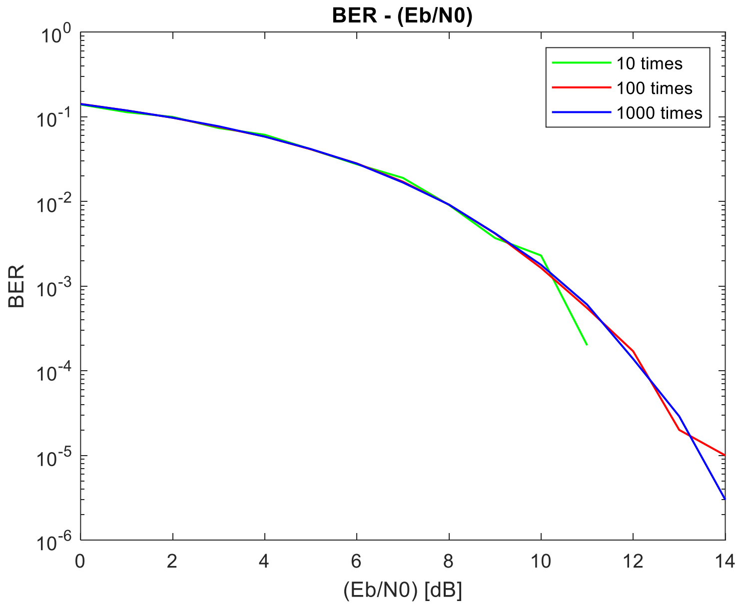
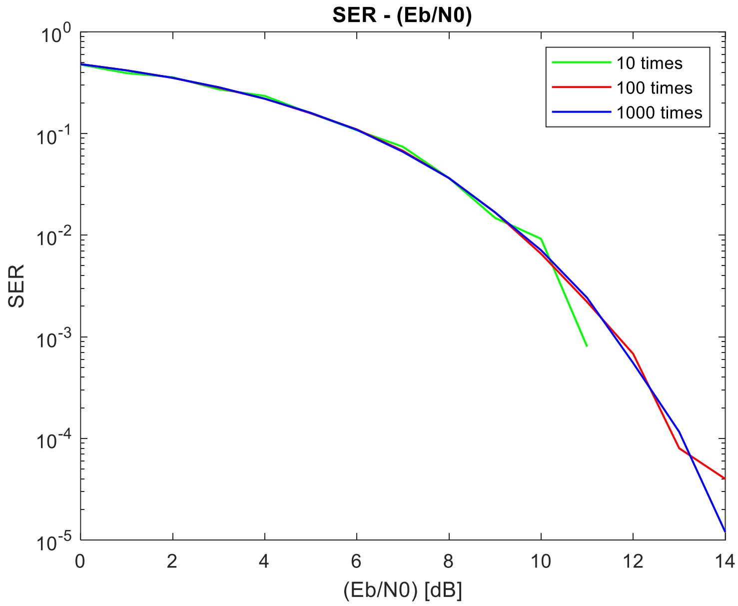
電機三 B05502145 林禹丞

Report

1. **SER & BER:**

   (a) Generate a sequence of n = 1000 bits and simulate transmitting the sequence over a 16-QAM communication system with different $Eb/N0$. Repeat the experiment R times. Each experiment randomly generates a sequence of bits. Plot the average SER vs. $Eb/N0$ and average BER vs. $Eb/N0$ curves with R = {10, 100, 1000}. Observe the changes among the curves.





   (b) (Handwriting) Please derive the theoretical SER of 16-QAM, and the upper/lower bound for SER of 16-QAM

16-QAM

$2d$

4 "corner": $1-\left(1-Q\left(\frac{d}{\sqrt{N_0/2}}\right)\right)^2$

8 "side": $1-\left(1-Q\left(\frac{d}{\sqrt{N_0/2}}\right)\right)\left(1-2Q\left(\frac{d}{\sqrt{N_0/2}}\right)\right)$

4 "middle": $1-\left(1-2Q\left(\frac{d}{\sqrt{N_0/2}}\right)\right)^2$

$$P_e = \frac{1}{16}\left[4\left[2Q-Q^2\right]+8\left[3Q-2Q^2\right]+4\left[4Q-4Q^2\right]\right]$$

$$= \frac{1}{16}\left[48Q-36Q^2\right]$$

$$= 3Q-\frac{9}{4}Q^2 \quad \text{where } Q := Q\left(\frac{d}{\sqrt{N_0/2}}\right)$$

energy per symbol of 16-QAM is $10d^2$

$$\Rightarrow SNR = \frac{10d^2}{N_0} \Rightarrow \sqrt{\frac{SNR}{10}} = \frac{d}{\sqrt{N_0}} \Rightarrow \sqrt{\frac{SNR}{5}} = \frac{d}{\sqrt{N_0/2}}$$

$$P_e = 3Q\left(\sqrt{\frac{SNR}{5}}\right)-\frac{9}{4}Q^2\left(\sqrt{\frac{SNR}{5}}\right) \#$$

