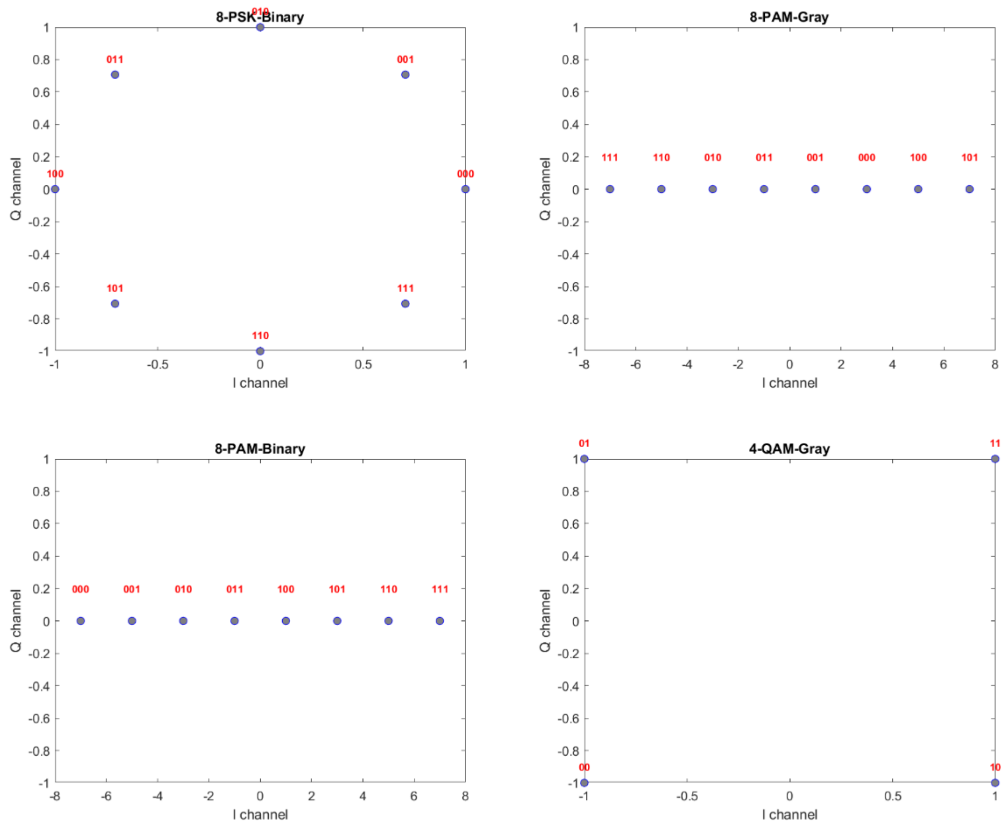
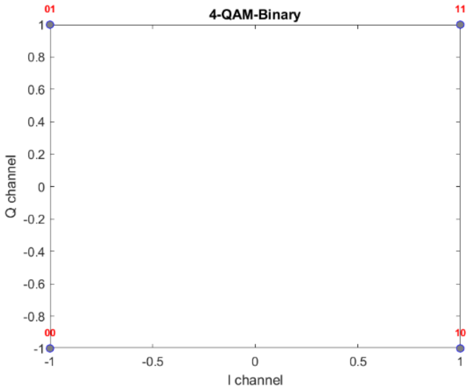
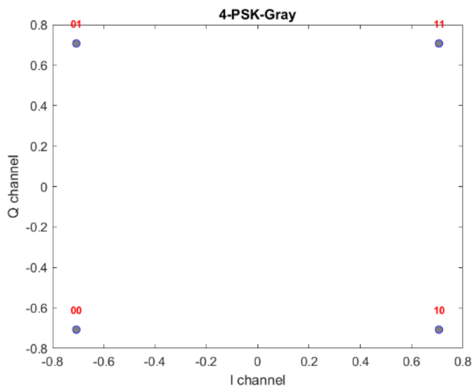
電機三 B05502145 林禹丞

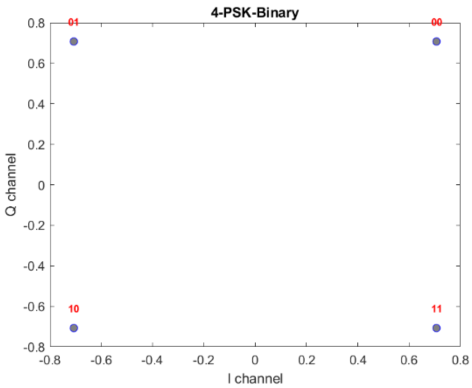
Report

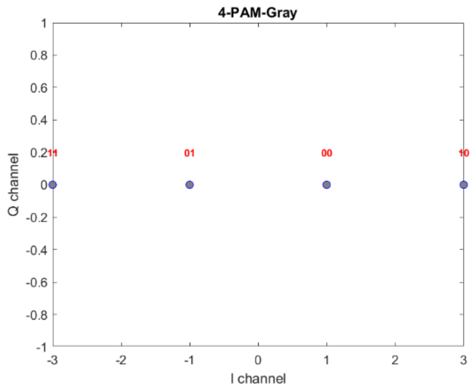
1. **Symbol Mapping**: Write a Matlab function to modulate an incoming binary sequence using different constellations
   symbol_sequence = symbol_mapper(binary_sequence, M, d, ... constellation, mapping)
   • Inputs:
   – M: The number of points in the signal constellation and M = 2,4,8,16 for 'PAM' and 'PSK', M = 4,16 for 'QAM'
   – d: The minimum distance among the constellation.
   – constellation: 'PAM', 'PSK' or 'QAM'.
   – mapping: 'Binary' or 'Gray'
   If input M is not the power of 2 or if input constellation is not defined, your function should be able to throw error and display message. Plot all the constellations, and show the bits of each symbol on the constellation points.

