```
clear all;
close all;
load ch_coeff.mat;

n = 10^5;
K = length(ch_coeff);
L = length(ch_coeff);

eb_n0_db = -12:2:14;
snr_db = -12:2:14;

tx_data_bp = rand(1,n);
tx_data_bpsk = 2 * (tx_data_bp > 0.5) - 1;
tx_data_qpr = rand(1,n/2);
tx_data_qpi = rand(1,n/2);
tx_data_qpsk = (2*(tx_data_qpr>0.5)-1) + j*(2*(tx_data_qpi>0.5)-1);

eb_n0_lin = 10.^(eb_n0_db/10);
snr_lin = 10.^(snr_db/10);
codes = [31, 63, 127];

bpsk1 = zeros(length(codes), length(eb_n0_db));
qpsk1 = zeros(length(codes), length(eb_n0_db));
bpsk_spread1 = zeros(length(codes), length(eb_n0_db));

bpsk2 = zeros(length(codes), length(eb_n0_db));
qpsk2 = zeros(length(codes), length(eb_n0_db));
bpsk_spread2 = zeros(length(codes), length(eb_n0_db));

% Iterating over all code lengths, to find the optimal code sequence
for k = 1:length(codes)
  code = gold(k);
  spread_data_bpsk = kron(tx_data_bpsk, code');
  spread_data_qpsk = kron(tx_data_qpsk, code');
  code_len = codes(k);

  for i = 1:length(eb_n0_db)
    clear decoded_sig1
    clear decoded_sig2
    % Adding noise
    fprintf('Epoch %d \n', i);
    noise_bpsk = randn(1, n)/(sqrt(2*eb_n0_lin(i)));
    noise_bpsk2 = randn(1,n)/sqrt(2*snr_lin(i));
    noise_qpsk = (randn(1, n/2)+j*randn(1, n/2))/sqrt(2*(2*eb_n0_lin(i)));
    noise_qpsk2 = (randn(1,n/2)+j*randn(1,n/2))/sqrt(2*2*0.5*snr_lin(i));

    % Added noise for spreading
    noise_bpsk_spread = randn(1,code_len*n)/(sqrt(2*eb_n0_lin(i)/code_len));
    noise_bpsk_spread2 = randn(1, code_len*n)/sqrt(2*snr_lin(i));
    noise_qpsk_spread = (randn(1,code_len*n/2) + j*randn(1,code_len*n/2))/(sqrt(2*2*
eb_n0_lin(i))/code_len);
    noise_qpsk_spread2 = (randn(1,n/2*code_len)+j*randn(1,n/2*code_len))/sqrt(2*2*0.
5*snr_lin(i));
    sigma = var(noise_qpsk_spread2);

    % Transmitted Data stream, unspread, power normalized for QPSK
    rx_bpsk = tx_data_bpsk + noise_bpsk;
    rx_bpsk2 = tx_data_bpsk + noise_bpsk2;
    rx_qpsk = tx_data_qpsk/sqrt(2) + noise_qpsk;
    rx_qpsk2 = tx_data_qpsk/sqrt(2) + noise_qpsk2;

    % Transmitted Data stream, spread
    rx_bpsk_spread = spread_data_bpsk + noise_bpsk_spread;
    rx_bpsk_spread2 = spread_data_bpsk + noise_bpsk_spread2;
    rx_qpsk_spread = spread_data_qpsk/sqrt(2) + noise_qpsk_spread2;

    % Applying the channel and equalization followed by synchronization, for QPSK
    rx_qpsk_spread = filter(ch_coeff, 1, rx_qpsk_spread);
    %[rx_qpsk_spread, a] = MMSE_eq(rx_qpsk_spread, ch_coeff, sigma);

    % Triggered to see effect of channel and equalization on simple unspread QPSK [O
ptional]
    %rx_qpsk2 = filter(ch_coeff, 1, rx_qpsk2);
```

```
    %[rx_qpsk2, a] = MMSE_eq(rx_qpsk2, ch_coeff, sigma);

    rx_bpsk = 2 * (rx_bpsk > 0) - 1;
    rx_bpsk2 = 2 * (rx_bpsk2 > 0) - 1;

    % Dealing with I, Q components
    rx_qpsk_i = 2 * (real(rx_qpsk) > 0) - 1;