By Union Bound:

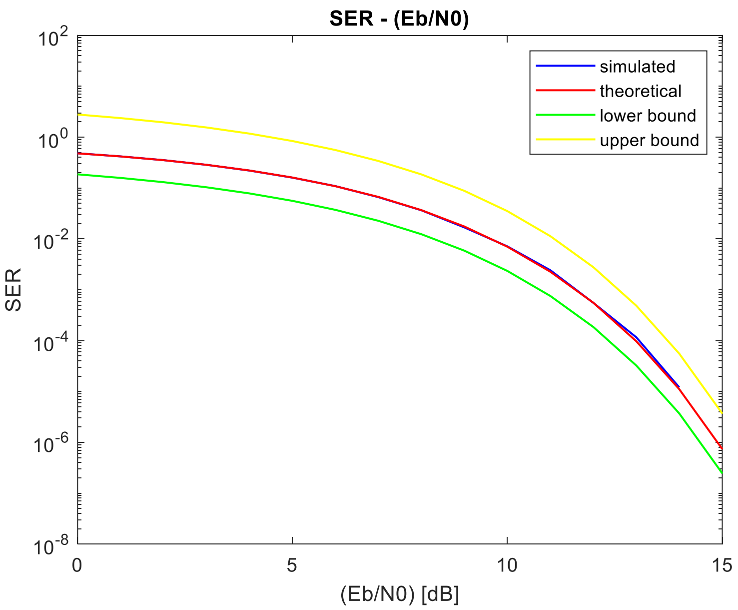
$$Q\left(\frac{d}{\sqrt{N_0/2}}\right) \le P_e \le 15Q\left(\frac{d}{\sqrt{N_0/2}}\right)$$

$\therefore$ lower bound is $Q\left(\sqrt{\frac{SNR}{5}}\right)$

upper bound is $15Q\left(\sqrt{\frac{SNR}{5}}\right)$

(c) Show simulated SER, theoretical SER, and upper/lower bound for SER in one figure.

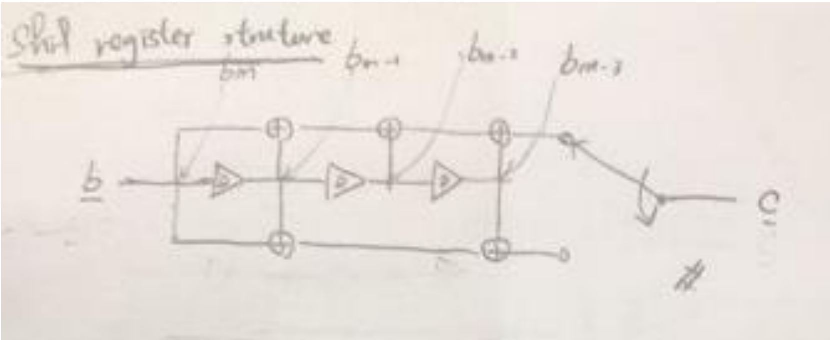
Follow the union bound derive from (c), we can get:



2. **Convolutional Code:**

(a) (Handwriting) A convolutional encoder has two finite impulse response (FIR), $\mathbf{h}_{(1)}$ = [1 1 1 1], $\mathbf{h}_{(2)}$ = [1 1 0 1]. Please **calculate the code rate**, and **plot shift register structure, state transition diagram, and trellis diagram**.

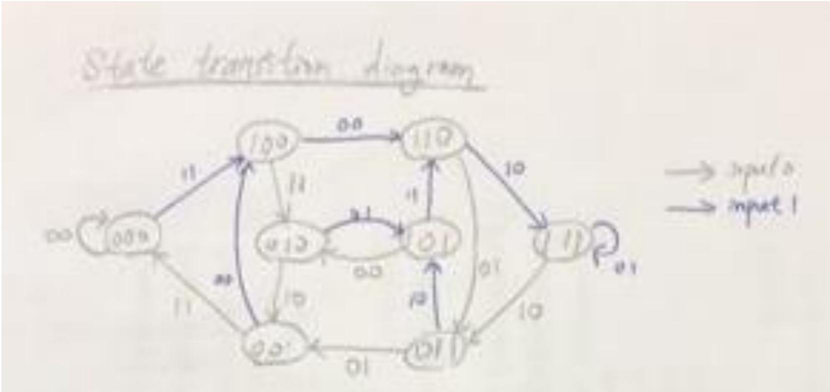
The code rate is 1/2.