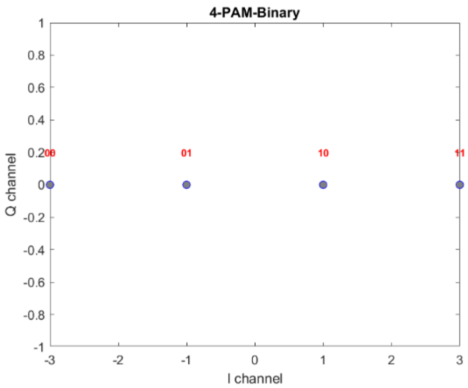


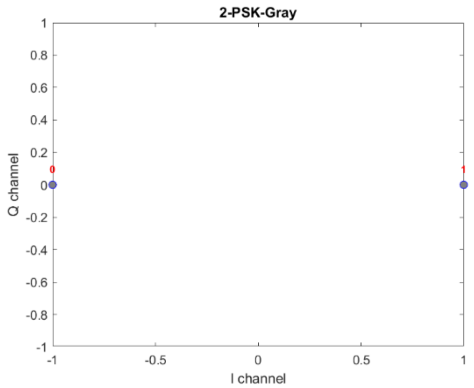

4-QAM-Binary
01
11
00
10
Q channel
I channel

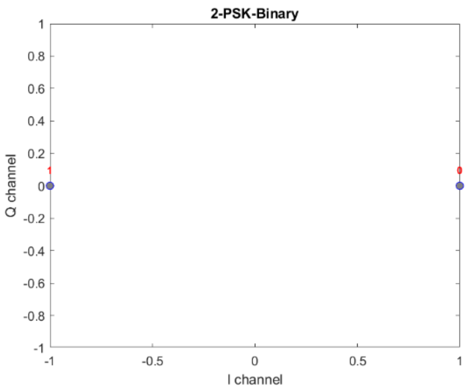

4-PSK-Gray
01
11
00
10
Q channel
I channel

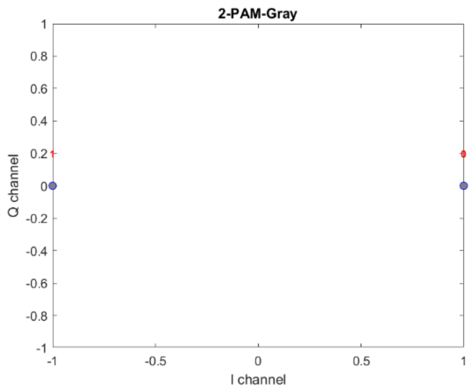

4-PSK-Binary
01
00
10
11
Q channel
I channel

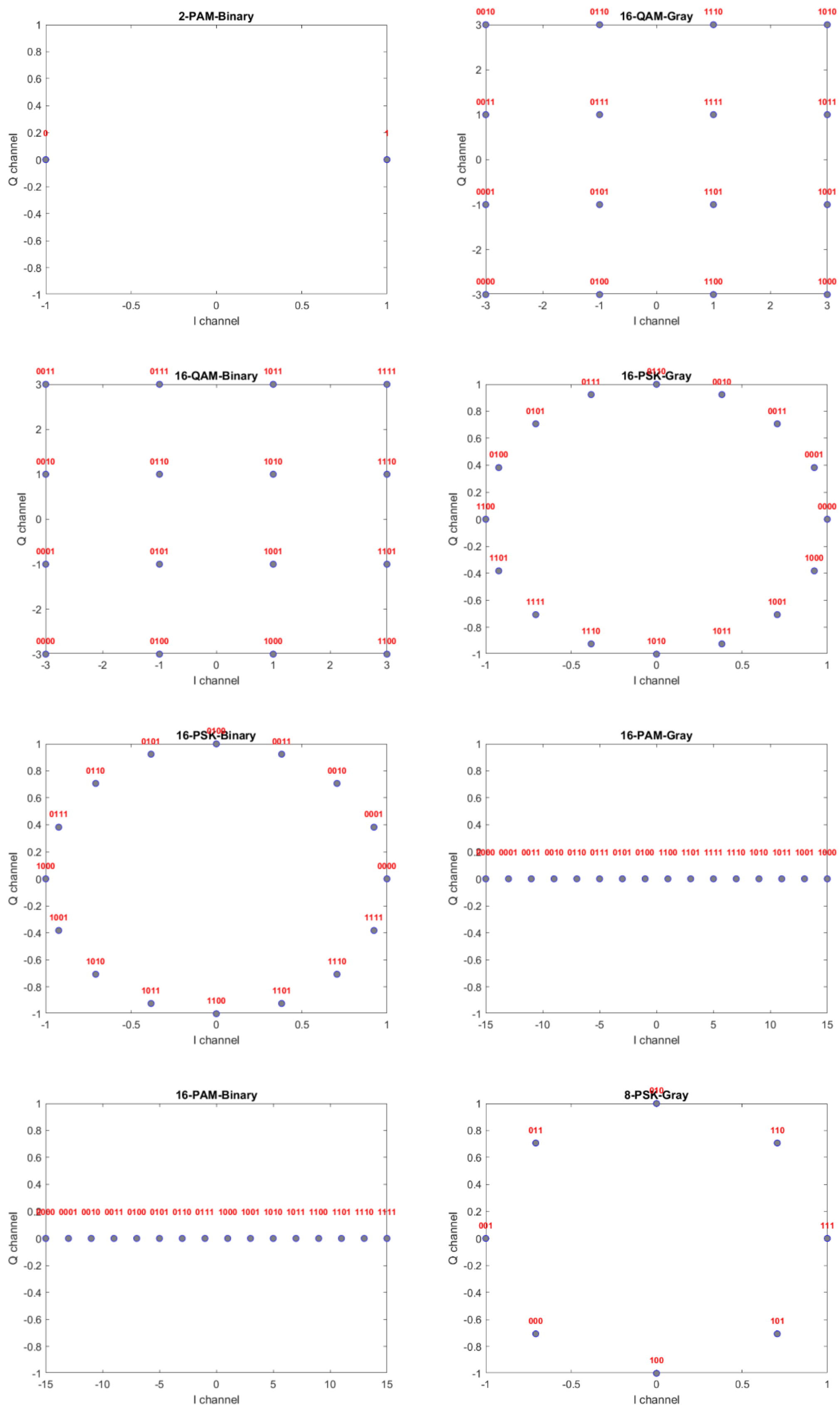

4-PAM-Gray
11
01
00
10
Q channel
I channel


4-PAM-Binary
00
01
10
11
Q channel
I channel


2-PSK-Gray
0
1
Q channel
I channel


2-PSK-Binary
1
0
Q channel
I channel


2-PAM-Gray
1
0
Q channel
I channel



2. **Pulse Shaping**: We will design $p(t)$ and $q(t)$ such that $u_m = \hat{u}_m$ for all $m \in Z$. Note that pulse function $p(t)$ satisfies $g(t) = p(t) * q(t)$. One possible choice of $q(t)$ is to

use matched filtering q(t) = p ∗ (−t). In Matlab, we simulate the continuous time by **oversampling**.

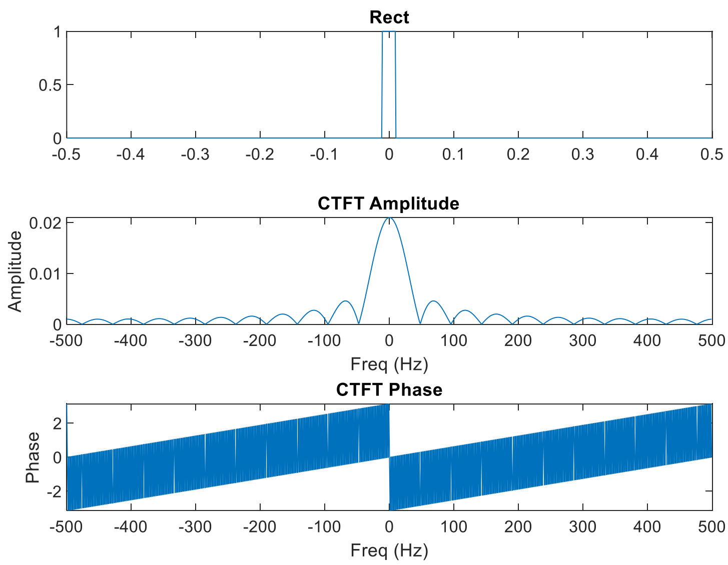
In time domain, the Ideal Nyquist is hold iff

$$g(nT) = \begin{cases} 0 & \text{if } n \neq 0 \\ 1 & \text{if } n = 0 \end{cases}.$$

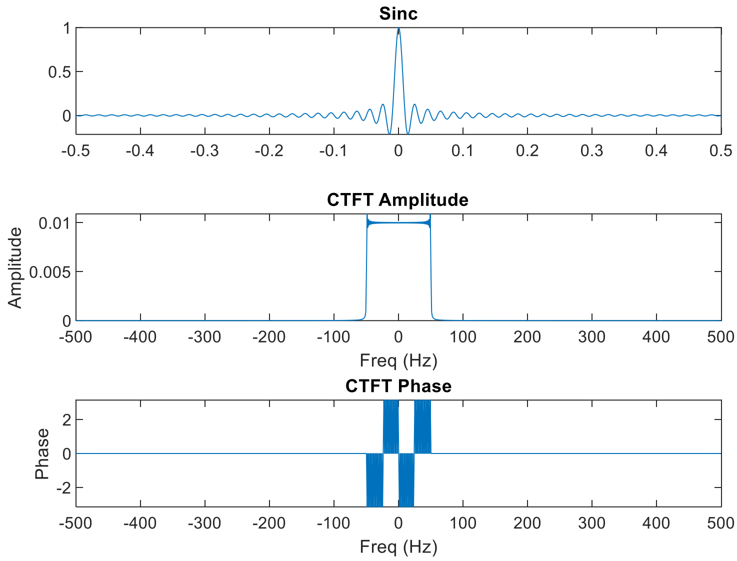
In freq. domain, the Nyquist criterion is hold iff

$$\sum_{m=-\infty}^{\infty} \breve{g}\left(f - \frac{m}{T}\right) = T, \quad for\ |f| \leq W \equiv \frac{1}{2T}.$$

(a) Set the operational bandwidth W = 50 Hz. Choose **rectangular function** as g(t). Plot the function and its Fourier transform. **Validate the Nyquist criterion in the time-domain and the frequency-domain**.
In time domain, rect function is 1 at 0, and 0 at nT, so **it satisfy Ideal Nyquist**.
In freq. domain, we summing the shift of rect function and found that it is all T so it **satisfy Nyquist criterion**.



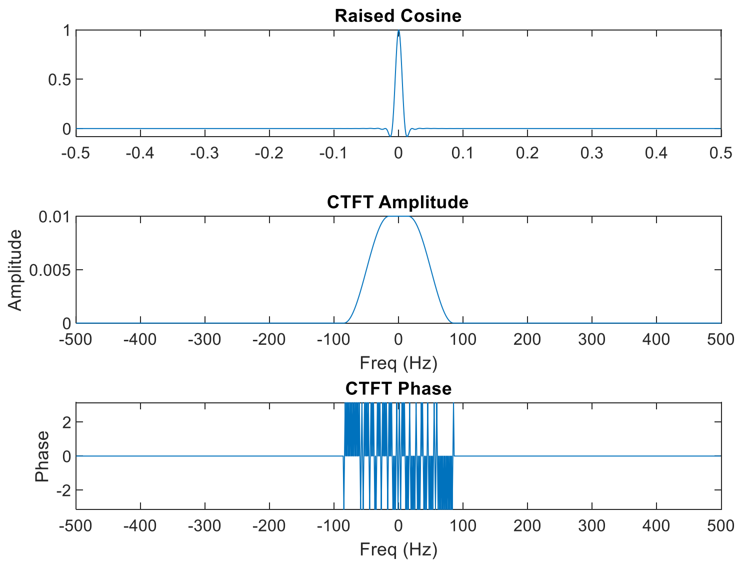
(b) Following (a), change g(t) to **sinc function** g(t) = sinc(t/T) Plot the function and its Fourier transform. **Validate the Nyquist criterion in the time-domain and the frequency-domain**.
In time domain, sinc function is 1 at 0, and 0 at nT, so **it satisfy Ideal Nyquist**.
In freq. domain, sinc function satisfy band-edge symmetry so **it also satisfy Nyquist criterion**.



(c) Following (a), change g(t) to **raised cosine function**. Plot the function and its Fourier transform. **Validate the Nyquist criterion in the time-domain and the frequency-domain**.

In time domain, raised cosine function is 1 at 0, and 0 at nT, so **it satisfy Ideal Nyquist**.

In freq. domain, raised cosine function satisfy band-edge symmetry so **it also satisfy Nyquist criterion**.



(d) **Observe the power decay in time-domain and bandwidth in frequency-domain of different pulses**. Describe which pulse you will choose for pulse shaping.

We will choose the raised-cosine pulse as pulse shaping, since its power in time domain is concentrate near zero. When we do pulse shaping, we must

truncate the signal, so if we choose other pulse for shaping, we might loss large portion of energy which cause more error.
We can observe that the decay speed in time and freq. domain has duality, if the decay is faster in freq. domain, the decay is slower in time domain. So in order to reduce error for truncation, we choose pulse with fast decay in time, and slow(smooth) decay in freq., that is Raised cosine.

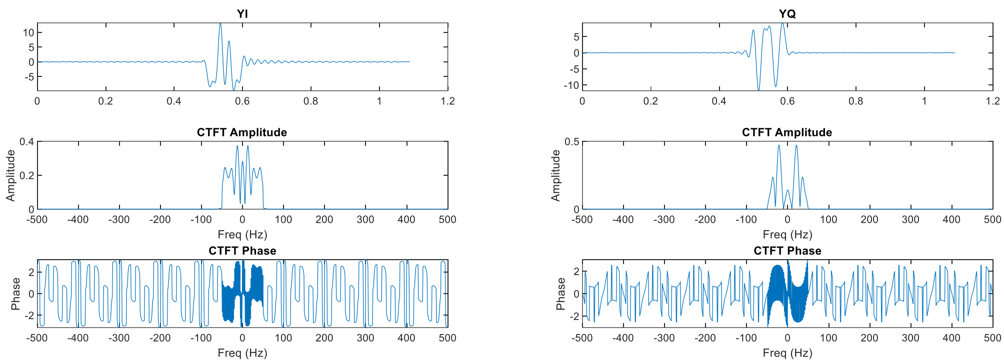
(e) Write a Matlab function to implement the pulse shaping filter p(t). Your own pulse shaping function should have the following arguments,
y = pulse_shaper(x, pulse_shape, W)
• Input:
– x: Input symbols after modulation (QPSK, 16-QAM, etc.).
– pulse_shape: 'sinc' or 'raised cosine' or others.
– W: The operational bandwidth.
• Output:
– y: The signal after pulse-shaping

```
function y = pulse_shaper(x, pulse_shape, W)
    oversampling_factor = 100000;
    T_os = 1/oversampling_factor; % time spacing after oversampling
    W = 50;
    T = 1/W;
    pulse_duration = 1; % 1 sec
    t_axis = (-pulse_duration/2 : T_os : pulse_duration/2 - T_os);
    Rect = ones(1, length(t_axis));
    RSinc = sin(pi*t_axis/T)./(pi*t_axis/T) / sqrt(T); RSinc(1,
find(isnan(RSinc))) = 1 / sqrt(T);
    beta = 0;
    RRaised = ( sin(pi*t_axis*(1-beta)/T) +
4*beta*t_axis/T.*cos(pi*t_axis/T*(1+beta)) ) ./ ( pi*t_axis/T.*(1-
(4*beta*t_axis/T).^2) ) / sqrt(T);

    switch pulse_shape
    case 'sinc'
        Pul = RSinc;
    case 'raised cosine'
        Pul = RRaised;
    otherwise
```

```
        error("Only 'sinc' and 'raised cosine' are valid for pulse
shape.")
    end

    save('Pulse.mat', 'Pul');
    O = zeros(1, (length(x))*length(Pul));
    O(1:length(Pul)) = Pul;
    y = zeros(1, length(O));
    for ii = 1 : length(x)
        y = y + x(1,ii)*circshift(O,length(Pul)*(ii-1));
    end

end
```

