Simulating MRC, MRT, Alamouti in a Rayleigh Fading, MIMO Environment

Project #4

Intelligent Communication Systems (ICS) Lab.
노용재

Winter Intern Seminar (2023-1)

Contents

---

1 Implementation

1.1 SIMO: Maximum Ratio Combining(MRC)

For simplicity, let's assume $N_T = 1, N_R = 2$. The channel matrix H would look like the following.

$$\boldsymbol{H} = \begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix} \tag{1}$$

The received symbol will look something like this.

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{H}s + \boldsymbol{n} \qquad (2) \\ &= \begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix} s + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \qquad (3) \end{aligned}$$

By multiplying $\boldsymbol{H}^H$, we can obtain the channel gain as shown below.

$$
\begin{aligned}
z &= \boldsymbol{H}^H \boldsymbol{y} && (4)\\
&= \begin{bmatrix} h_{11}^* & h_{21}^* \end{bmatrix} \left(\begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix} s + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \right) && (5)\\
&= \|h\|_F^2 s + \tilde{\boldsymbol{n}} && (6)
\end{aligned}
$$

```
NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt

y = ReceivedSymbolSequence;
z = H'*y;

DetectedSignal = qamdemod(z/norm(H,'fro')^2*NormalizationFactor, M);
```

1.2 Channel Unknown: Alamouti Scheme

1.2.1 MISO (N_T=2, N_R=1)

$$
\begin{aligned}
\boldsymbol{z} &= \boldsymbol{H_{eff}}^H \boldsymbol{y} \\
&= \begin{bmatrix} h_1^* & h_2 \\ h_2^* & -h_1 \end{bmatrix} \left(\begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix} \boldsymbol{s} + \boldsymbol{n} \right) \\
&= \begin{bmatrix} \|h\|_F^2 & 0 \\ 0 & \|h\|_F^2 \end{bmatrix} \boldsymbol{s} + \tilde{\boldsymbol{n}}
\end{aligned} \tag{7}
$$

```
NormalizationFactor = sqrt(2/3*(M-1) * Nt);
NormalizedSymbol = SymbolSequence(1:Nt, 1)/ NormalizationFactor;

% Row prepresents antenna, Column represents time-slot
STBC = [NormalizedSymbol.'; -conj(NormalizedSymbol(2,1)) conj(
    NormalizedSymbol(1,1))].';

y_alamouti = (H(1:Nr, 1:Nt) * STBC).' + NoiseSequence(1:Nt, 1) * sqrt(1 /
    EsN0(indx_EbN0));

Augmented_H = [H; conj(H(1,2)) -conj(H(1,1))];

y = [ReceivedSymbolSequence(1,1); conj(ReceivedSymbolSequence(2,1))];
z = Augmented_H' * y;

FrobSquared = norm(H,'fro')^2
DetectedSignal = qamdemod(z/FrobSquared*NormalizationFactor, M);
```

1.2.2 MIMO (N_T=2, N_R=2)

$$
\begin{aligned}
\boldsymbol{z} &= \boldsymbol{H_{eff}}^H \boldsymbol{y} \\
&= \begin{bmatrix} h_{11}^* & h_{21}^* & h_{12} & h_{22}^* \\ h_{12}^* & h_{22}^* & -h_{11} & -h_{21}^* \end{bmatrix} \left(\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ h_{12}^* & -h_{11}^* \\ h_{22}^* & -h_{21}^* \end{bmatrix} \boldsymbol{s} + \boldsymbol{n} \right) \\
&= \begin{bmatrix} \|h\|_F^2 & 0 \\ 0 & \|h\|_F^2 \end{bmatrix} \boldsymbol{s} + \tilde{\boldsymbol{n}}
\end{aligned} \tag{8}
$$

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

NormalizedSymbol = SymbolSequence(1:Nt, 1) / NormalizationFactor;

% Row prepresents antenna, Column represents time-slot
STBC = [NormalizedSymbol.'; -conj(NormalizedSymbol(2,1)) conj(
    NormalizedSymbol(1,1))].';
Hs = reshape((H(1:Nr, 1:Nt) * STBC), [], 1);
y_alamouti = Hs + NoiseSequence * sqrt(1 / EsN0(indx_EbN0));

tmp_H = conj(H(:,[2 1]));
tmp_H(:,2) = -tmp_H(:,2);