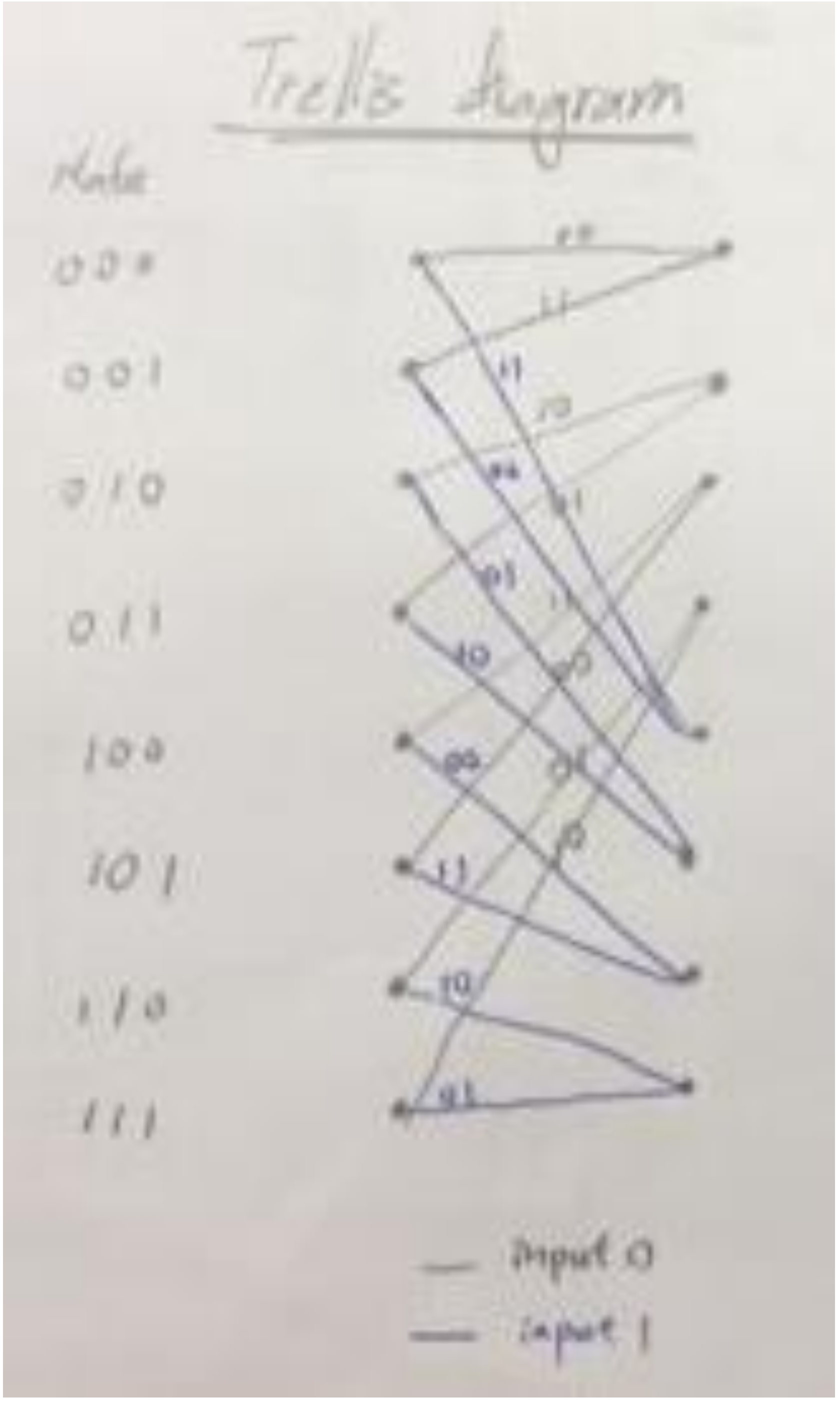


Set "$b_{m-1}\, b_{m-2}\, b_{m-3}$" as states,

b_m as input

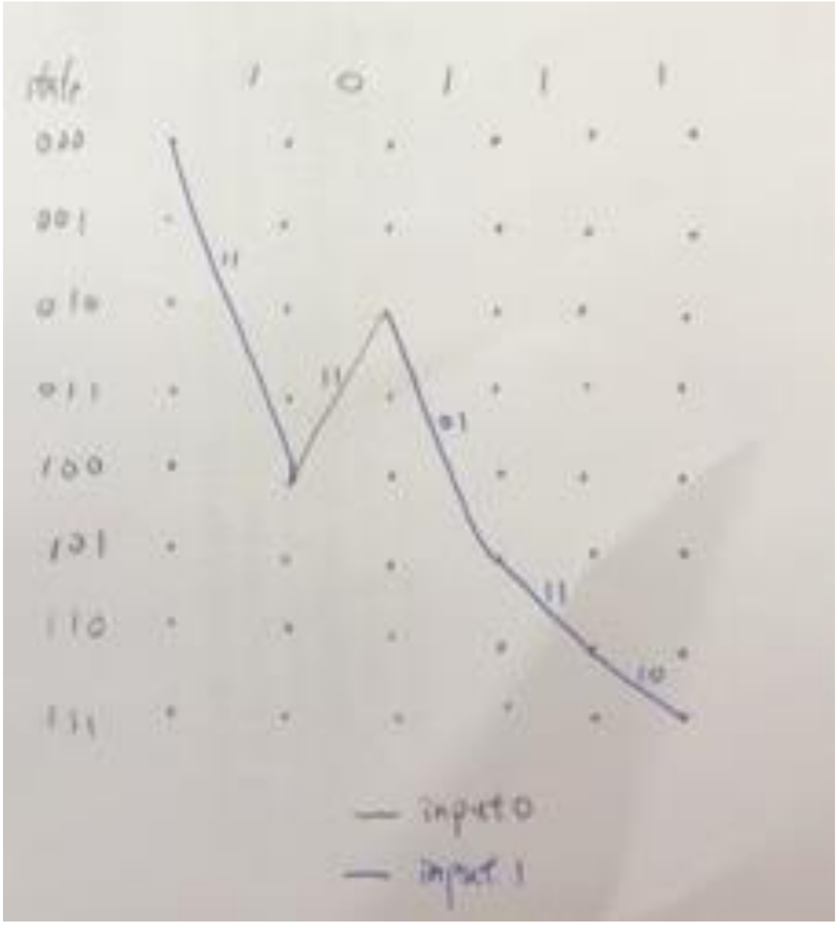
input	state	output	state (after)
0	000	00	000
0	001	11	000
0	010	10	001
0	011	01	001
0	100	11	010
0	101	00	010
0	110	01	011
0	111	10	011
1	000	11	100
1	001	00	100
1	010	01	101
1	011	10	101
1	100	00	110
1	101	11	110
1	110	10	111
1	111	01	111


State transition diagram
→ input 0
→ input 1


Trellis diagram
state
000
001
010
011
100
101
110
111
— input 0
— input 1

(b) (Handwriting) Following (a), trace the path through the trellis diagram corresponding to the message sequence *{10111}*.

The output is 11 11 01 11 10.



