Simulating Different Receivers in a Rayleigh Fading, SISO Environment

Project #1

Intelligent Communication Systems (ICS) Lab.
노용재

Winter Intern Seminar (2023-1)

Contents

Binary Modulation, 4-QAM, 16-QAM의 modulation 환경에서 Zero-forcing (ZF) receiver, Minimum Mean Square Error (MMSE) receiver, Maximum Likelihood Detection (MLD)의 BER (bit error rate)를 구하기 위해 실험을 MATLAB에서 수행하였다. 실험의 공통된 조건은 다음과 같다.

- SISO; Single Input, Single Output
- Rayleigh Fading
- Es/N0는 -2dB∼20dB(2dB 간격)
- 신호의 평균전력은 1W이 되도록한다. ($Es = 1$)

1 Implementation

1.1 Common Environment Variables

$$y = hs + n \tag{1}$$

```
EsN0_dB = -2:2:20;
EsN0 = db2pow(EsN0_dB);

EbN0 = EsN0 / log2(M);
EbN0_dB = pow2db(EbN0);

NumberOfSignals = 1;
LengthBitSequence = NumberOfSignals*log2(M); % log2(M) bits per signal

% Bit Generation
BitSequence = randi([0 1], 1, LengthBitSequence);
SymbolSequence = qammod(BitSequence.', M, 'InputType', 'bit', 'UnitAveragePower', 1).';

% Noise (n) Generation
NoiseSequence = (randn(1, length(SymbolSequence)) + 1j * randn(1, length(SymbolSequence))) / sqrt(2);

% Channel (h) Generation
H = (randn(1, length(SymbolSequence)) + 1j * randn(1, length(SymbolSequence))) ./ sqrt(2);

% Received Signal (y = s + n) Generation
ReceivedSymbolSequence = H .* SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0));
```

1.2 ZF(Zero-forcing)

$$z = wy : w_{ZF} = (h)^{-1} \tag{2}$$

$$\hat{s} = \underset{s}{\operatorname{argmin}} |z - s|^2 \tag{3}$$

```
w_zf = H.^(-1);
DetectionSymbolSequence_ZF = ReceivedSymbolSequence .* w_zf; % Detection (Zero-Forcing: y / h)

DetectionBitSequence_ZF = qamdemod(DetectionSymbolSequence_ZF.', M, 'OutputType', 'bit', 'UnitAveragePower', 1)';
```

1.3 MMSE(Minimum Mean Square Error)

$$z = wy : w_{MMSE} = (|h|^2 + 1/\rho)^{-1} h^* \tag{4}$$

$$\hat{s} = \underset{s}{\operatorname{argmin}} |z - s|^2 \tag{5}$$

```
w_mmse = (abs(H).^2+1/EsN0(indx_EbN0)).^(-1) .* conj(H);
DetectionSymbolSequence_MMSE = ReceivedSymbolSequence .* w_mmse;
DetectionBitSequence_MMSE = qamdemod(DetectionSymbolSequence_MMSE.', M, 'OutputType', 'bit', 'UnitAveragePower', 1)';
```

1.4 MLD(Maximum Likelihood Detection)

$$\hat{s} = \underset{s}{\operatorname{argmin}} |y - hs|^2 \tag{6}$$

```
alphabet = qammod([0:M-1], M, 'UnitAveragePower', true);
arg = (ones(length(alphabet),1) * ReceivedSymbolSequence) - (alphabet.' * H);
[val,idx] = min(abs(arg).^2;);
DetectionBitSequence_MLD = reshape(de2bi(idx-1, log2(M), 'left-msb')', 1, []);
```

arg는 $(y - hs)$를 나타낸 matrix이다. arg의 $length(alphabet) \times length(SymbolSequence)$이다. $(M \times l)$