(f) Generate a 20-bits random binary sequence, and let the [0 1] with equal probability. Modulate the binary sequence using Gray-coded QPSK with d = 2, and then pass the symbols through the pulse shaper. **Plot real part and imaginary part of the output signal** and **indicate what pulse do you use**.

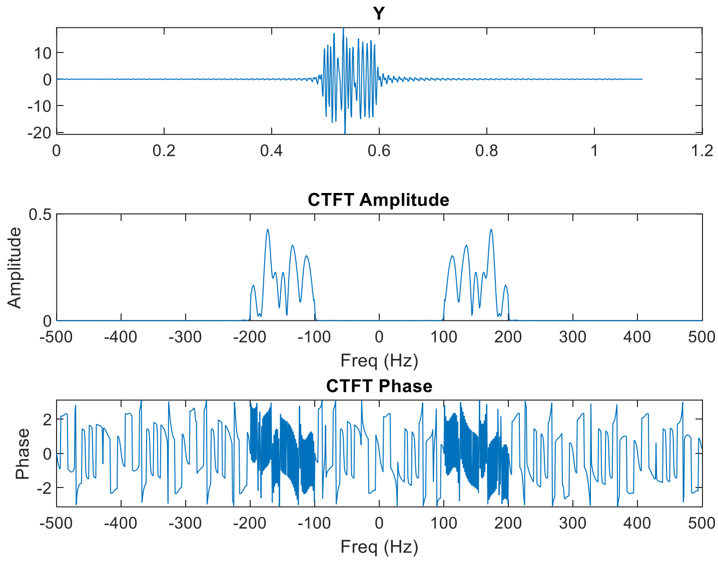
The left-hand-side is the real part, and the other is the imaginary part. We apply raised-cosine pulse as pulse shaping.



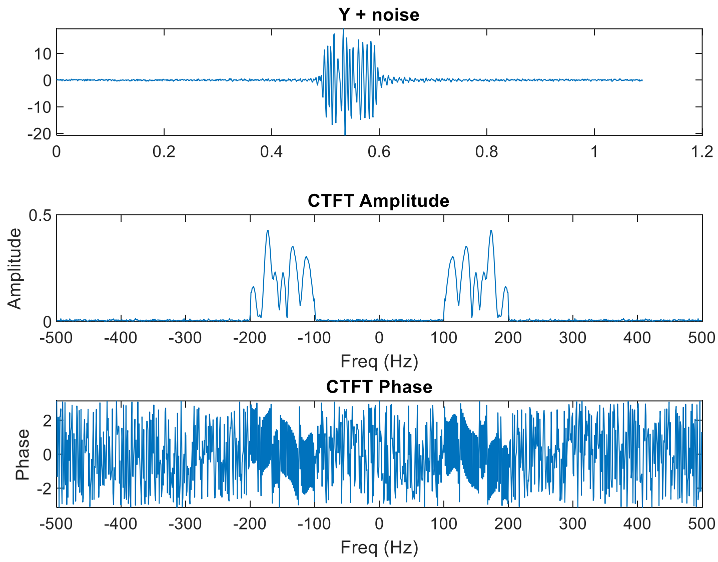
s

3. Baseband & Passband conversion: Following 2(f), here we will implement up-converter and downconverter. Assume carrier frequency is 100 Hz.

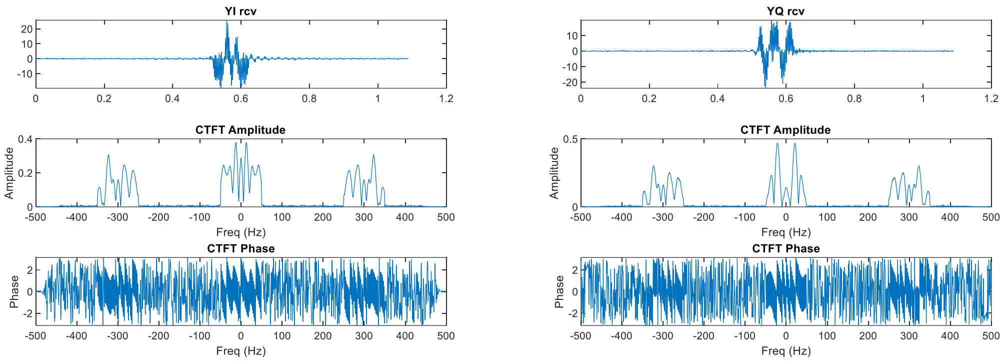
(a) You have a baseband signal $x_b(t) = x^{(I)}_b(t) + jx^{(Q)}_b(t)$ complex output signal of pulse shaper. Multiply $x^{(I)}_b(t)$ with in-phase carrier and multiply $x^{(Q)}_b(t)$ with quadrature-phase carrier. $x(t) = x^{(I)}_b(t)\sqrt{2}\cos(2\pi f_c t) - x^{(Q)}_b(t)\sqrt{2}\sin(2\pi f_c t)$ **Plot the output signal x(t)**.



(b) In real world, we have noisy channels. Assume we pass the transmitted signal through AWGN channel with SNR = 25 dB. **Plot the noisy signal**.

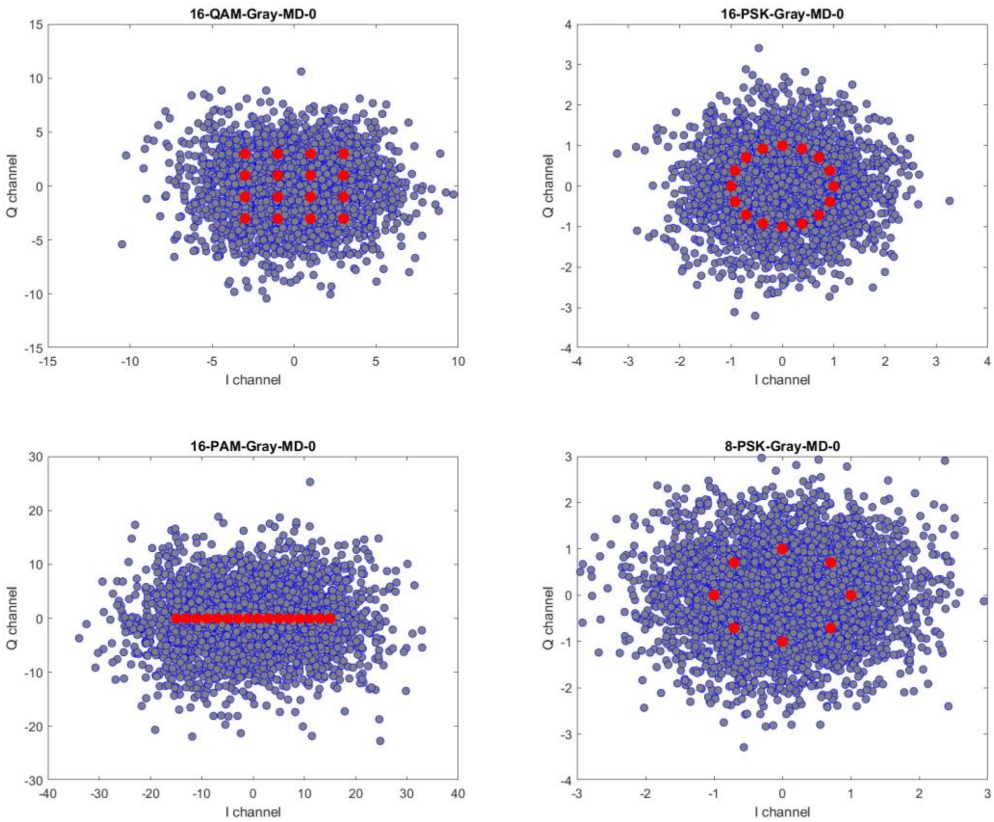


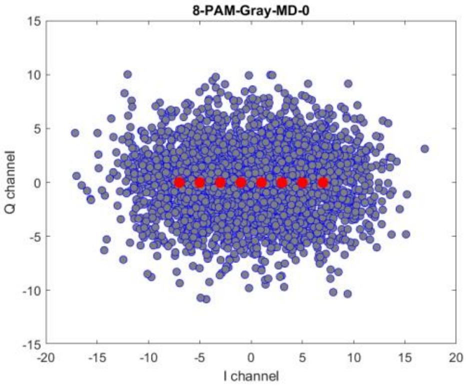
(c) Multiply the received signal with in-phase carrier, and multiply the received signal with quadrature-phase carrier. Next, the I/Q channels can be obtained from **lowpass filtering**. You can implement lowpass by convolve signal with a sinc function. **Plot real part and imaginary part of the output signal after lowpass filtering**.