Augmented_H = [H; tmp_H];

y([3 4], :) = conj(y([3 4], :));
z = Augmented_H' * y;
FrobSquared = norm(H,'fro')^2;

DetectedSignal = qamdemod(z/FrobSquared*NormalizationFactor, M);
```

1.3 Channel Known: Maximum Ratio Transmission (MRT)

1.3.1 MISO (N_T=2, N_R=1)

While $\|\boldsymbol{w}\|_F^2 = N_T$

$$\boldsymbol{y} = \sqrt{\frac{E_s}{N_T}}\boldsymbol{hws} \tag{9}$$

$$\boldsymbol{w} = \sqrt{N_T}\frac{\boldsymbol{h}^H}{\sqrt{\|\boldsymbol{h}\|_F^2}} \tag{10}$$

```
NormalizationFactor = sqrt(2/3*(M-1) * norm(H, "fro")^2);
w = H';
TransmitSymbol = w * SymbolSequence / NormalizationFactor;
ReceivedSymbol = H*TransmitSymbol + Noise;

FrobSquared = norm(H,'fro')^2
DetectedSignal = qamdemod(ReceivedSymbol/FrobSquared*NormalizationFactor,
    M)
```

1.3.2 Dominant Eigenmode Transmission MIMO (N_T=2, N_R=2)

$\boldsymbol{H}$의 특성

SVD에 따르면 orthogonal한 행렬 $\boldsymbol{U}$와 $\boldsymbol{V}$에 대해 다음이 만족된다.

$$\boldsymbol{H} = \boldsymbol{U\Sigma V^H} \tag{11}$$

$$\boldsymbol{H}^H = \boldsymbol{V\Sigma}^H\boldsymbol{U}^H \tag{12}$$

$$\boldsymbol{HH}^H = \boldsymbol{U\Sigma}^2\boldsymbol{U}^H \tag{13}$$

$$\boldsymbol{H}^H\boldsymbol{H} = \boldsymbol{V\Sigma}^2\boldsymbol{V}^H \tag{14}$$

$$\tag{15}$$

이론

$$\boldsymbol{y} = \sqrt{\frac{E_s}{N_T}}\boldsymbol{H}\boldsymbol{w}s + \boldsymbol{n} \tag{16}$$

$$\begin{aligned} z &= \boldsymbol{g^H}\boldsymbol{y} \\ &= \sqrt{\frac{E_s}{N_T}}\boldsymbol{g^H}\boldsymbol{H}\boldsymbol{w}s + \boldsymbol{g^H}\boldsymbol{n} \end{aligned} \tag{17}$$

Channel gain을 최대화하기 위해 $\boldsymbol{g}^H = (\boldsymbol{H}\boldsymbol{w})^H$로 두자.

$$\begin{aligned} \boldsymbol{g}^H\boldsymbol{H}\boldsymbol{w} &= (\boldsymbol{H}\boldsymbol{w})^H\boldsymbol{H}\boldsymbol{w} \\ &= \boldsymbol{w}^H\boldsymbol{H}^H\boldsymbol{H}\boldsymbol{w} \\ &= \boldsymbol{w}^H\boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^H\boldsymbol{w} \ (\because equation(3)) \end{aligned} \tag{18}$$

2x2 MIMO에서 $\boldsymbol{w} = [w_1\ w_2]^T$이다. 식(7)을 전개하면 $\sigma_1^2|\boldsymbol{v_1}^H\boldsymbol{w}|^2 + \sigma_2^2|\boldsymbol{v_2}^H\boldsymbol{w}|^2$이므로 $\boldsymbol{w}$의 전력이 고정돼 있을 때, $\boldsymbol{v_1} = \boldsymbol{w}$로 두는 게 최선이다.

$$\boldsymbol{y} = \sqrt{\frac{E_s}{N_T}}\boldsymbol{H}\boldsymbol{w}s + \boldsymbol{n} \tag{19}$$

$$\boldsymbol{z} = \boldsymbol{g}^H\sqrt{\frac{E_s}{N_T}}\boldsymbol{H}\boldsymbol{w}\boldsymbol{s} + \boldsymbol{n} \tag{20}$$

$$\tag{21}$$

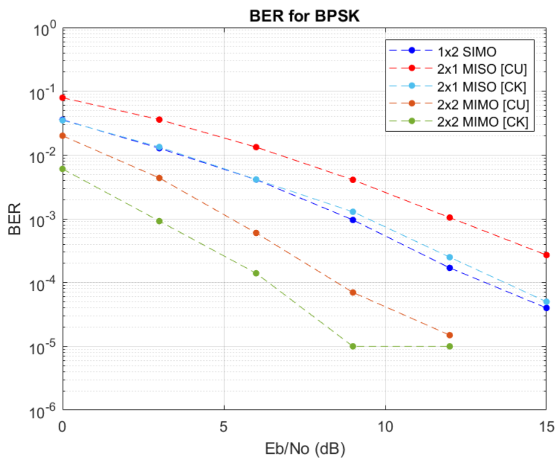
If $\boldsymbol{H} = \boldsymbol{U\Sigma V^H}$, the best solution would be, $\boldsymbol{U} = \boldsymbol{g}$, $\boldsymbol{V} = \boldsymbol{w}/\sqrt{\boldsymbol{N_T}}$ to maximize channel gain.