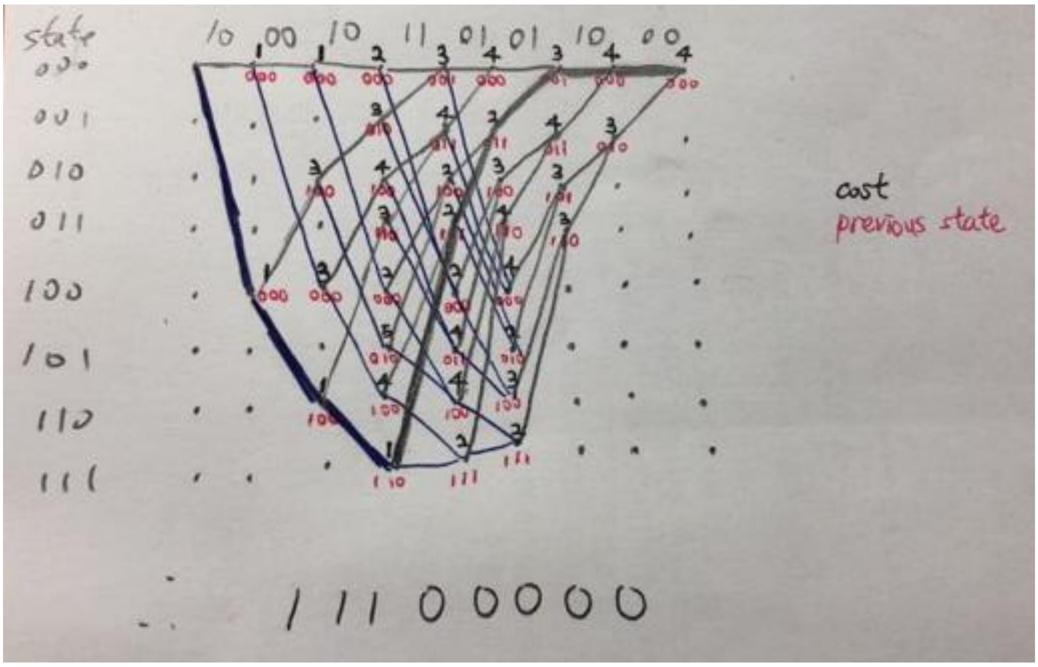
(c) Write a Matlab function to implement a convolutional encoder. The function should have the following arguments:

encoded_data = convolutional_enc(binary_data, impulse_response)

```
function encoded_data = convolutional_enc(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    len = length(binary_data);
    for ii = 1 : row
        encoded_data(ii,:) = mod(conv(binary_data,
impulse_response(ii,:)), 2);
    end
    encoded_data = reshape(encoded_data,1,[]);
end
```

(d) (Handwriting) Following (a), the received data is *{1000101101011000}*. Using the Viterbi algorithm, compute the decoded data.

