitrate;
chiprate = 10000;
tc = 1/chiprate;
t = tc:tc:tb*length_user;
% Baseband signal
basebandsig = repelem(user, 10);
figure(1)
stairs(t(1:800), basebandsig(1:800))
xlabel('Time(sec)')
ylabel('Binary value')
set(gca,'ytick',[ -1  1 ])
title('A segment of original binary sequence for a single user')
% BPSK modulation
bpskmod = [];
for i = 1:length_user
   for j = tc:tc:tb
       bpskmod = [bpskmod sqrt(2*eb)*user(i)*cos(2*pi*fc*j)];
   end
end
% Frequency spectrum
number = length(t);
spectrum = abs(fft(bpskmod));
sampling_frequency = 2*fc;
sampling_interval = 1.0/sampling_frequency;
for i = 1:number
   frequency(i) = (1.0/(number*sampling_interval))*i;
end
figure(2)
plot(frequency, spectrum)
title('Frequency Domain analysis of BPSK modulated signal for a single user')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
grid on
% PN sequence (manual)
seed = [1 -1 1 -1];
pn = [];
for i = 1:length_user
```

```
    for j = 1:10
        pn = [pn seed(4)];
        if seed(4) == seed(3)
            temp = -1;
        else
            temp = 1;
        end
        seed(4) = seed(3);
        seed(3) = seed(2);
        seed(2) = seed(1);
        seed(1) = temp;
    end
end
% Upsample PN
pnupsampled = repelem(pn, 1);
% Transmit signal
sigtx = bpskmod .* pnupsampled;
figure(3)
plot(t(1:200), sigtx(1:200))
title('A segment of Transmitted DS CDMA signal')
xlabel('Time(sec)')
ylabel('Amplitude')
grid on
% Simulate Rayleigh fading manually (no toolbox)
fading = (1/sqrt(2)) * (randn(size(sigtx)) + 1i*randn(size(sigtx)));
fadedsig = real(fading .* sigtx);
% BER Simulation
snr_in_dBs = 0:1.0:10;
ber = zeros(size(snr_in_dBs));
for m = 1:length(snr_in_dBs)
    % Manual AWGN addition
    sig_power = mean(fadedsig.^2);
    snr_linear = 10^(snr_in_dBs(m)/10);
    noise_power = sig_power / snr_linear;
    noise = sqrt(noise_power) * randn(1, length(fadedsig));
    composite_signal = fadedsig + noise;
    % Receiver
    rx = composite_signal .* pnupsampled;
    demodcar = [];
    for i = 1:length_user
        for j = tc:tc:tb
            demodcar = [demodcar sqrt(2*eb)*cos(2*pi*fc*j)];
        end
    end
    bpskdemod = rx .* demodcar;
    % Integrate and dump
    sum_vals = zeros(1, length_user);
    for i = 1:length_user
        idx_start = (i-1)*10 + 1;
```

```
        idx_end = i*10;
        sum_vals(i) = sum(bpskdemod(idx_start:idx_end));
    end
    rxbits = double(sum_vals > 0);
    % Manual Viterbi decoding (hard decision, trace-back = 3)
    % For simplicity, we use brute-force decoding here (slow for long msgs)
    decoded = [];
    for i = 1:2:length(rxbits)-1
        bit1 = rxbits(i);
        bit2 = rxbits(i+1);
        if bit1 == 1 && bit2 == 1
            decoded = [decoded 1];
        elseif bit1 == 0 && bit2 == 0
            decoded = [decoded 0];
        else
            % Assume previous bit (not accurate, placeholder)
            decoded = [decoded 0];
        end
    end
    min_len = min(length(decoded), length(msg));
    errors = sum(decoded(1:min_len) ~= msg(1:min_len));
    ber(m) = errors / min_len;
end
figure(4)
plot(snr_in_dBs, ber, 'o-');
xlabel('Signal to noise ratio (dB)')
ylabel('BER')
legend('BER simulation for a single user')
title('Coded BER under AWGN and Rayleigh fading (No Toolbox)')
grid on
```

3.