```
% S = svd(A) returns the singular values of matrix A in descending order.
[U,S,V] = svd(H);
w =V(:,1);
g = U(:,1);

NormalizationFactor = sqrt(2/3*(M-1));
TransmitSymbol = w * SymbolSequence / NormalizationFactor;
ReceivedSymbol = H * TransmitSymbol + Noise;

PostProcessing = (g' * ReceivedSymbol) / S(1,1);

DetectedSignal = qamdemod(PostProcessing*NormalizationFactor , M);
```

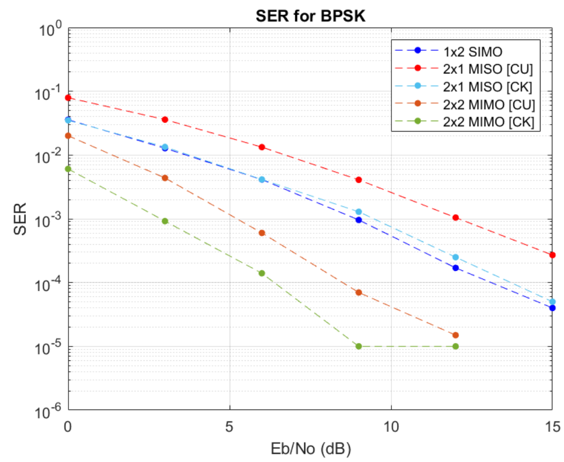
2 결과 및 분석

2.1 Simulation Result

2.1.1 BPSK



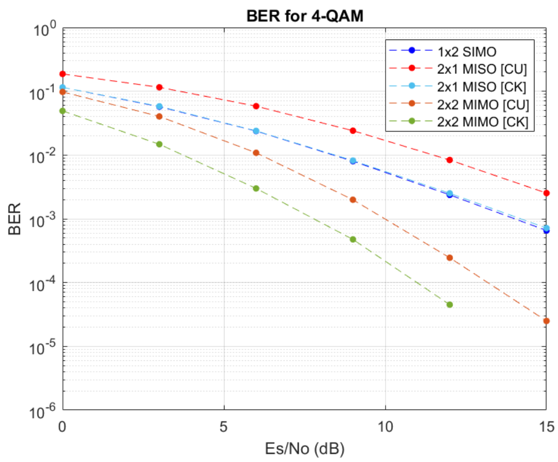
(a) BER



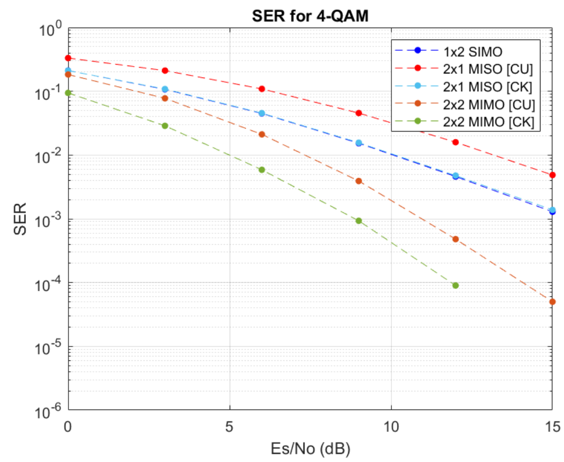