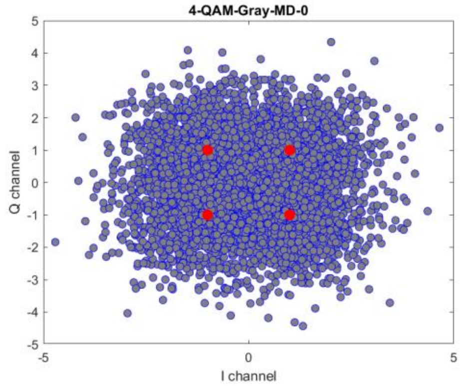


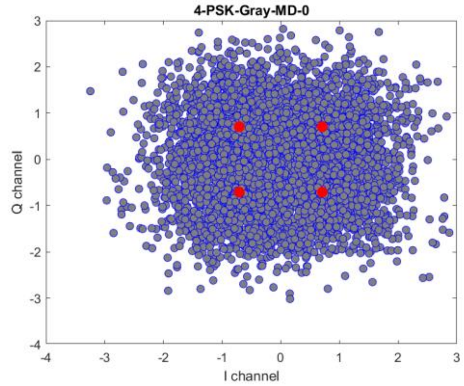
We can see that the result seems the same with 2(f).

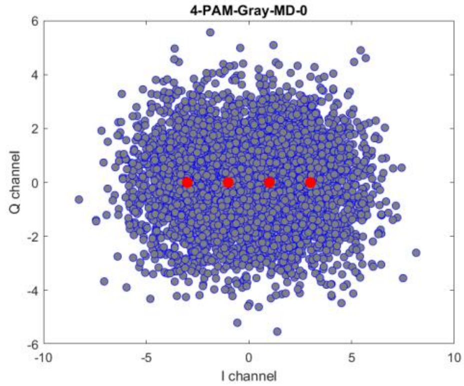
4. **MAP/ML/MD**: In this part we will have to demodulate the received noisy signal. With different prior probability of [0 1] and detection rule, we may get different demodulated symbols. Here, to simplifying the analysis of communication in the presence of noise, we only need to focus on the constellation domain. The transmitted symbol sequence does not go through pulse shaping and Baseband & Passband process. We directly add circularly symmetric complex Gaussian noise to the transmitted symbol sequence.
   (a) **Plot** the constellation of noisy symbol sequence with SNR = {0,10,20}.
      (1) All constellation with gray code and SNR = 0.

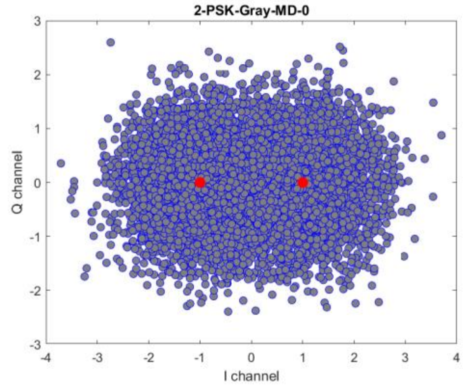


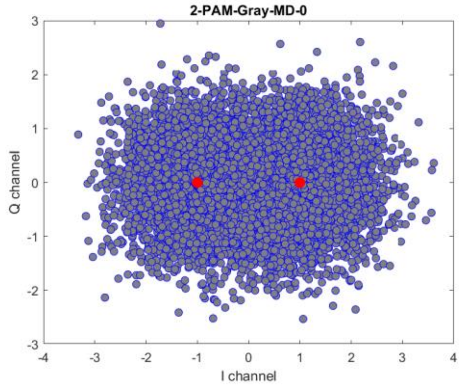




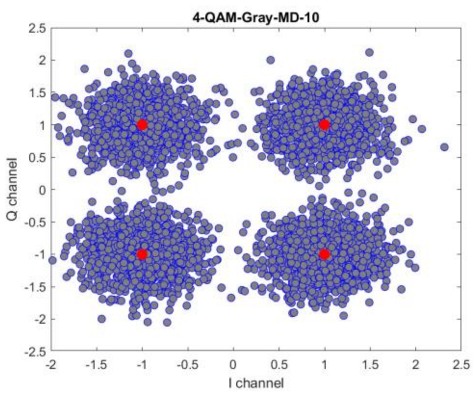


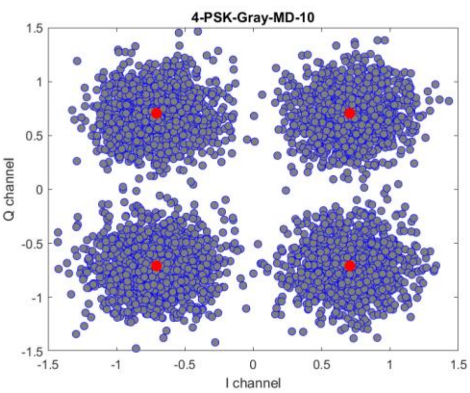


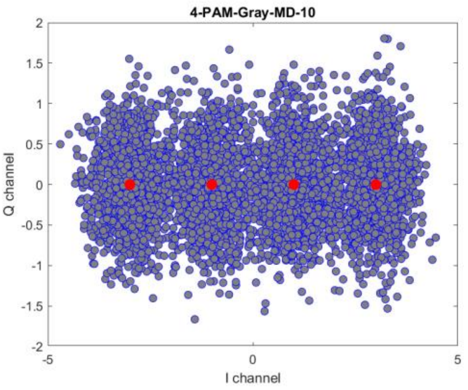


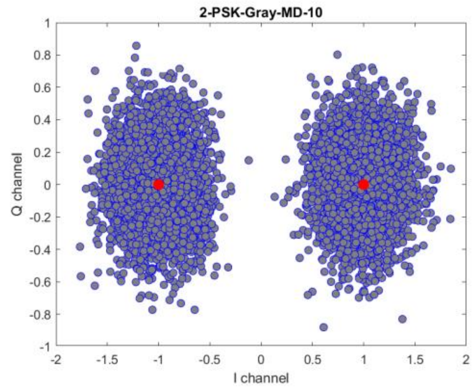


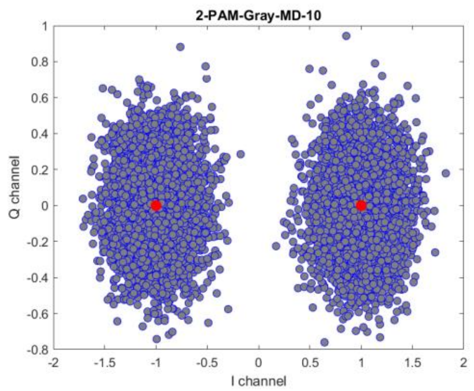
(2) All constellation with gray code and SNR = 10.