By the Viterbi algorithm, we find that the input seq. might be: 11100 with 3 filling 0's.



(e) Write a Matlab function to implement a convolutional decoder. The function has the following arguments:

decoded_data = convolutional_dec(binary_data, impulse_response)

```
function decoded_data = convolutional_dec(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    num_state = 2^(col-1);
    states = de2bi([0:num_state-1], col-1, 'left-msb');
    Cu = zeros(num_state, row);
    Cd = zeros(num_state, row);
    for ii = 1 : row
        Cu(:,ii) = mod(sum([states
zeros(num_state,1)].*impulse_response(ii,:),2),2);
        Cd(:,ii) = mod(sum([states
ones(num_state,1)].*impulse_response(ii,:),2),2);
    end

    num_level = length(binary_data) / row;
    cost = inf(num_state, 1);
```

```
    cost(1) = 0;
    binary_data = reshape(binary_data,row,[])';
    idu = mod([0:num_state-1]'*2, num_state) + 1;
    idd = mod([0:num_state-1]'*2+1, num_state) + 1;
    previous = zeros(num_state, num_level);
    for ii = 1 : num_level
        costu = cost(idu) + sum(Cu ~= binary_data(ii,:), 2);
        costd = cost(idd) + sum(Cd ~= binary_data(ii,:), 2);
        [cost, idx] = min([costu costd], [], 2);
        idx = idx - 1;
        previous(:, ii) = idu.*(1 - idx) + idd.*(idx);
    end

    place = 1;
    decoded_data(num_level) = 0;
    for ii = num_level : -1 : 2
        place = previous(place, ii);
        decoded_data(ii-1) = (place > num_state/2);
    end
    decoded_data = decoded_data(1:(end-row));
end
```

3. **Communication System:** After four Labs, now you have gone through the entire communication system. Combine all processing blocks in each Lab, and simulate transmitting audio file (handel.ogg) to the communication system. Discuss the influence of the parameters (e.g., the number of levels in the quantizer, the constellation, Huffman coding, error correction code, SNR, and so on) on the quality of the received signal.

Quantizer(x-bit)	Code rate(ECC)	Constellation	SNR	Result
6	1/2	16-QAM	20	可清楚聽到原音樂
6	1/2	16-QAM	10	有些微雜音
6	1/2	16-QAM	5	完全都是雜音
4	1/2	16-QAM	20	有輕微雜音
4	1/2	16-QAM	10	有輕微雜音，聽起來

				比6-bit的情況少雜音
4	1/2	16-QAM	5	完全都是雜音
6	1/2	4-PAM	20	有非常非常小的雜音，聽起來和原音樂幾乎一樣
6	1/2	4-PAM	10	輕微雜音，聽起來和原音樂差不多
6	1/3	4-PAM	20	接近完美
6	1/3	4-PAM	10	非常小的雜音，幾乎聽不到雜音
6	1/3	4-PAM	5	聽到雜音，但還是可以辨別出是原音樂
6	1/3	8-PAM	5	非常重的雜音

Quantizer: 4-bit 和 6-bit 比較可以聽出來 4-bit 的雜訊聽起來比較沒有那麼明顯，我覺得原因可能是 4-bit 量化之後的點比較少，所以她的資料會比較短在 Huffmann 的時候比較不會導致錯一個、後面全錯的問題，還有可表示的震幅比較少，所以錯誤的地方也不會那麼明顯，比較明顯的雜訊主要是來自量化位元太少所產生的雜音。

Constellation: 4-PAM 和 16-QAM 聽起來差別不大，而 4-PAM 和 8-PAM 可以聽出來 8-PAM 的雜音明顯比較重。

SNR: 越大雜訊越小

Code rate: 也就是有幾個 filter 來做 convolution，可以看到在同樣 6-bit 4-PAM SNR5 的情況下，原本聽起來幾乎是完全雜訊，convolution code 1/3就可以把錯誤率大幅降低，讓我們可以聽到原音訊。