(b) SER

Figure 1

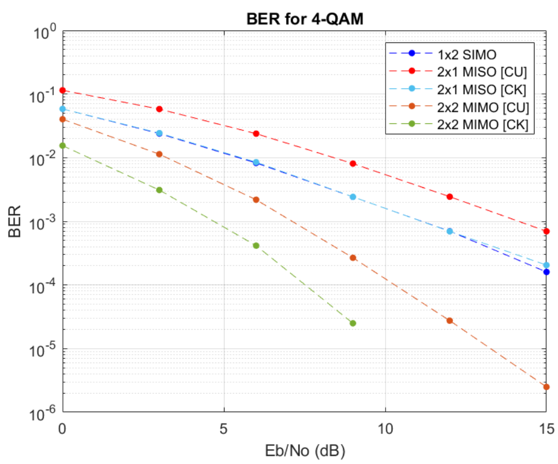
2.1.2 4-QAM



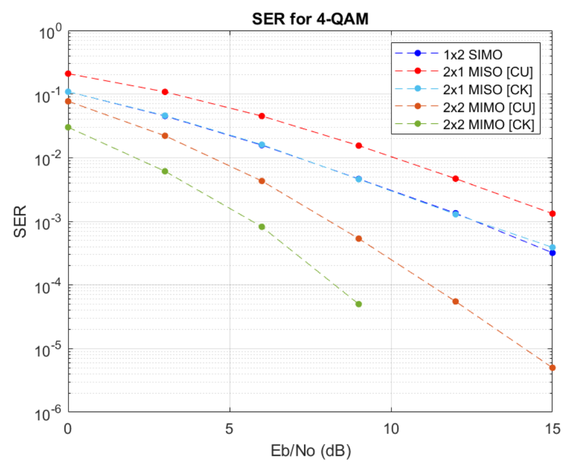
(a) BER



(b) SER



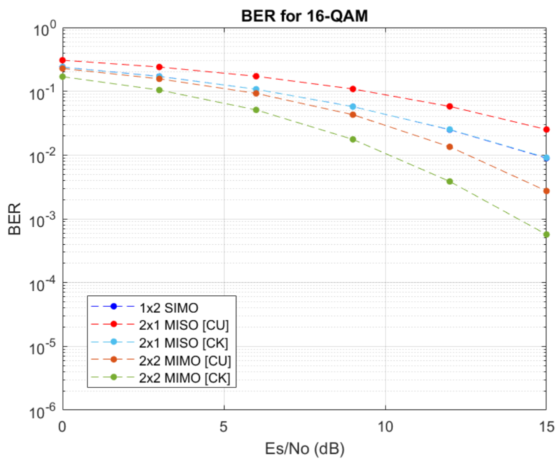
(c) BER



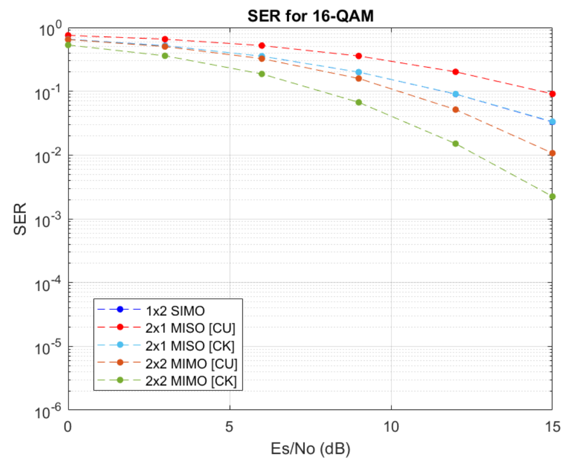
(d) SER

Figure 2

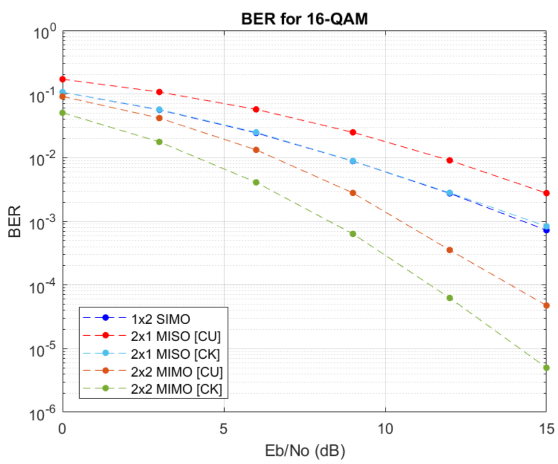
2.1.3 16-QAM



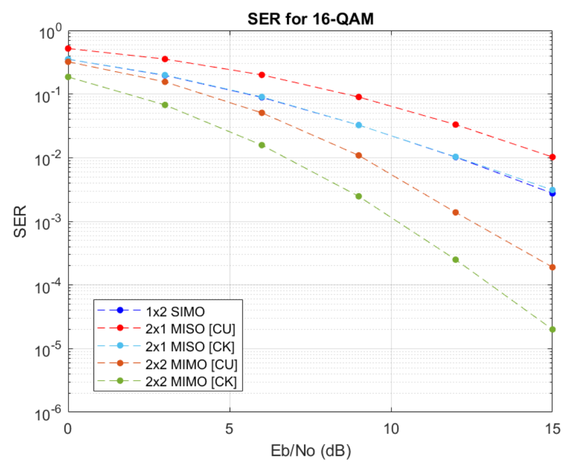
(a) BER



(b) SER



(c) BER



(d) SER

Figure 3

2.2 결과 분석

3 개선사항

- 구현은 했으나, Eigenmode decomposition 내용을 정확하게 이해하지 못해 구체적인 분석은 하지 못했다.

4 Entire Code [1]

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

main.m