$$ReceivedSymbolSequence = \{d_1, d_2, ..., d_l\} \tag{7}$$

$$alphabet = \{a_1, a_2, ..., a_M\} \tag{8}$$

$$H = \{h_1, h_2, ..., h_l\} \tag{9}$$

$$\tag{10}$$

$$
\begin{aligned}
arg &= \begin{bmatrix} d_1 & d_2 & ... & d_{l-1} & d_l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_1 & d_2 & ... & d_{l-1} & d_l \end{bmatrix} - \begin{bmatrix} a_1 h_1 & a_1 h_2 & ... & a_1 h_{l-1} & a_1 h_l \\ a_2 h_1 & a_2 h_2 & ... & a_2 h_{l-1} & a_2 h_l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{M-1} h_1 & a_{M-1} h_2 & ... & a_{M-1} h_{l-1} & a_{M-1} h_l \\ a_M h_1 & a_1 h_2 & ... & a_{M-1} h_{l-1} & a_M h_l \end{bmatrix} \\
&= \begin{bmatrix} d_1 - a_1 h_1 & d_2 - a_1 h_2 & ... & d_{l-1} - a_1 h_{l-1} & d_l - a_1 h_l \\ d_1 - a_2 h_1 & d_2 - a_2 h_2 & ... & d_{l-1} - a_2 h_{l-1} & d_l - a_2 h_l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_1 - a_{M-1} h_1 & d_2 - a_{M-1} h_2 & ... & d_{l-1} - a_{M-1} h_{l-1} & d_l - a_{M-1} h_l \\ d_1 - a_M h_1 & d_2 - a_1 h_2 & ... & d_{l-1} - a_{M-1} h_{l-1} & d_l - a_M h_l \end{bmatrix}
\end{aligned} \tag{11}
$$

2 결과 및 분석

2.1 Simulation Result

이론값은 *Matlab*의 *berfading* 함수를 통해 얻어졌다.

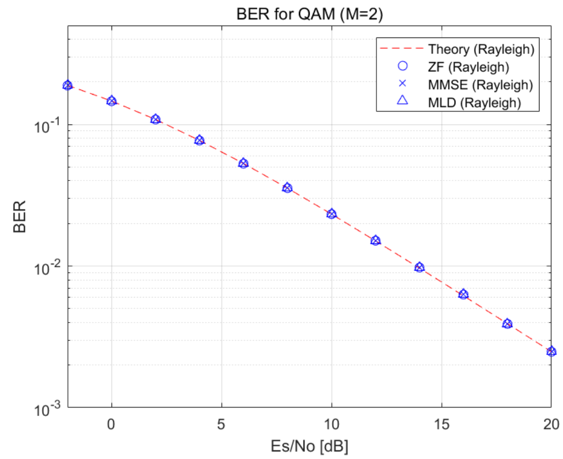
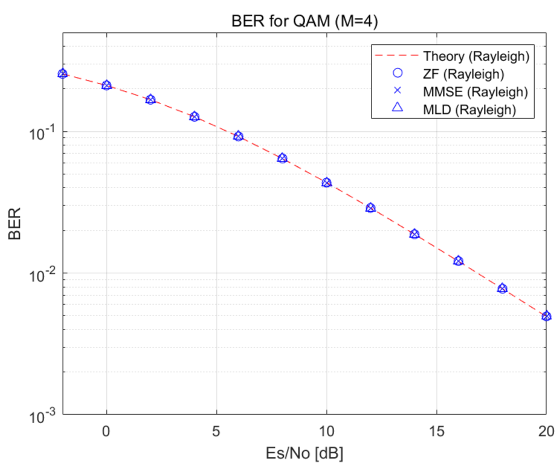
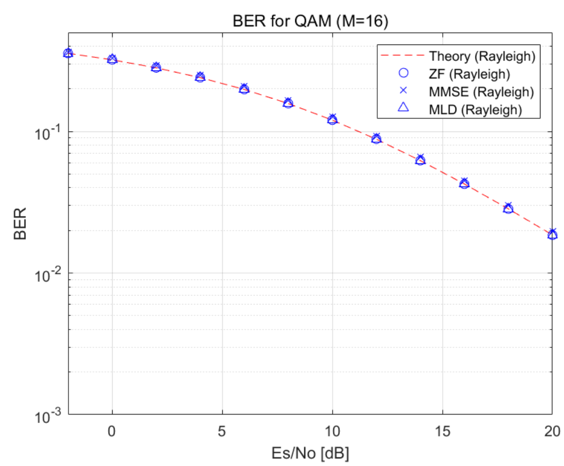


Figure 1: Binary Modulation



(a) 4-QAM



(b) 16-QAM

Figure 2: QAM

2.2 Normalization Factor

코드 내에서 직접적으로 쓰이지는 않았지만 생각해볼 만한 부분은 *Normalization Factor*이다. 이 *Normalization Factor*를 사용하여 평균 전력이 1W가 되게끔 할 수 있다.