    rx_qpsk_q = 2 * (imag(rx_qpsk) > 0) - 1;
    rx_qpsk = rx_qpsk_i + j*rx_qpsk_q;
    rx_qpsk2_i = 2 * (real(rx_qpsk2) > 0) - 1;
    rx_qpsk2_q = 2 * (imag(rx_qpsk2) > 0) - 1;
    rx_qpsk2 = rx_qpsk2_i + j*rx_qpsk2_q;

    % BPSK Spreading, 1
    temp_sig1 = rx_bpsk_spread';
    temp_sig2 = reshape(temp_sig1, code_len, n);
    temp_sig3 = kron(ones(n,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig = (sum(temp_sig4'))/code_len;
    decoded_sig1 = 2 * (despread_sig>0) - 1;

    % BPSK Spreading, 2
    temp_sig1 = rx_bpsk_spread2';
    temp_sig2 = reshape(temp_sig1, code_len, n);
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig = (sum(temp_sig4'))/code_len;
    decoded_sig2 = 2 * (despread_sig>0) - 1;

    rx_qpsk_spread_i = real(rx_qpsk_spread);
    rx_qpsk_spread_q = imag(rx_qpsk_spread);

    %channel i despreading
    temp_sig1 = rx_qpsk_spread_i';
    temp_sig2 = reshape(temp_sig1, code_len, n/2);
    temp_sig3 = kron(ones(n/2,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig_qpsk_i = (sum(temp_sig4'))/code_len;
    decoded_sig1_qpsk_i = 2 * (despread_sig_qpsk_i>0) - 1;

    %channel q despreading
    temp_sig1 = rx_qpsk_spread_q';
    temp_sig2 = reshape(temp_sig1, code_len, n/2);
    temp_sig3 = kron(ones(n/2,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig_qpsk_q = (sum(temp_sig4'))/code_len;
    decoded_sig1_qpsk_q = 2 * (despread_sig_qpsk_q>0) - 1;

    %merging decoded_sig1_i&q into 1 matrix
    decoded_sig1_qpsk = decoded_sig1_qpsk_i + j*decoded_sig1_qpsk_q;

    % Decision Time!
    bpsk1(k,i) = sum(rx_bpsk ~= tx_data_bpsk)/n;
    bpsk2(k,i) = sum(rx_bpsk2 ~= tx_data_bpsk)/n;
    bpsk_spread1(k,i) = sum(decoded_sig1 ~= tx_data_bpsk)/n;
    bpsk_spread2(k,i) = sum(decoded_sig2 ~= tx_data_bpsk)/n;

    qpsk1(k,i) = 0.5*(sum(rx_qpsk ~= tx_data_qpsk)*2/n);
    qpsk2(k,i) = 0.5*(sum(rx_qpsk2 ~= tx_data_qpsk)*2/n);
    qpsk_spread1(k,i) = 0.5*sum(decoded_sig1_qpsk ~= tx_data_qpsk)*2/n;

end
end


% For spread spectrum, we use only Ec : chip energy or we use SNR.
figure;
semilogy(snr_db,bpsk2(1,:),'b.-', snr_db,qpsk2(1,:), 'mx-', eb_n0_db,bpsk_spread2(1,
:), 'gx-', snr_db, qpsk_spread1(1,:), 'rx-');
grid on;
legend('BPSK', 'QPSK', 'BPSKspread', 'QPSKSpread');
axis([-12 14 1e-6 1]);

xlabel('SNR, dB');
```

```
ylabel('Bit Error Rate');
title('Bit error vs SNR for BPSK and QPSK');


% Selection of Optimal Code Length from qpsk spreading
[ii, jj] = min(sum(qpsk_spread1, 2));
codes(jj)

% Plot to select Optimal code
figure;
semilogy(snr_db,qpsk_spread1(1,:),'b.-', snr_db,qpsk_spread1(2,:), 'mx-', eb_n0_db,q
psk_spread1(3, :), 'gx-');
grid on;
legend('len: 31', 'len: 63', 'len: 127');
axis([-12 14 1e-6 1]);
xlabel('SNR, dB');
ylabel('Bit Error Rate');
title('BER vs SNR for varying code lengths');
```