```
close all
clear all
dbstop if error
% dbstop if warning

addpath('../tools/')

% Environment Varible
M = 4

NumberIteration = 10^5;
SimulationNum = 5;

% Simulation
EbN0_dB = 0:3:15;
EbN0 = db2pow(EbN0_dB);

EsN0 = EbN0 * log2(M);
EsN0_dB = pow2db(EsN0);

% EsN0_dB = 0:3:15;
% EsN0 = db2pow(EsN0_dB);
%
% EbN0 = EsN0 / log2(M);
% EbN0_dB = pow2db(EbN0);

% BitErrorCount_ZF = zeros(1, length(EsN0_dB));
% SignalErrorCount_ZF = zeros(1, length(EsN0_dB));
% BitErrorCount_MLD = zeros(1, length(EsN0_dB));
% SignalErrorCount_MLD = zeros(1, length(EsN0_dB));
% 33edd
% BitErrorCount_MMSE = zeros(1, length(EsN0_dB));
% SignalErrorCount_MMSE = zeros(1, length(EsN0_dB));

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

BitErrorCount = zeros(SimulationNum, length(EsN0_dB));
SignalErrorCount = zeros(SimulationNum, length(EsN0_dB));

BEC_tmp = zeros(size(BitErrorCount));
SEC_tmp = zeros(size(SignalErrorCount));

FivePercent = ceil(NumberIteration/20);

for iTotal = 1 : NumberIteration
    if mod(iTotal-100, FivePercent)==0
        tic
    end
    % Bit Generation
    SignalSequence = randi([0 M-1], 2, 1);
    SignalBinary = de2bi(SignalSequence, log2(M), 'left-msb');
    SymbolSequence = qammod(SignalSequence, M);
%     SymbolSequence = qammod(SignalSequence, M) / NormalizationFactor;

    NoiseSequence = (randn(4, 1) + 1j * randn(4, 1)) / sqrt(2); % Noise (n) Generation
%     NoiseSequence = zeros(4,1);
    H = (randn(2, 2) + 1j * randn(2, 2)) ./ sqrt(2); % Receiver x Transmitter

    for indx_EbN0 = 1 : length(EsN0)
        %% SIMO (MRC)
        Nt =1;
```

```
        Nr = 2;
        NormalizedSymbol = SymbolSequence(1:Nt, 1) / (NormalizationFactor * sqrt(Nt));
        Noise = NoiseSequence(1:Nr, 1) * sqrt(1 / EsN0(indx_EbN0));

        y = H(1:Nr, 1:Nt) * NormalizedSymbol + Noise;
        [BEC_tmp(1, indx_EbN0), SEC_tmp(1, indx_EbN0)] = simo_mrc(y, SignalSequence(1:Nt,1), SignalBinary(1:Nt, :),  M, H(1:Nr, 1:Nt));

        %% MISO (Alamouti)
        Nt = 2;
        Nr = 1;
        NormalizedSymbol = SymbolSequence(1:Nt, 1)/ (NormalizationFactor * sqrt(Nt));
        % Row prepresents antenna, Column represents time-slot
        STBC = [NormalizedSymbol.'; -conj(NormalizedSymbol(2,1)) conj(NormalizedSymbol(1,1))].';
        y_alamouti = (H(1:Nr, 1:Nt) * STBC).' + NoiseSequence(1:Nt, 1) * sqrt(1 / EsN0(indx_EbN0));