Code1

```
alphabet = qammod([0:M-1], M, 'UnitAveragePower', true);
```

Code 2

```
Normalization_Factor = sqrt(2/3*(M-1));
```

```
alphabet = qammod([0:M-1], M) / Normalization_Factor;
```

*Code 1*과 *Code 2*는 동일한 결과를 이룬다.

$M = 2^{2n}$ $(n = 1, 2, 3, ...)$일 때의 *Normalization Factor*를 일반화 시켜보겠다.
일반적인 QAM의 Constellation Diagram을 살펴보면 실수 $\sqrt{M}$개, 허수 $\sqrt{M}$개의 point를 갖는 것을 알 수 있다.
하나의 신호에 대한 값을 그 신호의 *alphabet*이라고하자. M개의 *alphabet*이 다음과 같다고하자.

$$alphabet = \pm(2n-1) \pm j \cdot (2n-1) \qquad n \in 1, 2, ..., \frac{\sqrt{M}}{2} \tag{12}$$

그렇다면 신호의 평균전력은 다음과 같이 일반화 가능하다.

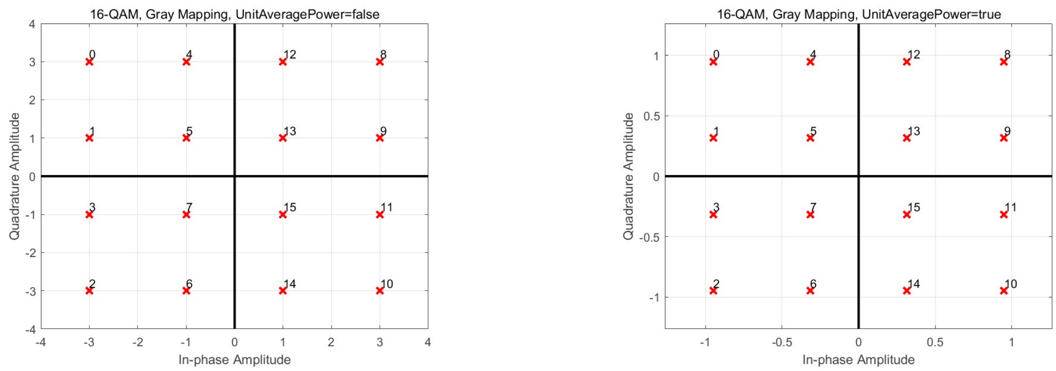
$$\begin{aligned} E_s = E[|s|^2] &= \frac{1}{M}\sum_{n=1}^{M}|s_n|^2 \\ &= \frac{1}{M} \cdot 4 \sum_{n=1}^{\frac{\sqrt{M}}{2}} \sum_{m=1}^{\frac{\sqrt{M}}{2}} [(2n-1)^2 + (2m-1)^2] \\ &= \frac{1}{M} \cdot 4 \sum_{n=1}^{\frac{\sqrt{M}}{2}} \sum_{m=1}^{\frac{\sqrt{M}}{2}} [(2n-1)^2] + \sum_{n=1}^{\frac{\sqrt{M}}{2}} \sum_{m=1}^{\frac{\sqrt{M}}{2}} [(2m-1)]^2 \\ &= \frac{1}{M} \cdot 4 \sum_{n=1}^{\frac{\sqrt{M}}{2}} \sum_{m=1}^{\frac{\sqrt{M}}{2}} [(2n-1)^2] \cdot 2 \\ &= \frac{1}{M} \cdot 4 \sum_{n=1}^{\frac{\sqrt{M}}{2}} [(2n-1)^2 \cdot \sqrt{M}] \\ &= \frac{4}{\sqrt{M}} \sum_{n=1}^{\frac{\sqrt{M}}{2}} [4n^2 - 4n + 1] \\ &= \frac{2}{3}(M-1) \end{aligned} \tag{13}$$