Appendix

1. Code for problem1(Calculating BER and SER and save it)

```
% prolbem 1

R = [10 100 1000];                              % different
simulation times
N = 1000;                                      % number of bits
sym = load('../hw3/cell_prob1/16-QAM-Gray.mat');    % load
constellation
sym = [sym.A{:}];
EboN0_list = 10.^([0:30]/10);
ber_res = zeros(length(R), length(EboN0_list));
ser_res = zeros(length(R), length(EboN0_list));

rng(2);
for ii = 1 : length(R)
    for jj = 1 : length(EboN0_list)
        EboN0 = EboN0_list(jj);
        message = randi([0 1], R(ii), N);
        symbol_map = reshape(bi2de(fliplr(reshape(message', 4,
[])'))' + 1, N/4, [])';
        symbol = sym(symbol_map);
        Es = sum(abs(symbol).^2, 2) / (N/4);
        noise_var = Es / (4*EboN0);

        pd = makedist('Normal');
        noise = random(pd, R(ii), N / 4).*sqrt(noise_var/2) + i *
random(pd, R(ii) ,N / 4).*sqrt(noise_var/2);
        symbol_rec = symbol + noise;
        [symbol_demap, message_rec] = demap(symbol_rec);

        ber_res(ii, jj) = sum(message ~= message_rec, 'all') /
numel(message);
        ser_res(ii, jj) = sum(symbol_map ~= symbol_demap, 'all') /
numel(message) * 4;

        % figure;
```

```
        % xx = roundn(real(symbol_rec), -6);
        % yy = roundn(imag(symbol_rec), -6);
        % plot(xx, yy, 'o', 'MarkerSize', 6, 'MarkerEdgeColor','b',
'MarkerFaceColor',[0.5,0.5,0.5]);
        % hold on;
        % xx = roundn(real(symbol), -6);
        % yy = roundn(imag(symbol), -6);
        % plot(xx, yy, 'o', 'MarkerSize', 8, 'MarkerEdgeColor','r',
'MarkerFaceColor',[1,0,0]);
    end
end

save('problem1_result.mat', 'ber_res', 'ser_res');

% 16-QAM demapper
function [sym_out, bin_out] = demap(sym_in)
    persistent sym
        if isempty(sym)
            sym = load('../hw3/cell_prob1/16-QAM-Gray.mat');
            sym = [sym.A{:}];
        end
    [row, sym_len] = size(sym_in);
    C = zeros(row, sym_len);
    for ii = 1 : sym_len
        [minvalue, index_of_min] = min(abs(sym_in(:, ii) - sym), [],
2);
        C(:,ii) = index_of_min - 1;
    end
    sym_out = C + 1;
    bin_out = reshape(fliplr(de2bi(C',4))',sym_len*4,[])';
end
```

2. Code for plotting BER/SER of problem1 and plotting upper/lower bounds.

```
% plot the BER-(Eb/N0) curve

% load ber data
T = load('problem1_result.mat');
EboN0_list = [0:30];        % dB
```

```
ber_res = T.ber_res;
ser_res = T.ser_res;
SNR = 10.^(EboN0_list/10)*4;
ser_formula = qfunc(sqrt(SNR/5));

% plot BER - (Eb/N0)
figure;
plot(EboN0_list, ber_res(1,:), '-', 'color', 'g', 'LineWidth', 1);
hold on;
plot(EboN0_list, ber_res(2,:), '-', 'color', 'r', 'LineWidth', 1);
plot(EboN0_list, ber_res(3,:), '-', 'color', 'b', 'LineWidth', 1);
legend('10 times', '100 times', '1000 times');
set(gca, 'YScale', 'log')
title('BER - (Eb/N0)');
ylabel('BER');
xlabel('(Eb/N0) [dB]');
hold off;

% plot SER - (Eb/N0)
figure;
plot(EboN0_list, ser_res(1,:), '-', 'color', 'g', 'LineWidth', 1);
hold on;
plot(EboN0_list, ser_res(2,:), '-', 'color', 'r', 'LineWidth', 1);
plot(EboN0_list, ser_res(3,:), '-', 'color', 'b', 'LineWidth', 1);
legend('10 times', '100 times', '1000 times');
set(gca, 'YScale', 'log')
title('SER - (Eb/N0)');
ylabel('SER');
xlabel('(Eb/N0) [dB]');
hold off;

% plot SER - (Eb/N0)
figure;
plot(EboN0_list, ser_res(3,:), '-', 'color', 'b', 'LineWidth', 1);
hold on;
plot(EboN0_list, 3*ser_formula-9/4*ser_formula.^2, '-', 'color',
'r', 'LineWidth', 1);
plot(EboN0_list, ser_formula, '-', 'color', 'g', 'LineWidth', 1);
```

```
plot(EboN0_list, 15*ser_formula, '-', 'color', 'y', 'LineWidth', 1);
legend('simulated', 'theoretical', 'lower bound', 'upper bound');
set(gca, 'YScale', 'log')
title('SER - (Eb/N0)');
xlim([0,15])
ylabel('SER');
xlabel('(Eb/N0) [dB]');
hold off;
```