        [BEC_tmp(2, indx_EbN0), SEC_tmp(2, indx_EbN0)] = miso_alamouti(y_alamouti, SignalSequence(1:Nt, 1), SignalBinary(1:Nt, :),  M, H(1:Nr,1:Nt));

        %% MISO (MRT)
        Nt = 2;
        Nr = 1;
        H_new = H(1:Nr,1:Nt);
        [BEC_tmp(3, indx_EbN0), SEC_tmp(3, indx_EbN0)] = miso_mrt(SymbolSequence(1), SignalSequence(1), Noise(1:Nr), SignalBinary(1, :), M, H_new);

        %% MIMO (Alamouti)
        Nt = 2;
        Nr = 2;
        NormalizedSymbol = SymbolSequence(1:Nt, 1)/ (NormalizationFactor * sqrt(Nt));
        % Row prepresents antenna, Column represents time-slot
        STBC = [NormalizedSymbol.'; -conj(NormalizedSymbol(2,1)) conj(NormalizedSymbol(1,1))].';
        Hs = reshape((H(1:Nr, 1:Nt) * STBC), [], 1);
        y_alamouti = Hs + NoiseSequence * sqrt(1 / EsN0(indx_EbN0));
        [BEC_tmp(4, indx_EbN0), SEC_tmp(4, indx_EbN0)] = mimo_alamouti(y_alamouti, SignalSequence(1:Nt, 1), SignalBinary(1:Nt, :),  M, H(1:Nr, 1:Nt));

        %% MIMO (MRT)
        Nt = 2;
        Nr = 2;
        H_new = H(1:Nr,1:Nt);
        Noise = NoiseSequence(1:Nr, 1) * sqrt(1 / EsN0(indx_EbN0));
        [BEC_tmp(5, indx_EbN0), SEC_tmp(5, indx_EbN0)] = mimo_mrt(SymbolSequence(1), SignalSequence(1), Noise(1:Nr), SignalBinary(1, :), M, H_new);
    end

    BitErrorCount = BitErrorCount + BEC_tmp;
    SignalErrorCount = SignalErrorCount + SEC_tmp;

    if mod(iTotal-100, FivePercent)==0
        ElapsedTime = toc;
        EstimatedTime = (NumberIteration-iTotal)*ElapsedTime;
        disp(sprintf("%d%%, estimated wait time %d minutes %d seconds", round(iTotal/NumberIteration*100), floor(EstimatedTime/60), floor(mod(EstimatedTime, 60))))
    end
end

BER = BitErrorCount / (NumberIteration*log2(M));
```

```
SER = SignalErrorCount / (NumberIteration);

BER(2,:) = BER(2,:) / 2;
SER(2,:) = SER(2,:) / 2

BER(4,:) = BER(4,:) / 2;
SER(4,:) = SER(4,:) / 2;

% Plot
BER_Title = sprintf("BER for %d-QAM", M);
SER_Title = sprintf("SER for %d-QAM", M);
x_axis = "Eb/No (dB)";

legend_order = ["1x2 SIMO", "2x1 MISO [CU]", "2x1 MISO [CK]", "2x2 MIMO [CU]", "2x2 MIMO
    [CK]"];
myplot(EbN0_dB, BER, BER_Title, x_axis, "BER", legend_order);
ylim([10^(-6) 1])
myplot(EbN0_dB, SER, SER_Title, x_axis, "SER", legend_order);
ylim([10^(-6) 1])
```

simo_mrc.m

```
function [BitErrorCount, SignalErrorCount] = simo_mrc(ReceivedSymbolSequence,
   SignalSequence, SignalBinary,  M, H)
    Nt = size(H,2);
    Nr = size(H,1);
    assert(Nt==1, 'Nt is not 1')
    NormalizationFactor = sqrt(2/3*(M-1) * Nt);

    y = ReceivedSymbolSequence;
    z = H'*y;

    DetectedSignal = qamdemod(z/norm(H,'fro')^2*NormalizationFactor, M);
    DetectedBinary = de2bi(DetectedSignal, log2(M), 'left-msb');