(13)에서의 결과를 토대로 normalization이 이뤄진 alphabet을 구할 수 있다.

$$normalized\ alphabet = \Big[\pm \frac{2n-1}{\sqrt{\frac{2}{3}(M-1)}} \pm j \cdot \frac{2n-1}{\sqrt{\frac{2}{3}(M-1)}}\Big] \qquad n \in \{1, 2, ..., \sqrt{M}\} \tag{14}$$

해당 결과를 토대로 다시 평균 전력을 구한다면 E_s가 1W임을 확인할 수 있다.

참고자료
다음은 16-QAM의 Constellation이다.



(a) Non-normalized Constellation (b) Normalized Constellation

Figure 3: 16-QAM Constellation

2.3 Binary Modulation, 4-QAM에서 ZF, MMSE간 BER의 동일성

실험 결과에 따르면, **Binary Modulation**일 때와 **4-QAM**일 때 ZF 혹은 MMSE 방식을 사용한지와 무관하게 BER이 동일한 것을 관찰할 수 있었다.
ZF와 MMSE에 해당하는 조건식들은 다음과 같다.

$$\hat{s} = \underset{s}{\operatorname{argmin}} |z - s|^2 \tag{15}$$

$$z = w(hs + n) \tag{16}$$

$$w_{ZF} = (h)^{-1} \tag{17}$$

$$w_{MMSE} = (|h|^2 + 1/\rho)^{-1} h^* \tag{18}$$

z는 post-processing signal에 해당된다. ZF와 MMSE, 각각의 post-processing signal을 살펴보겠다.

ZF의 Post-processing Signal

$$\begin{aligned} z_{ZF} &= w_{ZF}(hs + n) \\ &= h^{(} - 1)(hs + n) \\ &= s + \frac{n}{h} \end{aligned} \tag{19}$$

MMSE의 Post-processing Signal

$$\begin{aligned} z_{MMSE} &= w_{MMSE}(hs + n) \\ &= (|h|^2 + 1/\rho)^{-1} h^* (hs + n) \\ &= (|h|^2 + 1/\rho)^{-1} h^* \, h(s + \frac{n}{h}) \end{aligned} \tag{20}$$

위의 결과를 토대로 다음과 같이 표현이 가능하다.

$$\begin{aligned} z_{MMSE} &= (|h|^2 + 1/\rho)^{-1} h^* \, h \cdot z_{ZF} \\ \operatorname{sgn}(z_{MMSE}) &= \operatorname{sgn}((|h|^2 + 1/\rho)^{-1} h^* \, h \cdot z_{ZF}) \\ &= \operatorname{sgn}(z_{ZF}) \quad (\because (|h|^2 + 1/\rho)^{-1} h^* \, h \geq 0) \end{aligned} \tag{21}$$

$\operatorname{sgn}(z_{MMSE}) = \operatorname{sgn}(z_{ZF})$라는 것은 극 좌표계로 나타냈을 때의 각도가 같다는 것을 의미한다. **Binary modulation**이나 **4-QAM**의 경우 **amplitude**가 아닌 **phase**에만 영향을 받기 때문에 감지되는 signal은 receiver가 ZF인지, MMSE인지에 관계없이 동일해진다.