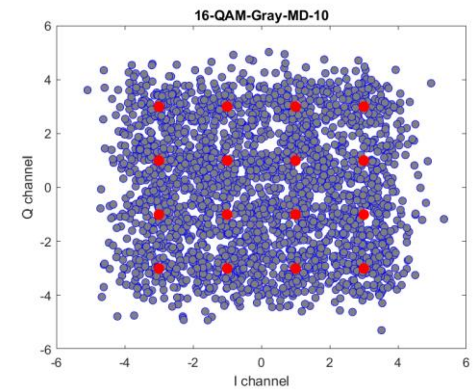


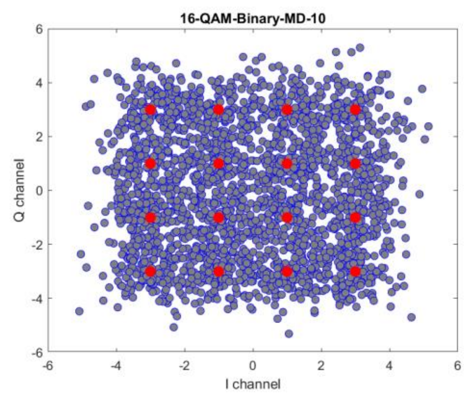


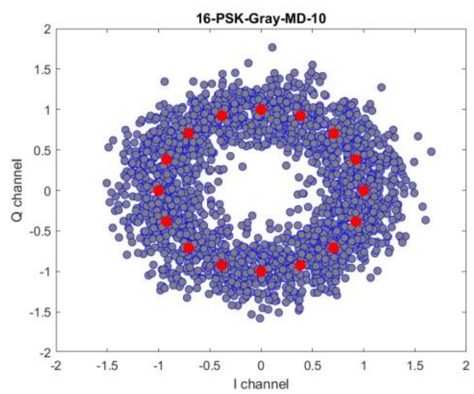


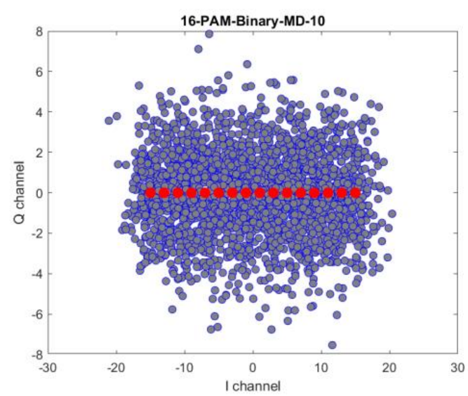


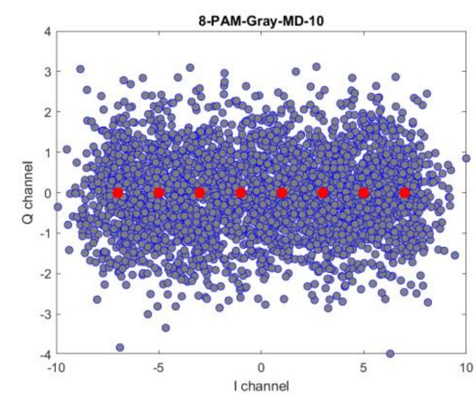




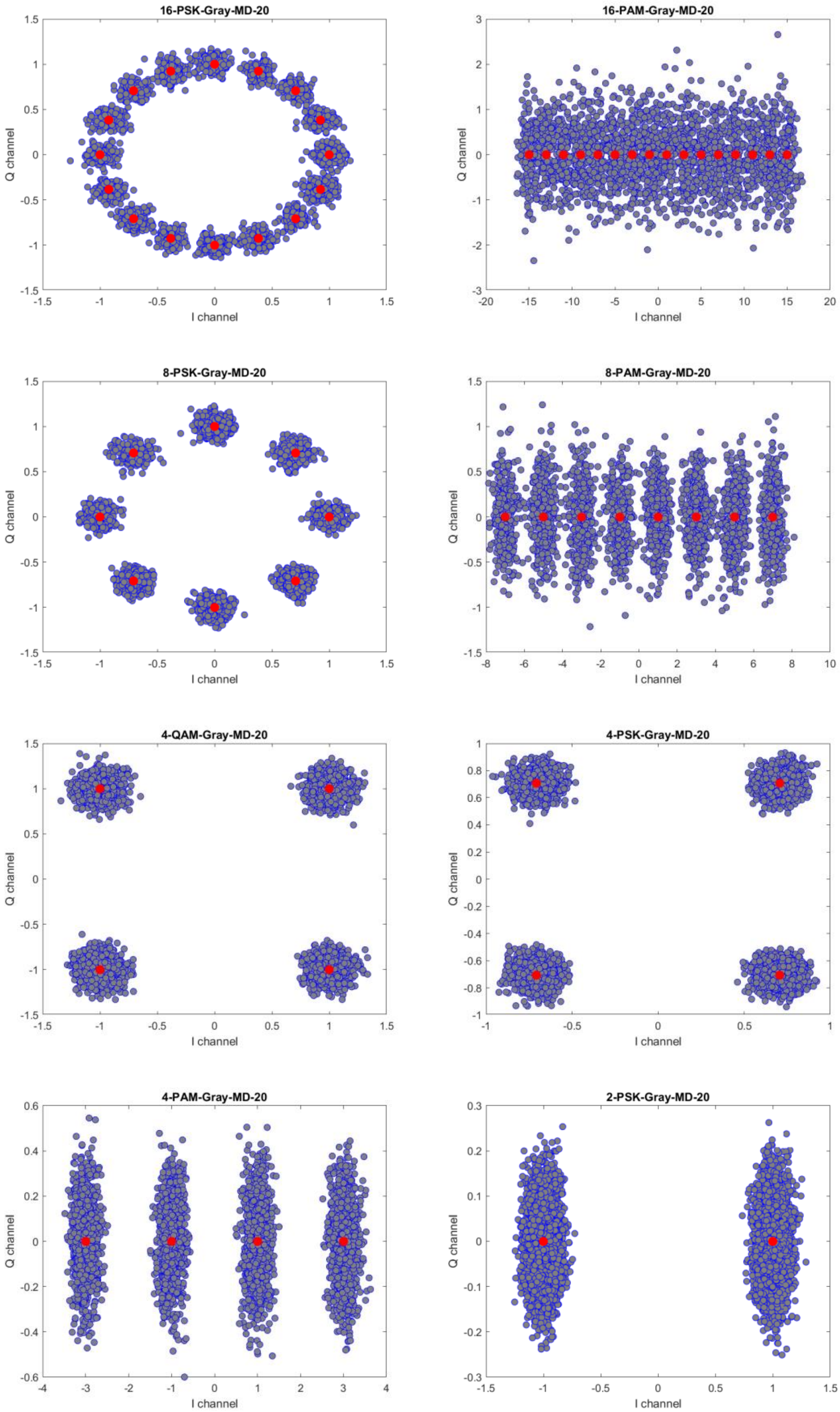


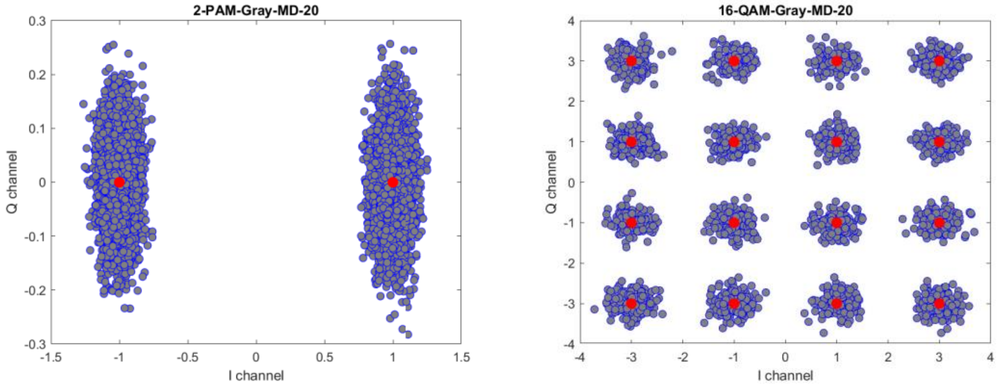






(3) All constellation with gray code and SNR = 20.


16-PSK-Gray-MD-20
16-PAM-Gray-MD-20
8-PSK-Gray-MD-20
8-PAM-Gray-MD-20
4-QAM-Gray-MD-20
4-PSK-Gray-MD-20
4-PAM-Gray-MD-20
2-PSK-Gray-MD-20
I channel
Q channel



(b) Write a Matlab function to demap the symbol sequence to binary sequence. Your demapper function should have the following arguments, binary_sequence = symbol_demapper(symbol_sequence, M, d, ... constellation, mapping, decision_rule) where decision_rule is 'MD'. (Bonus) decision_rule: 'MAP' and 'ML'.

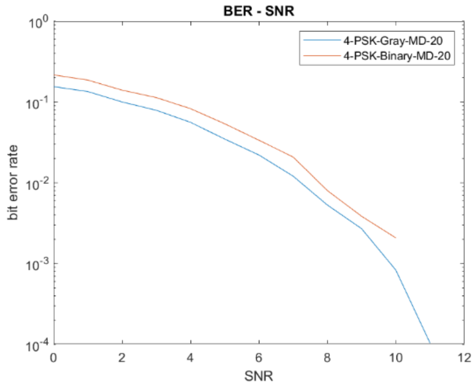
```
function binary_sequence = symbol_demapper(symbol_sequence, M, d,
constellation, mapping, decision_rule)
    % where decision_rule is 'MD'.
    % (Bonus) decision_rule: 'MAP' and 'ML'.

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

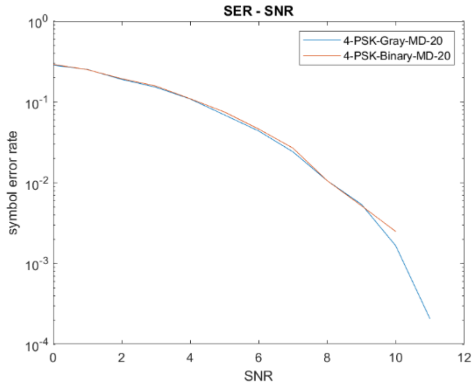
```
            C = cell(length(symbol_sequence),1);
            tit = strcat(num2str(M), '-', constellation, '-', mapping);
            A = {};
            load(strcat('cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            P = ones(1,length(B))/length(B);
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                % abs(B - symbol_sequence(ii))
                [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('cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                % abs(B - symbol_sequence(ii))
                [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
```

(c) First, you should generate a random binary sequence. Next, go through the process, binary seqence → symbol mapping → add circularly symmetric complex Gaussian noise → symbol demapping → detection (MD rule) For each case, **plot constellations**, **symbol error rate**, and **bit error rate** with SNR = 0 ~ 25 dB.

We simulate by sending 9600-bit sequence through SNR from 0, 1, 2 to 25. We found that when SNR is high, the error rate is 0, so we did not plot the error rate 0 on the figure. The constellations of SNR = 0, 10, 20 are plot in problem 4(a), so we only show the BER-SNR and SER-SNR in this problem.

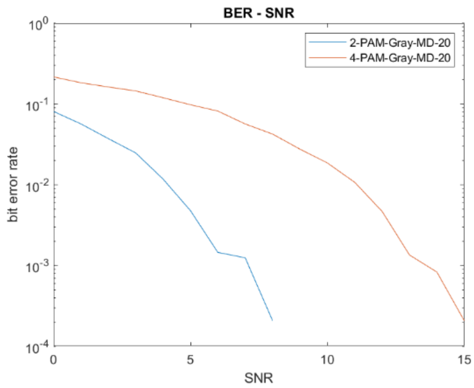
• case 1: QPSK + binary code vs. QPSK + Gray code

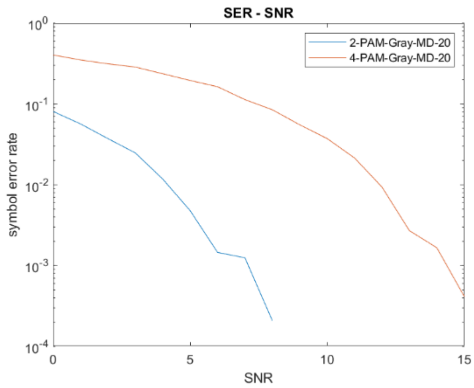




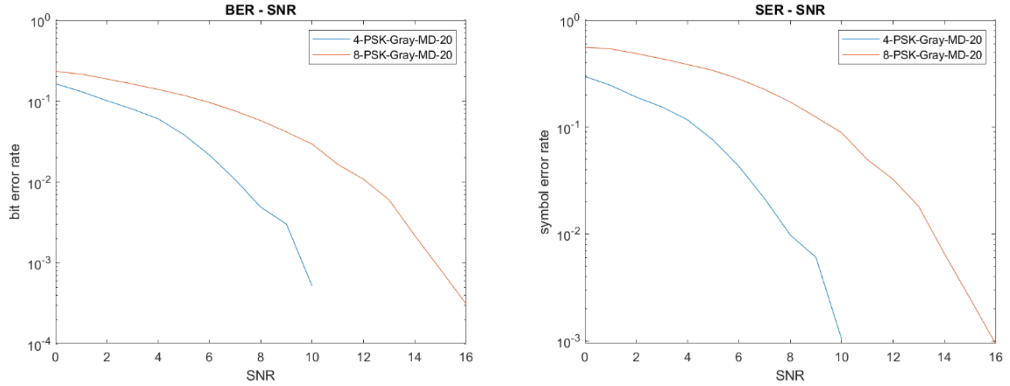
We can see that although the symbol error of binary and gray code is almost the same, gray code tend to have lower bit error rate. This is because in gray code 00 and 11 is in the opposite side of the constellation, symbol error performing by 10 or 01 which causes lower bit error rate to 00 is more likely than by 11.

• case 2: 2-PAM + Gray code vs. 4-PAM Gray code

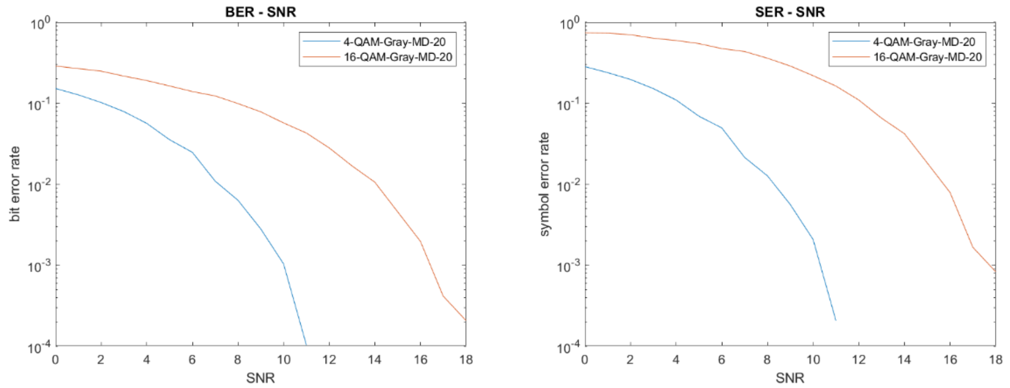




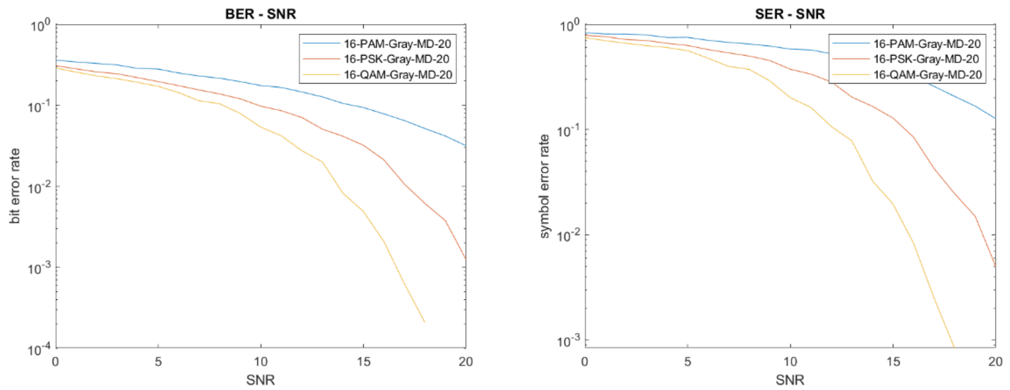
• case 3: QPSK + Gray code vs. 8-PSK + Gray code



• case 4: 4-QAM + Gray code vs. 16-QAM + Gray code



• case 5: 16-PAM + Gray code vs. 16-PSK + Gray code + 16-QAM + Gray code



5. **Modulation and Demodulation**: In Lab 2, we have completed the source coding of 'handel.ogg'. In this part, please pass the output of source coding in Lab 2 to the process mentioned in 4(c). Describe what it sounds like.
Since there might be many bit error without error-correction-code(ECC), so we can’t apply Huffman encoder and decoder to recover the sequence; otherwise, the Huffman decode process won’t run if bit error occurs. So we apply PCM encode and decode.
We pass the signal through the following constellation.
• case 1: QPSK + binary code vs. QPSK + Gray code

These two constellation **sounds the same** under same SNR. When SNR is low, the noise becomes extremely large that we can't hear the original signal.

• case 2: 2-PAM + Gray code vs. 4-PAM Gray code

• case 3: QPSK + Gray code vs. 8-PSK + Gray code

• case 4: 4-QAM + Gray code vs. 16-QAM + Gray code

These three cases have similar results.

Under same SNR, **we can hear** that 4-PAM has more noise than 2-PAM, and 8-PSK and 16-QAM has more noise than QPSK.

• case 5: 16-PAM + Gray code vs. 16-PSK + Gray code + 16-QAM + Gray code

Under same SNR, **we can hear** that 16-PAM is the noisiest than 16-PSK than 16-QAM.

Appendix

1. Code for problem1