    SignalErrorCount = sum(DetectedSignal~=SignalSequence, 'all');
    BitErrorCount = sum(SignalBinary~=DetectedBinary, 'all');
end
```

miso_alamouti.m

```
function [BitErrorCount, SignalErrorCount] = miso_alamouti(ReceivedSymbolSequence,
   SignalSequence, SignalBinary,  M, H)
    Nt = size(H,2);
    Nr = size(H,1);
    assert(Nt==2 && Nr==1, "Need H of size 1x2")

    Augmented_H = [H; conj(H(1,2)) -conj(H(1,1))];

    NormalizationFactor = sqrt(2/3*(M-1) * Nt);

    y = [ReceivedSymbolSequence(1,1); conj(ReceivedSymbolSequence(2,1))];
    z = Augmented_H' * y;

    FrobSquared = H*H'; % equivalent to norm(H,'fro')^2
    DetectedSignal = qamdemod(z/FrobSquared*NormalizationFactor, M);
    DetectedBinary = de2bi(DetectedSignal, log2(M), 'left-msb');

    SignalErrorCount = sum(DetectedSignal~=SignalSequence, 'all');
    BitErrorCount = sum(SignalBinary~=DetectedBinary, 'all');
end
```

miso_mrt.m

```
function [BitErrorCount, SignalErrorCount] = miso_mrt(SymbolSequence, SignalSequence,
    Noise, SignalBinary,  M, H)
    Nt = size(H,2);
    Nr = size(H,1);
    assert(Nt==2 && Nr==1, "Need H of size 1x2")

    NormalizationFactor = sqrt(2/3*(M-1) * norm(H, "fro")^2);
    w = H';
    TransmitSymbol = w * SymbolSequence / NormalizationFactor;
    ReceivedSymbol = H*TransmitSymbol + Noise;

    FrobSquared = H*H'; % equivalent to norm(H,'fro')^2
    DetectedSignal = qamdemod(ReceivedSymbol/FrobSquared*NormalizationFactor, M);
    DetectedBinary = de2bi(DetectedSignal, log2(M), 'left-msb');

    SignalErrorCount = sum(DetectedSignal~=SignalSequence, 'all');
    BitErrorCount = sum(SignalBinary~=DetectedBinary, 'all');
end
```

mimo_alamouti.m

```
function [BitErrorCount, SignalErrorCount] = mimo_alamouti(y, SignalSequence,
    SignalBinary,  M, H)
    Nt = size(H,2);
    Nr = size(H,1);
    assert(Nt==2 && Nr==2, 'H is not size 2x2')
    assert(length(SignalSequence), "Signal Sequence is not 2")

    tmp_H = conj(H(:,[2 1]));
    tmp_H(:,2) = -tmp_H(:,2);

    Augmented_H = [H; tmp_H];
    NormalizationFactor = sqrt(2/3*(M-1) * Nt);

    y([3 4], :) = conj(y([3 4], :));
    z = Augmented_H' * y;
    FrobSquared = norm(H,'fro')^2;

    DetectedSignal = qamdemod(z/FrobSquared*NormalizationFactor, M);
    DetectedBinary = de2bi(DetectedSignal, log2(M), 'left-msb');

    SignalErrorCount = sum(DetectedSignal~=SignalSequence, 'all');
    BitErrorCount = sum(SignalBinary~=DetectedBinary, 'all');
end
```

mimo_mrt.m

```
function [BitErrorCount, SignalErrorCount] = mimo_mrt(SymbolSequence, SignalSequence,
    Noise, SignalBinary,  M, H)
    Nt = size(H,2);
    Nr = size(H,1);

    % S = svd(A) returns the singular values of matrix A in descending order.
    [U,S,V] = svd(H);
    w =V(:,1);
    g = U(:,1);

    NormalizationFactor = sqrt(2/3*(M-1));
    TransmitSymbol = w * SymbolSequence / NormalizationFactor;
    ReceivedSymbol = H * TransmitSymbol + Noise;

    Processing = g' * ReceivedSymbol;

    DetectedSignal = qamdemod(Processing/S(1,1)*NormalizationFactor, M);
```

```
    DetectedBinary = de2bi(DetectedSignal, log2(M), 'left-msb');

    SignalErrorCount = sum(DetectedSignal~=SignalSequence, 'all');
    BitErrorCount = sum(SignalBinary~=DetectedBinary, 'all');
end
```