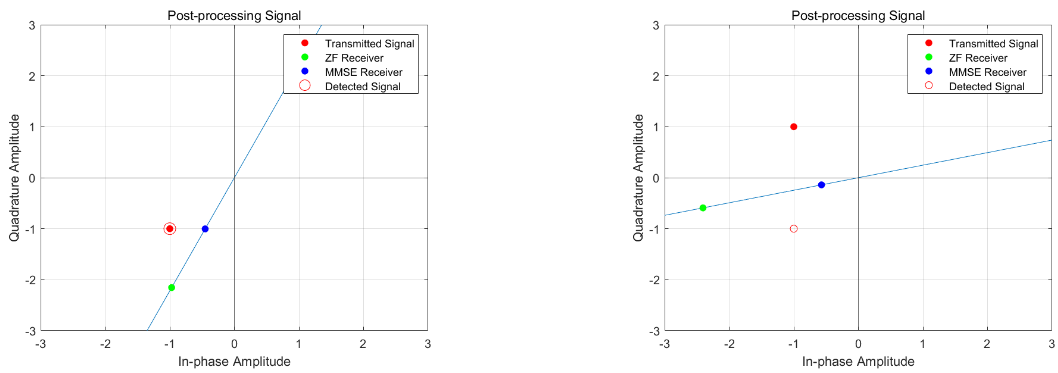
```
>> [theta_zf, rho_zf] = cart2pol(real(DetectionSymbolSequence_ZF), imag(DetectionSymbolSequence_ZF));
>> [theta_mmse, rho_mmse] = cart2pol(real(DetectionSymbolSequence_MMSE), imag(DetectionSymbolSequence_MMSE));
>> max(abs(theta_zf-theta_mmse))

ans =

   4.4409e-16
```

Figure 4: EsN0=5dB

Figure 4는 *Matlab*을 통해 z_{MMSE}와 z_{ZF}를 직교좌표계로 바꾼 뒤, 두 θ 값을 비교한 것이다. θ값의 차이 중 가장 큰 값은 $4.4409e^{-16}$이었다. 이는 매우 작은 값으로 컴퓨터가 가지는 **'finite precision'**로 인해 생겨난 오차로 생각할 수 있다. 그러므로 모든 경우에 대해서 $\theta_{MMSE} - \theta_{ZF} = 0$로 생각할 수 있다. 즉, 실험적으로도 $\theta_{MMSE} = \theta_{ZF}$이 성립함을 확인한 것이다. **Figure 5**의 z_{MMSE}와 z_{ZF}가 하나의 직선 위에 나타난 것을 통해 이를 시각적으로도 확인할 수 있다.