3. Code for problem 2

```
rng(0);
impulse_response = [1 1 1 1; 1 1 0 1; 1 0 0 0];
% message = randi([0 1], 1, 20);
message = [1 1 1 0 0];
encoded_data = convolutional_enc(message, impulse_response);

message2 = [1 0 0 0 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 0 1 0 0 0];
decoded_data = convolutional_dec(message2, impulse_response);

function encoded_data = convolutional_enc(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    len = length(binary_data);
    for ii = 1 : row
        encoded_data(ii,:) = mod(conv(binary_data,
impulse_response(ii,:)), 2);
    end
    encoded_data = reshape(encoded_data,1,[]);
end

function decoded_data = convolutional_dec(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    num_state = 2^(col-1);
    states = de2bi([0:num_state-1], col-1, 'left-msb');
    Cu = zeros(num_state, row);
    Cd = zeros(num_state, row);
    for ii = 1 : row
```

```
        Cu(:,ii) = mod(sum([states
zeros(num_state,1)].*impulse_response(ii,:),2),2);
        Cd(:,ii) = mod(sum([states
ones(num_state,1)].*impulse_response(ii,:),2),2);
    end

    num_level = length(binary_data) / row;
    cost = inf(num_state, 1);
    cost(1) = 0;
    binary_data = reshape(binary_data,row,[])';
    idu = mod([0:num_state-1]'*2, num_state) + 1;
    idd = mod([0:num_state-1]'*2+1, num_state) + 1;
    previous = zeros(num_state, num_level);
    for ii = 1 : num_level
        costu = cost(idu) + sum(Cu ~= binary_data(ii,:), 2);
        costd = cost(idd) + sum(Cd ~= binary_data(ii,:), 2);
        [cost, idx] = min([costu costd], [], 2);
        idx = idx - 1;
        previous(:, ii) = idu.*(1 - idx) + idd.*(idx);
    end

    place = 1;
    decoded_data(num_level) = 0;
    for ii = num_level : -1 : 2
        place = previous(place, ii);
        decoded_data(ii-1) = (place > num_state/2);
    end
    decoded_data = decoded_data(1:(end-row));
end
```