```
clear all;
close all;
%% Generate random message
msg = round(rand(1,1000));
%% Manual Convolutional Encoder (rate 1/2, constraint length 3, generators [6
7])
g1 = [1 1 0]; % 6 in binary is 110
g2 = [1 1 1]; % 7 in binary is 111
k = length(g1);
msg_padded = [zeros(1, k-1), msg];
user = [];
for i = 1:length(msg)
    reg = msg_padded(i:i+k-1);
    u1 = mod(sum(g1 .* reg), 2);
    u2 = mod(sum(g2 .* reg), 2);
    user = [user u1 u2];
end
```

```
%% Map 0 -> -1
user(user==0) = -1;
%% BPSK Parameters
fc = 5000;
eb = 0.5;
bitrate = 1000;
tb = 1 / bitrate;
chiprate = 10000;
tc = 1 / chiprate;
%% Time vector
t = tc:tc:tb*length(user);
%% Baseband signal
basebandsig = repelem(user, tb/tc);
%% Plot binary signal
figure(1)
stairs(t(1:800), basebandsig(1:800))
xlabel('Time(sec)')
ylabel('Binary value')
set(gca, 'ytick', [-1 1])
title('A segment of original binary sequence for a single user')
%% BPSK modulation
bpskmod = sqrt(2*eb) * basebandsig .* cos(2*pi*fc*t);
%% Spectrum
spectrum = abs(fft(bpskmod));
sampling_frequency = 2 * fc;
frequency = (1:length(t)) * (sampling_frequency / length(t));
figure(2)
plot(frequency, spectrum)
title('Frequency Domain analysis of BPSK modulated signal for a single user')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
grid on
%% PN Sequence generation
seed = [1 -1 1 -1];
pn = [];
for i = 1:length(user)
   for j = 1:10
       pn = [pn seed(4)];
       temp = xor(seed(4)==1, seed(3)==1) * 2 - 1;
       seed = [temp seed(1:3)];
   end
end
pnupsampled = repelem(pn, tb/tc);
%% Transmitted signal
samples_per_bit = round(tb / tc);  % Number of samples per bit (usually 10)
pnupsampled = repelem(pn, samples_per_bit);  % Repeat each chip accordingly
pnupsampled = pnupsampled(1:length(bpskmod));  % Trim if needed
sigtx = bpskmod .* pnupsampled;  % Now should work fine
figure(3)
```

```
plot(t(1:200), sigtx(1:200))
title('A segment of Transmitted  DS CDMA signal')
xlabel('Time(sec)')
ylabel('Amplitude')
grid on
%% Simulate Rician Fading (simple manual model)
K = 10; % Rician K-factor (power ratio)
LOS = sqrt(K / (K + 1));
NLOS = sqrt(1 / (2 * (K + 1))) * (randn(1, length(t)) + 1j*randn(1,
length(t)));
h = LOS + NLOS;
fadedsig = abs(h) .* sigtx;
%% Add noise manually and calculate BER
snr_in_dBs = 0:1:10;
ber = zeros(1,length(snr_in_dBs));
for m = 1:length(snr_in_dBs)
   snr_linear = 10^(snr_in_dBs(m)/10);
   noise_power = var(fadedsig) / snr_linear;
   noise = sqrt(noise_power) * randn(1, length(fadedsig));
   composite_signal = fadedsig + noise;
   %% Receiver
   rx = composite_signal .* pnupsampled;
   demodcar = sqrt(2*eb) * cos(2*pi*fc*t);
   bpskdemod = rx .* demodcar;
   %% Integrate and dump (10 samples per bit)
   samples_per_bit = round(tb/tc);
   len_bits = length(user);
   rxbits = zeros(1, len_bits);
   for i = 1:len_bits
       start_idx = (i-1)*samples_per_bit + 1;
       end_idx = i*samples_per_bit;
       if sum(bpskdemod(start_idx:end_idx)) > 0
           rxbits(i) = 1;
       else
           rxbits(i) = 0;
       end
   end
   %% Manual Viterbi Decoder (Hard Decision, 2-state)
   % Only for constraint length 3, (2 memory bits) [6 7]
   decoded = [];
   state = 0;
   for i = 1:2:length(rxbits)-1
       bit_pair = rxbits(i:i+1);
       if isequal(bit_pair, [0 0])
           input = 0; % From state 0
       elseif isequal(bit_pair, [1 1])
           input = 0; % From state 2
       elseif isequal(bit_pair, [1 0])
           input = 1; % From state 1
```

```
        else
            input = 1; % From state 3
        end
        decoded = [decoded input];
    end
    %% Compare
    min_len = min(length(decoded), length(msg));
    errors = sum(decoded(1:min_len) ~= msg(1:min_len));
    ber(m) = errors / min_len;
end
%% Plot BER
figure(4)
plot(snr_in_dBs, ber, 'o-');
xlabel('Signal to Noise Ratio (dB)');
ylabel('Bit Error Rate (BER)');
legend('BER simulation for a single user');
title('Coded BER under AWGN + Rician fading (manual)');
grid on
```