(a) Correct Detection Example

(b) Signal Detection Error Example

Figure 5: 4-QAM의 ZF, MMSE Post-processing Signal Example

3 미해결 및 추가연구 필요

3.1 SNR이 커져도 ZF와 MMSE가 같아지지 않는 이유?

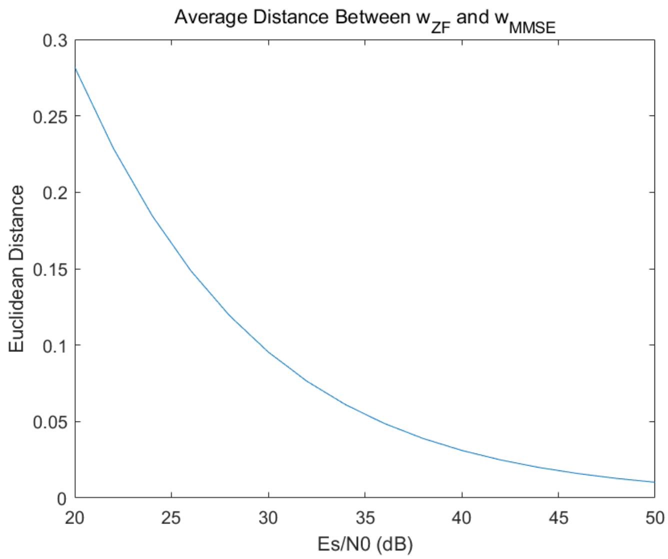
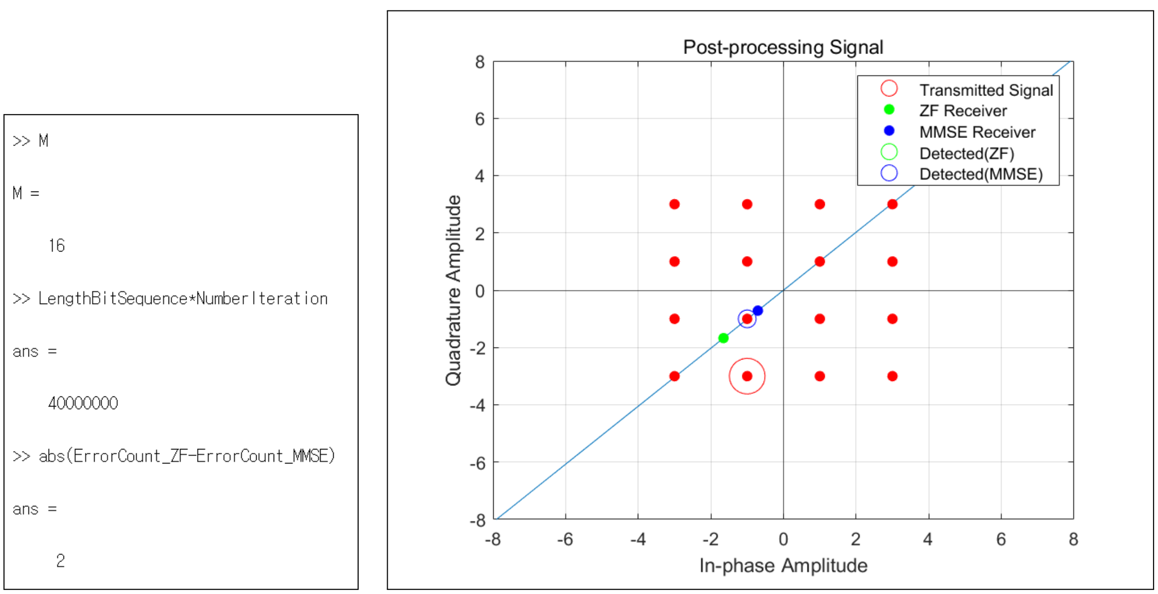


Figure 6



(a) (b)

Figure 7: EsN0=65dB

Figure 6에서 볼 수 있다싶이, SNR이 커질수록 w_{MMSE}와 w_{ZF} 간의 Euclidean Distance는 줄어들고 있다. SNR이 증가함에 따라 BER의 간극이 줄어드는 경향이 보인다. 하지만 **Figure 7 (a)**에서 보다시피 $Es/N0 = 65(dB)$의 조건하에서도 차이가 없어지지는 않는다. **Figure 7 (b)**는 s_{MMSE}와 s_{ZF}의 차이가 크게 나온 하나의 symbol 예시이다.

EsN0: 65 dB = 3.1623e-07
H: -1.8265e-04 + 9.1521e-04i
abs(H): 9.3325e-04
mean(abs(H)): 0.8862
w_mmse: -1.5385e+02 - 7.7090e+02i
w_zf: -2.0971e+02 - 1.0508e+03i

s_{MMSE}와 s_{ZF} 차이가 큰 경우, 일반적으로 abs(H)의 값이 작은 듯하다. 추가조사 필요..

3.2 ZF와 MLD간 BER의 동일성

$$\begin{aligned}\hat{s}_{ZF} &= \underset{s}{\operatorname{argmin}}|z - s|^2 \\ &= \underset{s}{\operatorname{argmin}}|wy - s|^2 \\ &= \underset{s}{\operatorname{argmin}}|\frac{y}{h} - s|^2 \\ &= \underset{s}{\operatorname{argmin}}|\frac{y - hs}{h}|^2\end{aligned} \tag{22}$$

여기에서 $\operatorname{argmin}_s|\frac{y-hs}{h}|^2 = \operatorname{argmin}_s|y - hs|^2$ 성립하는가?