4. Code for problem 3

```
%% problem 2

% assume amp is in [-1, 1]
clear all;
[x, fs] = audioread('handel.ogg');
xmax = 1;
bit = 6;
```

```
level = 2^bit;

% quantizer
fprintf("Quantizing...\n");
xt = quantizer_L_level(x, xmax, level)';

% huffman encode
fprintf("Huffman encoding...\n");
delta = 2 * xmax / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p = histc(xt, symbols);
p = p / sum(p);
dict = huffmandict(symbols, p);
y_huffen = huffmanenco(xt, dict);

% convolution encode
fprintf("ECC encoding...\n");
impulse_response = [1 1 1 1; 1 1 0 1];
y_conven = convolutional_enc(y_huffen, impulse_response);

% constellation mapping & demapping
fprintf("Passing constellation...\n");
num1 = 16;
constellation1 = 'QAM';
mapping1 = 'gray';
SNR = 5;
y_cons = Run(y_conven, num1, 1, constellation1, mapping1, 'MD',
SNR);

% convolution decode
fprintf("ECC decoding...\n");
y_convde = convolutional_dec(y_cons, impulse_response);

% huffman decode
fprintf("Huffman decoding...\n");
y_huffde = huffmandeco(y_convde, dict);

function y = quantizer_L_level(x, xmax, level)
```

```
    delta = 2 * xmax / level;
    partition = [-xmax:delta:xmax];
    codebook = [0,-(level-1)*delta/2:delta:(level-1)*delta/2,0];
    [I, y] = quantiz(x,partition,codebook);
end

function binary_sequence_rec = Run(binary_sequence, M, d,
constellation, mapping, decision_rule, SNR)
    tit = strcat(num2str(M), '-', constellation, '-', mapping);
    A = {};
    load(strcat('../hw3/cell_prob1/', tit,'.mat'), 'A');
    B = [A{:}];
    Es = sum(abs(B).^2)/length(B);
    noise_var = Es/(10^(SNR/10));
    pd = makedist('Normal');
    symbol_sequence = symbol_mapper(binary_sequence, M, d,
constellation, mapping);
    save('Noise.mat','noise_var');
    NI = random(pd,1,length(symbol_sequence))*sqrt(noise_var/2);
    NQ = random(pd,1,length(symbol_sequence))*sqrt(noise_var/2);
    symbol_sequence_rec = symbol_sequence + NI + i*NQ;
    binary_sequence_rec = symbol_demapper(symbol_sequence_rec, M, d,
constellation, mapping, decision_rule);
    binary_seq_len = length(binary_sequence);
    berr = sum(binary_sequence ~= binary_sequence_rec) /
binary_seq_len;
    fprintf("%13s Bit error rate   %f\n", tit, berr);
    tit = strcat(tit, '-', decision_rule);
end

function binary_sequence = symbol_demapper(symbol_sequence, M, d,
constellation, mapping, decision_rule)
    switch decision_rule
        case 'MD'
            C = cell(length(symbol_sequence),1);
            tit = strcat(num2str(M), '-', constellation, '-',
mapping);
            A = {};
```

```
            load(strcat('../hw3/cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                [minvalue, index_of_min] = min(abs(symbol_sequence(ii)
- B));
                C{ii,1} = index_of_min-1;
            end
            binary_sequence =
reshape(fliplr(de2bi([C{:}],log2(M)))',1,[]);
        case 'ML'
            noise_var = 0;
            load('Noise.mat', 'noise_var');
            C = cell(length(symbol_sequence),1);
            tit = strcat(num2str(M), '-', constellation, '-',
mapping);
            A = {};
            load(strcat('../hw3/cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            P = ones(1,length(B))/length(B);
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                [maxvalue, index_of_max] = max(exp(-abs(B -
symbol_sequence(ii)).^2/noise_var).*P);
                C{ii,1} = index_of_max-1;
            end
            binary_sequence =
reshape(fliplr(de2bi([C{:}],log2(M)))',1,[]);
        case 'MAP'
            P = 0;
            noise_var = 0;
            load('Prob.mat','P');
            load('Noise.mat', 'noise_var');
            C = cell(length(symbol_sequence),1);
            tit = strcat(num2str(M), '-', constellation, '-',
mapping);
            A = {};
            load(strcat('../hw3/cell_prob1/', tit,'.mat'), 'A');
```

```
            B = [A{:}];
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                [maxvalue, index_of_max] = max(exp(-abs(B -
symbol_sequence(ii)).^2/noise_var).*P);
                C{ii,1} = index_of_max-1;
            end
            binary_sequence =
reshape(fliplr(de2bi([C{:}],log2(M)))',1,[]);
        otherwise
            error("Only 'MD' 'ML' 'MAP' are available")
    end
end

function symbol_sequence = symbol_mapper(binary_sequence, M, d,
constellation, mapping);
    seq = reshape(binary_sequence,log2(M),[])';
    C = cell(length(seq),1);
    tit = strcat(num2str(M), '-', constellation, '-', mapping);
    A = {};
    load(strcat('../hw3/cell_prob1/', tit,'.mat'), 'A');
    B = [A{:}];
    for ii = 1:length(seq)
        C{ii,1} = B(bi2de(fliplr(seq(ii,:)))+1);
    end
    symbol_sequence = [C{:}];
    P = zeros(1,length(B));
    for ii = 1:length(B)
        P(ii) = sum(symbol_sequence(:) == B(ii));
    end
    P = P / length(symbol_sequence);
    save('Prob.mat', 'P');
end

function encoded_data = convolutional_enc(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    len = length(binary_data);
```

```
    for ii = 1 : row
        encoded_data(ii,:) = mod(conv(binary_data,
impulse_response(ii,:)), 2);
    end
    encoded_data = reshape(encoded_data,1,[]);
end

function decoded_data = convolutional_dec(binary_data,
impulse_response)
    [row, col] = size(impulse_response);
    num_state = 2^(col-1);
    states = de2bi([0:num_state-1], col-1, 'left-msb');
    Cu = zeros(num_state, row);
    Cd = zeros(num_state, row);
    for ii = 1 : row
        Cu(:,ii) = mod(sum([states
zeros(num_state,1)].*impulse_response(ii,:),2),2);
        Cd(:,ii) = mod(sum([states
ones(num_state,1)].*impulse_response(ii,:),2),2);
    end

    num_level = length(binary_data) / row;
    cost = inf(num_state, 1);
    cost(1) = 0;
    binary_data = reshape(binary_data,row,[])';
    idu = mod([0:num_state-1]'*2, num_state) + 1;
    idd = mod([0:num_state-1]'*2+1, num_state) + 1;
    previous = zeros(num_state, num_level);
    for ii = 1 : num_level
        costu = cost(idu) + sum(Cu ~= binary_data(ii,:), 2);
        costd = cost(idd) + sum(Cd ~= binary_data(ii,:), 2);
        [cost, idx] = min([costu costd], [], 2);
        idx = idx - 1;
        previous(:, ii) = idu.*(1 - idx) + idd.*(idx);
    end

    place = 1;
    decoded_data(num_level) = 0;
```

```
    for ii = num_level : -1 : 2
        place = previous(place, ii);
        decoded_data(ii-1) = (place > num_state/2);
    end
    decoded_data = decoded_data(1:(end-row));
end
```