```
 %% problem 1
binary_sequence1 = reshape(fliplr(de2bi([0:1]))',  1, []);
binary_sequence2 = reshape(fliplr(de2bi([0:3]))',  1, []);
binary_sequence3 = reshape(fliplr(de2bi([0:7]))',  1, []);
binary_sequence4 = reshape(fliplr(de2bi([0:15]))', 1, []);
d = 1;
symbol_sequence1  = symbol_mapper(binary_sequence1,  2, d, 'PAM',
'Binary');
symbol_sequence2  = symbol_mapper(binary_sequence2,  4, d, 'PAM',
'Binary');
symbol_sequence3  = symbol_mapper(binary_sequence3,  8, d, 'PAM',
'Binary');
symbol_sequence4  = symbol_mapper(binary_sequence4, 16, d, 'PAM',
'Binary');
symbol_sequence5  = symbol_mapper(binary_sequence1,  2, d, 'PSK',
'Binary');
symbol_sequence6  = symbol_mapper(binary_sequence2,  4, d, 'PSK',
'Binary');
symbol_sequence7  = symbol_mapper(binary_sequence3,  8, d, 'PSK',
'Binary');
symbol_sequence8  = symbol_mapper(binary_sequence4, 16, d, 'PSK',
'Binary');
```

```
symbol_sequence9  = symbol_mapper(binary_sequence2,  4, d, 'QAM',
'Binary');
symbol_sequence10 = symbol_mapper(binary_sequence4, 16, d, 'QAM',
'Binary');
symbol_sequence11 = symbol_mapper(binary_sequence1,  2, d, 'PAM',
'Gray');
symbol_sequence12 = symbol_mapper(binary_sequence2,  4, d, 'PAM',
'Gray');
symbol_sequence13 = symbol_mapper(binary_sequence3,  8, d, 'PAM',
'Gray');
symbol_sequence14 = symbol_mapper(binary_sequence4, 16, d, 'PAM',
'Gray');
symbol_sequence15 = symbol_mapper(binary_sequence1,  2, d, 'PSK',
'Gray');
symbol_sequence16 = symbol_mapper(binary_sequence2,  4, d, 'PSK',
'Gray');
symbol_sequence17 = symbol_mapper(binary_sequence3,  8, d, 'PSK',
'Gray');
symbol_sequence18 = symbol_mapper(binary_sequence4, 16, d, 'PSK',
'Gray');
symbol_sequence19 = symbol_mapper(binary_sequence2,  4, d, 'QAM',
'Gray');
symbol_sequence20 = symbol_mapper(binary_sequence4, 16, d, 'QAM',
'Gray');
%% testing error message
symbol_sequence21 = symbol_mapper(binary_sequence4, 14, d, 'QAM',
'Gray');

function symbol_sequence = symbol_mapper(binary_sequence, M, d,
constellation, mapping)
    % - M: The number of points in the signal constellation.
        % M = 2,4,8,16 for 'PAM' & 'PSK'; M = 4,16 for 'QAM'.
    % - d: The minimum distance among the constellation.
    % - constellation: 'PAM', 'PSK' or 'QAM'.
    % - mapping: 'Binary' or 'Gray'.

    % For Gray code mapping of PAM, please assign 00 · · · 0 to the
leftest constellation point.
```

```
    % Gray code mapping of PSK and QAM is shown in Figure 1 and Figure 
2. If input M is
    % not the power of 2 or if input constellation is not defined, your 
function should be able
    % to throw error and display message.
    % Plot all the constellations, and show the bits of each symbol on 
the constellation points
    seq = [];
    mmax = 0;
    switch constellation
        case 'PAM'
            switch M
                case {2, 4, 8, 16}
                    mmax = 2;
                    seq = reshape(binary_sequence,log2(M),[])';
                    switch mapping
                        case 'Binary'
                            seq_len = length(seq);
                            C = cell(seq_len, 1);
                            for ii = 1:seq_len
                                C{ii,1} = (bi2de(fliplr(seq(ii,:)))*2-(M-
1))*d;
                            end
                            symbol_sequence = [C{:}];
                        case 'Gray'
                            A2 = [1 0];
                            A4 = [2 1 3 0];
                            A8 = [5 4 2 3 6 7 1 0];
                            A16 = [0 1 3 2 7 6 4 5 15 14 12 13 8 9 11 10];
                            seq_len = length(seq);
                            C = cell(seq_len, 1);
                            for ii = 1:seq_len
                                idx = bi2de(fliplr(seq(ii,:)));
                                switch M
                                    case 2
                                        idx = A2(idx+1);
                                        C{ii,1} = idx*2-(M-1)*d;
                                    case 4
```

```
                            idx = A4(idx+1);
                            C{ii,1} = idx*2-(M-1)*d;
                        case 8
                            idx = A8(idx+1);
                            C{ii,1} = idx*2-(M-1)*d;
                        case 16
                            idx = A16(idx+1);
                            C{ii,1} = idx*2-(M-1)*d;
                    end
                end
                symbol_sequence = [C{:}];
            otherwise
                error("mapping must be 'Binary' or 'Gray'")
        end
    otherwise
        error("M must be 2, 4, 8, 16")
end
case 'PSK'
    switch M
        case {2, 4, 8, 16}
            mmax = d;
            seq = reshape(binary_sequence,log2(M),[])';
            switch mapping
                case 'Binary'
                    seq_len = length(seq);
                    C = cell(seq_len, 1);
                    for ii = 1:seq_len
                        idx = bi2de(fliplr(seq(ii,:)));
                        if M == 4
                            C{ii,1} = cos(2*pi*idx/M+pi/4) +
i*sin(2*pi*idx/M+pi/4);
                        else
                            C{ii,1} = cos(2*pi*idx/M) +
i*sin(2*pi*idx/M);
                        end
                    end
                    symbol_sequence = [C{:}];
                case 'Gray'
```

```
                    A2 = [1 0];
                    A4 = [2 1 3 0];
                    A8 = [5 4 2 3 6 7 1 0];
                    A16 = [0 1 3 2 7 6 4 5 15 14 12 13 8 9 11 10];
                    seq_len = length(seq);
                    C = cell(seq_len, 1);
                    for ii = 1:seq_len
                        idx = bi2de(fliplr(seq(ii,:)));
                        switch M
                            case 2
                                idx = A2(idx+1);
                                C{ii,1} = cos(2*pi*idx/M) +
i*sin(2*pi*idx/M);
                            case 4
                                idx = A4(idx+1);
                                C{ii,1} = cos(2*pi*idx/M+pi/4) +
i*sin(2*pi*idx/M+pi/4);
                            case 8
                                idx = A8(idx+1);
                                C{ii,1} = cos(2*pi*idx/M) +
i*sin(2*pi*idx/M);
                            case 16
                                idx = A16(idx+1);
                                C{ii,1} = cos(2*pi*idx/M) +
i*sin(2*pi*idx/M);
                        end
                    end
                    symbol_sequence = [C{:}];
                otherwise
                    error("mapping must be 'Binary' or 'Gray'")
            end
        otherwise
            error("M must be 2, 4, 8, 16")
    end
case 'QAM'
    switch M
        case {4, 16}
            mmax = (sqrt(M)-1)*d;
```