4 Entire Code [1]

```
close all
clear
clc

% Simulation
M = [2 4 16]
Nt = 1;
NumberOfSignals = 10^2;
LengthBitSequence = Nt * NumberOfSignals*log2(M); % log2(M) bits per
    signal

NumberIteration = 10^3;

Es = 1;

EsN0_dB = -2:2:20;
EsN0 = db2pow(EsN0_dB);

EbN0 = EsN0 / log2(M);
EbN0_dB = pow2db(EbN0);

ErrorCount_ZF = zeros(1, length(EbN0_dB));
ErrorCount_MMSE = zeros(1, length(EbN0_dB));
ErrorCount_MLD = zeros(1, length(EbN0_dB));

alphabet = qammod([0:M-1], M, 'UnitAveragePower', true);

```

[1] Uploaded on https://github.com/lightwick/ICS_project

```
for iTotal = 1 : NumberIteration
    % Bit Generation
    BitSequence = randi([0 1], 1, LengthBitSequence);
    SymbolSequence = qammod(BitSequence.', M, 'InputType', 'bit', 'UnitAveragePower', 1).';

    % Noise (n) Generation
    NoiseSequence = (randn(1, length(SymbolSequence)) + 1j * randn(1, length(SymbolSequence))) / sqrt(2);

    % Channel (h) Generation
    H = (randn(1, length(SymbolSequence)) + 1j * randn(1, length(SymbolSequence))) ./ sqrt(2);

    for indx_EbN0 = 1 : length(EbN0)
        % Received Signal (y = s + n) Generation
        ReceivedSymbolSequence = H .* SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0));

        % ZF Receiver
        w_zf = H.^(-1);
        DetectionSymbolSequence_ZF = ReceivedSymbolSequence .* w_zf; % Detection (Zero-Forcing: y / h)

        % MMSE Receiver
        w_mmse = (abs(H).^2+1/EsN0(indx_EbN0)).^(-1) .* conj(H);
        DetectionSymbolSequence_MMSE = ReceivedSymbolSequence .* w_mmse;;

        % MLD Receiver;
        arg = (ones(length(alphabet),1) * ReceivedSymbolSequence) - (alphabet.' * H);
        arg = abs(arg).^2;
        [val,idx] = min(arg);

        % Symbol Sequence -> Bit Sequence
        DetectionBitSequence_ZF = qamdemod(DetectionSymbolSequence_ZF.', M, 'OutputType', 'bit', 'UnitAveragePower', 1)'; % Detection
        DetectionBitSequence_MMSE = qamdemod(DetectionSymbolSequence_MMSE.', M, 'OutputType', 'bit', 'UnitAveragePower', 1)'; % tmp value;
        DetectionBitSequence_MLD = reshape(de2bi(idx-1, log2(M), 'left-msb')', 1, []);

        ErrorCount_ZF(1, indx_EbN0) = ErrorCount_ZF(1, indx_EbN0) + sum(DetectionBitSequence_ZF~=BitSequence);
        ErrorCount_MMSE(1, indx_EbN0) = ErrorCount_MMSE(1, indx_EbN0) + sum(DetectionBitSequence_MMSE~=BitSequence);
        ErrorCount_MLD(1, indx_EbN0) = ErrorCount_MLD(1, indx_EbN0) + sum(DetectionBitSequence_MLD~=BitSequence);
    end
end

BER_Simulation_ZF = ErrorCount_ZF / (LengthBitSequence * NumberIteration);
```

```
BER_Simulation_MMSE = ErrorCount_MMSE / (LengthBitSequence *
    NumberIteration);
BER_Simulation_MLD = ErrorCount_MLD / (LengthBitSequence * NumberIteration
    );

if M==2
     BER_Theory = berfading(EbN0_dB, 'psk', 2, 1);
else
     BER_Theory = berfading(EbN0_dB, 'qam', M, 1); % not sure if 'dataenc'
         needs to be specified; I don't even know what it does
end

% Plot
figure()
semilogy(EsN0_dB, BER_Theory, 'r--');
hold on
semilogy(EsN0_dB, BER_Simulation_ZF, 'bo');
semilogy(EsN0_dB, BER_Simulation_MMSE, 'bx');
semilogy(EsN0_dB, BER_Simulation_MLD, 'b^');


axis([-2 20 10^-3 0.5])
grid on
legend('Theory (Rayleigh)', 'ZF (Rayleigh)', 'MMSE (Rayleigh)', 'MLD (
    Rayleigh)');
xlabel('Es/No [dB]');
ylabel('BER');
title('BER for QAM (M='+string(M)+')');
```