```
                    seq = reshape(binary_sequence,log2(M),[])';
                    switch mapping
                        case 'Binary'
                            seq_len = length(seq);
                            C = cell(seq_len, 1);
                            for ii = 1:seq_len
                                mid = log2(M)/2;
                                x_idx = bi2de(fliplr(seq(ii,1:mid)));
                                y_idx = bi2de(fliplr(seq(ii,mid+1:end)));
                                C{ii, 1} = ((x_idx*2-(sqrt(M)-1))*d) +
i*((y_idx*2-(sqrt(M)-1))*d);
                            end
                            symbol_sequence = [C{:}];
                        case 'Gray'
                            A = [0 1 3 2];
                            seq_len = length(seq);
                            C = cell(seq_len, 1);
                            for ii = 1:seq_len
                                mid = log2(M)/2;
                                x_idx = bi2de(fliplr(seq(ii,1:mid)));
                                y_idx = bi2de(fliplr(seq(ii,mid+1:end)));
                                x_idx = A(x_idx+1);
                                y_idx = A(y_idx+1);
                                C{ii, 1} = ((x_idx*2-(sqrt(M)-1))*d) +
i*((y_idx*2-(sqrt(M)-1))*d);
                            end
                            symbol_sequence = [C{:}];
                        otherwise
                            error("mapping must be 'Binary' or 'Gray'")
                    end
                otherwise
                    error("M must be 4, 16")
            end
        otherwise
            error("constellation must be 'PAM', 'PSK' or 'QAM'")
    end
    %% plot
    figure;
```

```
    seq_len = length(symbol_sequence);
    xx = roundn(real(symbol_sequence), -6);
    yy = roundn(imag(symbol_sequence), -6);
    plot(xx, yy, 'o', 'MarkerSize', 6, 'MarkerEdgeColor','b',
'MarkerFaceColor',[0.5,0.5,0.5]);
    xlabel('I channel');
    ylabel('Q channel');
    tit = strcat(num2str(M), '-', constellation, '-', mapping);
    A = cell(M,1);
    for ii = 1:M
        A{ii,1} = symbol_sequence(1,ii);
    end
    save(strcat('cell_prob1/', tit,'.mat'), 'A');
    title(tit);
    for ii = 1:seq_len
        text(xx(ii), yy(ii)+mmax/10, char(seq(ii,:)+48), 'Color', 'red',
'FontSize', 8, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
    end
    saveas(gcf, strcat('image_prob1/', tit,'.png'));
end
```

2. Code for problem2 & 3

```
clear all;
close all;
oversampling_factor = 1000;
T_os = 1/oversampling_factor; % time spacing after oversampling
W = 50;
T = 1/(2*W);
pulse_duration = 1; % 1 sec
t_axis = (-pulse_duration/2 : T_os : pulse_duration/2 - T_os);
X1 = zeros(1, length(t_axis));
X1(1,(length(t_axis)/2-T/T_os):(length(t_axis)/2+T/T_os)) = 1;
X2 = sin(pi*t_axis/T)./(pi*t_axis/T); X2(1, find(isnan(X2))) = 1;
beta = 0.7;
X3 = X2.*cos(pi*beta*t_axis/T)./(1-4*beta^2*t_axis.^2/T^2); X3(1,
find(isnan(X3))) = pi/4*sin(pi/2/beta)/(pi/2/beta);
```

```
% X4 = ( sin(pi*t_axis*(1-beta)/T) +
4*beta*t_axis/T.*cos(pi*t_axis/T*(1+beta)) ) ./ ( pi*t_axis/T.*(1-
(4*beta*t_axis/T).^2) ) / sqrt(T);
% X4(1, find(isnan(X4))) = 1/sqrt(T)*(1+beta*(4/pi-1));
% X5 = X2 / sqrt(T);

[Wt, A] = CTFT(X1, oversampling_factor, t_axis, 'Rect');
[Wt, A] = CTFT(X2, oversampling_factor, t_axis, 'Sinc');
[Wt, A] = CTFT(X3, oversampling_factor, t_axis, 'Raised Cosine');

% [Wt, A] = CTFT(X4, oversampling_factor, t_axis, 'Root raised
Cosine');
% [Wt, A] = CTFT(X5, oversampling_factor, t_axis, 'RSinc');
% X6 = conv(X5, X5)*T_os;
% [Wt, A] = CTFT(X6, oversampling_factor, (-
pulse_duration+T_os:T_os:pulse_duration-T_os), 'SRSinc' );
% X7 = conv(X4, X4)*T_os;
% [Wt, A] = CTFT(X7, oversampling_factor, (-
pulse_duration+T_os:T_os:pulse_duration-T_os), 'SRRaise' );

d = 2;
rng(19);
code = randi([0 1], 1, 20);
symbol_sequence = symbol_mapper_QPSK_gray(code, d);
Iphase = real(symbol_sequence);
Qphase = imag(symbol_sequence);

yI = pulse_shaper(Iphase, 'raised cosine');
t_axis2 = (0:T_os:length(yI)*T_os-T_os);
CTFT(yI, oversampling_factor, t_axis2, 'YI');

yQ = pulse_shaper(Qphase, 'raised cosine');
CTFT(yQ, oversampling_factor, t_axis2, 'YQ');

fc = 150;
t_axis2 = (0:T_os:length(yI)*T_os-T_os);
y = yI*sqrt(2).*cos(2*pi*fc*t_axis2) -
yQ*sqrt(2).*sin(2*pi*fc*t_axis2);
```

```
CTFT(y, oversampling_factor, t_axis2, 'Y');

snrr = 25;
pd = makedist('Normal');
y = y +
random(pd,1,length(y))*sqrt(sum(y.^2)/(10^(snrr/10))/length(y));
CTFT(y, oversampling_factor, t_axis2, 'Y + noise');
h = fir1(W, 0.9, 'low');

yIt = y*sqrt(2).*cos(2*pi*fc*t_axis2);
yIt = filter(h, 1, yIt);
CTFT(yIt, oversampling_factor, t_axis2, 'YI rcv');

yQt = -y*sqrt(2).*sin(2*pi*fc*t_axis2);
yQt = filter(h, 1, yQt);
CTFT(yQt, oversampling_factor, t_axis2, 'YQ rcv');

I_rcv = filter_sampling(yIt);
Q_rcv = filter_sampling(yQt);

function [W, A] = CTFT(x, fs, t_axis, tit)
    [col, row] = size(x);
    y = fft(x, row);

    N = row;
    w = 2*pi*(0:(N-1)) / N;
    w2 = fftshift(w);
    w3 = unwrap(w2 - 2*pi);

    y = y / fs;
    W = w3/pi*fs/2;
    A = abs(fftshift(y));
    figure;
    subplot(3,1,1);
    plot(t_axis, x);
    title(tit);

    subplot(3,1,2);
```

```
    plot(w3/pi*fs/2, abs(fftshift(y)));
    xlim([-500, 500]);
    title('CTFT Amplitude');
    xlabel('Freq (Hz)');
    ylabel('Amplitude');

    subplot(3,1,3);
    plot(w3/pi*fs/2, angle(fftshift(y)));
    xlim([-500, 500]);
    title('CTFT Phase');
    xlabel('Freq (Hz)');
    ylabel('Phase');
end

function y = pulse_shaper(x, pulse_shape, W)
    oversampling_factor = 1000;
    T_os = 1/oversampling_factor; % time spacing after oversampling
    W = 50;
    T = 1/(2*W);
    pulse_duration = 1; % 1 sec
    t_axis = (-pulse_duration/2 : T_os : pulse_duration/2 - T_os);
    Rect = zeros(1, length(t_axis));
    Rect(1,(length(t_axis)/2-T/T_os):(length(t_axis)/2+T/T_os)) = 1;
    RSinc = sin(pi*t_axis/T)./(pi*t_axis/T) / sqrt(T);
    RSinc(1, find(isnan(RSinc))) = 1 / sqrt(T);
    beta = 0;
    RRaised = ( sin(pi*t_axis*(1-beta)/T) +
4*beta*t_axis/T.*cos(pi*t_axis/T*(1+beta)) ) ./ ( pi*t_axis/T.*(1-
(4*beta*t_axis/T).^2) ) / sqrt(T);
    RRaised(1, find(isnan(RRaised))) = 1/sqrt(T)*(1+beta*(4/pi-1));

    switch pulse_shape
    case 'sinc'
        Pul = RSinc;
    case 'raised cosine'
        Pul = RRaised;
    otherwise
```

```
        error("Only 'sinc' and 'raised cosine' are valid for pulse 
shape.")
    end

    save('Pulse.mat', 'Pul');
    O = zeros(1, (T/T_os)*(length(x)-1)+length(Pul));
    O(1:length(Pul)) = Pul;
    y = zeros(1, length(O));
    for ii = 1 : length(x)
        y = y + x(1,ii)*circshift(O,(T/T_os)*(ii-1));
    end

end

function symbol_sequence = symbol_mapper_QPSK_gray(binary_sequence, d)
    seq = [];
    M = 4;
    seq = reshape(binary_sequence,log2(M),[])';
    A4 = [2 1 3 0];
    seq_len = length(seq);
    C = cell(seq_len, 1);
    for ii = 1:seq_len
        idx = bi2de(fliplr(seq(ii,:)));
        idx = A4(idx+1);
        C{ii,1} = cos(2*pi*idx/M+pi/4) + i*sin(2*pi*idx/M+pi/4);
    end
    symbol_sequence = [C{:}];
end

function y = filter_sampling(in)
    oversampling_factor = 100000;
    T_os = 1/oversampling_factor; % time spacing after oversampling
    C = {};
    A = load('Pulse.mat');
    Pul = A.Pul;
    RE = conv(in, Pul)*T_os;
    ii = 1;
    while ii*length(Pul) <= length(RE)
```

```
        C{ii} = RE(1,ii*length(Pul));
        ii = ii + 1;
    end
    y = [C{:}];
end
```

3. Code for problem4

```
%% problem 4
close all;
clear all;
binary_sequence = reshape(fliplr(de2bi(randi(16, 1, 2400)-1,4))',  1,
[]);
d = 1;
berr_all = {};
serr_all = {};
for SNR=0:1:20
    cnt = 0;
    tit = cell(1,50);
    berr = cell(1,50);
    serr = cell(1,50);
    % [tit{1}, berr{1}, serr{1}] = Run(binary_sequence,  2, d, 'PAM',
'Gray', 'MD', SNR);
    % [tit{2}, berr{2}, serr{2}] = Run(binary_sequence,  4, d, 'PAM',
'Gray', 'MD', SNR);
    % [tit{3}, berr{3}, serr{3}] = Run(binary_sequence,  8, d, 'PAM',
'Gray', 'MD', SNR);
    % [tit{4}, berr{4}, serr{4}] = Run(binary_sequence, 16, d, 'PAM',
'Gray', 'MD', SNR);
    % [tit{5}, berr{5}, serr{5}] = Run(binary_sequence,  2, d, 'PSK',
'Gray', 'MD', SNR);
    % [tit{6}, berr{6}, serr{6}] = Run(binary_sequence,  4, d, 'PSK',
'Gray', 'MD', SNR);
    % [tit{7}, berr{7}, serr{7}] = Run(binary_sequence,  8, d, 'PSK',
'Gray', 'MD', SNR);
    % [tit{8}, berr{8}, serr{8}] = Run(binary_sequence, 16, d, 'PSK',
'Gray', 'MD', SNR);
    [tit{9}, berr{9}, serr{9}] = Run(binary_sequence,  4, d, 'QAM',
'Gray', 'MD', SNR);
```

```
    [tit{10}, berr{10}, serr{10}] = Run(binary_sequence, 16, d, 'QAM',
'Gray', 'MD', SNR);
    % [tit{11}, berr{11}, serr{11}] = Run(binary_sequence,  2, d, 'PAM',
'Binary', 'MD', SNR);
    % [tit{12}, berr{12}, serr{12}] = Run(binary_sequence,  4, d, 'PAM',
'Binary', 'MD', SNR);
    % [tit{13}, berr{13}, serr{13}] = Run(binary_sequence,  8, d, 'PAM',
'Binary', 'MD', SNR);
    % [tit{14}, berr{14}, serr{14}] = Run(binary_sequence, 16, d, 'PAM',
'Binary', 'MD', SNR);
    % [tit{15}, berr{15}, serr{15}] = Run(binary_sequence,  2, d, 'PSK',
'Binary', 'MD', SNR);
    % [tit{16}, berr{16}, serr{16}] = Run(binary_sequence,  4, d, 'PSK',
'Binary', 'MD', SNR);
    % [tit{17}, berr{17}, serr{17}] = Run(binary_sequence,  8, d, 'PSK',
'Binary', 'MD', SNR);
    % [tit{18}, berr{18}, serr{18}] = Run(binary_sequence, 16, d, 'PSK',
'Binary', 'MD', SNR);
    % [tit{19}, berr{19}, serr{19}] = Run(binary_sequence,  4, d, 'QAM',
'Binary', 'MD', SNR);
    % [tit{20}, berr{20}, serr{20}] = Run(binary_sequence, 16, d, 'QAM',
'Binary', 'MD', SNR);
    % [tit{21}, berr{21}, serr{21}] = Run(binary_sequence, 16, d, 'PAM',
'Gray', 'MAP', SNR);
    % [tit{22}, berr{22}, serr{22}] = Run(binary_sequence, 16, d, 'PSK',
'Gray', 'MAP', SNR);
    % [tit{23}, berr{23}, serr{23}] = Run(binary_sequence, 16, d, 'QAM',
'Gray', 'MAP', SNR);
    % [tit{24}, berr{24}, serr{24}] = Run(binary_sequence, 16, d, 'PAM',
'Gray', 'ML', SNR);
    % [tit{25}, berr{25}, serr{25}] = Run(binary_sequence, 16, d, 'PSK',
'Gray', 'ML', SNR);
    % [tit{26}, berr{26}, serr{26}] = Run(binary_sequence, 16, d, 'QAM',
'Gray', 'ML', SNR);

    % figure;
    berr = [berr{:}];
    serr = [serr{:}];
```

```
    berr_all{SNR+1} = berr;
    serr_all{SNR+1} = serr;
    tit = tit(~cellfun('isempty',tit));
    % C = categorical(tit);
    % h = histogram('Categories', C, 'BinCounts', berr, 'FaceColor',
'b', 'FaceAlpha', 1);
    % title(strcat('SNR = ',{' '},num2str(SNR),'dB'));
    % ylabel('Bit error rate');
    % xlabel('constellation & mapping');
end
n = length(tit);
A = reshape([berr_all{1,:}],n,[]);
figure;
plot([0:1:20], A(1,:));
xlabel('SNR');
ylabel('bit error rate');
title('BER - SNR')
set(gca, 'YScale', 'log')
hold on;
for ii = 2:n
    plot([0:1:20], A(ii,:));
end
legend(tit,'Location','northeast')
hold off;
saveas(gcf, strcat('image_prob4/SNRplot/', [tit{:}], '_BER','.png'));

A = reshape([serr_all{1,:}],n,[]);
figure;
plot([0:1:20], A(1,:));
xlabel('SNR');
ylabel('symbol error rate');
title('SER - SNR')
set(gca, 'YScale', 'log')
hold on;
for ii = 2:n
    plot([0:1:20], A(ii,:));
end
legend(tit,'Location','northeast')
```

```
hold off;
saveas(gcf, strcat('image_prob4/SNRplot/', [tit{:}],'_SER','.png'));


function [tit, berr, serr] = Run(binary_sequence, M, d, constellation,
mapping, decision_rule, SNR)
    tit = strcat(num2str(M), '-', constellation, '-', mapping);
    A = {};
    load(strcat('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;
    binary_sequence_in_symbol = bi2de(reshape(binary_sequence, log2(M),
[])');
    binary_sequence_rec_in_symbol = bi2de(reshape(binary_sequence_rec,
log2(M), [])');
    serr = sum(binary_sequence_in_symbol ~=
binary_sequence_rec_in_symbol) / ( binary_seq_len/log2(M) );
    fprintf("%16s Bit error rate   %f\n", tit, berr);
    fprintf("%16s Symbol error rate   %f\n", tit, serr);
    tit = strcat(tit, '-', decision_rule, '-', int2str(SNR));

    % figure;
    % seq_len = length(symbol_sequence);
    % xx = roundn(real(symbol_sequence_rec), -6);
    % yy = roundn(imag(symbol_sequence_rec), -6);
    % plot(xx, yy, 'o', 'MarkerSize', 6, 'MarkerEdgeColor','b',
'MarkerFaceColor',[0.5,0.5,0.5]);
```

```
    % hold on;
    % xx = roundn(real(symbol_sequence), -6);
    % yy = roundn(imag(symbol_sequence), -6);
    % plot(xx, yy, 'o', 'MarkerSize', 8, 'MarkerEdgeColor','r',
'MarkerFaceColor',[1,0,0]);
    % xlabel('I channel');
    % ylabel('Q channel');
    % A = cell(M,1);
    % for ii = 1:M
    %     A{ii,1} = symbol_sequence(1,ii);
    % end
    % title(tit);
    % saveas(gcf, strcat('image_prob4/', tit,'.png'));
    % hold off;
end

function binary_sequence = symbol_demapper(symbol_sequence, M, d,
constellation, mapping, decision_rule)
    % where decision_rule is 'MD'.
    % (Bonus) decision_rule: 'MAP' and 'ML'.

    switch decision_rule
        case 'MD'
            C = cell(length(symbol_sequence),1);
            tit = strcat(num2str(M), '-', constellation, '-', mapping);
            A = {};
            load(strcat('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('cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            P = ones(1,length(B))/length(B);
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                % abs(B - symbol_sequence(ii))
                [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('cell_prob1/', tit,'.mat'), 'A');
            B = [A{:}];
            sym_len = length(symbol_sequence);
            for ii = 1:sym_len
                % abs(B - symbol_sequence(ii))
                [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('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
```

4. Code for problem5

```
%% problem 2

% assume amp is in [-1, 1]
clear all;
[x, fs] = audioread('handel.ogg');
xmax = 1;
bit = 4;
level = 2^bit;
xt = quantizer_L_level(x, xmax, level)';
y = pcm_enc(xt, 4);
% length_y = length(y);
num1 = 16;
num2 = 16;
num3 = 16;
constellation1 = 'PAM';
```

```matlab
constellation2 = 'PSK';
constellation3 = 'QAM';
mapping1 = 'gray';
mapping2 = 'gray';
mapping3 = 'gray';
y_rec2 = Run(y, num1, 1, constellation1, mapping1, 'MD', 20);
y_rec3 = Run(y, num1, 1, constellation1, mapping1, 'MD', 10);
y_rec4 = Run(y, num1, 1, constellation1, mapping1, 'MD', 0);
y_rec6 = Run(y, num2, 1, constellation2, mapping2, 'MD', 20);
y_rec7 = Run(y, num2, 1, constellation2, mapping2, 'MD', 10);
y_rec8 = Run(y, num2, 1, constellation2, mapping2, 'MD', 0);
y_rec10 = Run(y, num3, 1, constellation3, mapping3, 'MD', 20);
y_rec11 = Run(y, num3, 1, constellation3, mapping3, 'MD', 10);
y_rec12 = Run(y, num3, 1, constellation3, mapping3, 'MD', 0);

x_rec2 = pcm_dec(y_rec2, 4);
x_rec3 = pcm_dec(y_rec3, 4);
x_rec4 = pcm_dec(y_rec4, 4);
x_rec6 = pcm_dec(y_rec6, 4);
x_rec7 = pcm_dec(y_rec7, 4);
x_rec8 = pcm_dec(y_rec8, 4);
x_rec10 = pcm_dec(y_rec10, 4);
x_rec11 = pcm_dec(y_rec11, 4);
x_rec12 = pcm_dec(y_rec12, 4);

% 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 = huffmanenco(xt, dict);
% length_y = length(y)
% y_rec = Run(y, 16, 1, 'QAM', 'Gray', 'MD', 10);
% xd = huffmandeco(y_rec, dict);

sound(x_rec1, fs);

% clear all;
```

```
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('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('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('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('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('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 y = pcm_enc(x, numBits)
    level = 2^numBits;
    delta = 2 / level;
    x = (x + (level-1)*delta/2) / delta + 1;
    y = reshape(fliplr(de2bi(x))',1,[]);
end
```

```
function x = pcm_dec(y, numBits)
    level = 2^numBits;
    delta = 2 / level;
    x = bi2de(fliplr(reshape(y, 4, [])'));
    x = (x - 1)*delta - (level-1)*delta/2;
end
```