Simulating Different Receivers in a Rayleigh Fading, MIMO Environment

Project #2

Intelligent Communication Systems (ICS) Lab.
노용재

Winter Intern Seminar (2023-1)

Contents

---

1 Implementation

다음의 조건을 만족하는 환경에 해당한다.

$$N_t \leq N_r \tag{1}$$

```
placeholder = 4

% Environment Variables
M = placeholder
Nt = placeholder
Nr =placeholder
NormalizationFactor = sqrt(2/3*(M-1)*Nt);

% Signal Generation
SignalSequence = randi([0 M-1], Nt, 1);
SignalBinary = de2bi(SignalSequence, log2(M), 'left-msb');
SymbolSequence = qammod(SignalSequence, M) / NormalizationFactor;
```

```
% Noise (n) Generation
NoiseSequence = (randn(Nt, 1) + 1j * randn(Nt, 1)) / sqrt(2);
H = (randn(Nr, Nt) + 1j * randn(Nr, Nt)) ./ sqrt(2);
```

*Normalization Factor*를 $\sqrt{\frac{2}{3}(M-1)N_t}$로 설정한 이유는 하나의 trasmitter가 $\frac{1}{N_t}$W의 전력을 갖도록 하기 위해서이다.

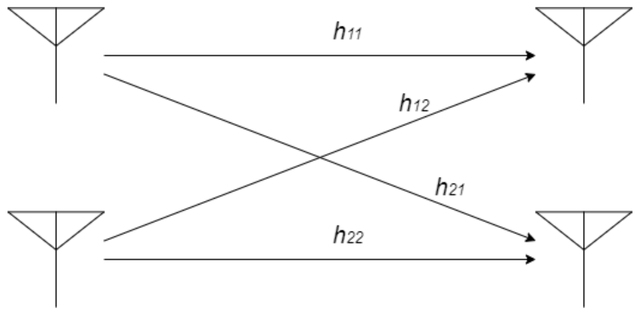


Figure 1

$$H = \begin{bmatrix} h_{11} & \dots & h_{1N_T} \\ \vdots & \ddots & \vdots \\ h_{N_R 1} & \dots & h_{N_R N_T} \end{bmatrix} \tag{2}$$

1.1 ZF(Zero-forcing)

Moore Penrose Pseudo Inverse

$W_{ZF}H = I$를 만족하는 W_{ZF}를 찾으려고 한다. $H^H H$는 square matrix라는 사실을 이용하여 W_{ZF}를 구할 수 있다는 사실을 이용해 $W_{ZF} = (H^H H)^{-1} H^H$임을 알 수 있다.

$$\begin{aligned} W_{ZF}H &= (H^H H)^{-1} H^H H \\ &= (H^H H)^{-1}(H^H H) \quad (\because \textit{associative property}) \\ &= I \end{aligned} \tag{3}$$

이때, W_{ZF}를 H의 *right pseudo-inverse*라고 할 수 있다.
*Matlab*에는 Moore Penrose Pseudo Inverse를 구할 수 있는 **pinv(H)** 함수가 존재한다.

ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0))

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

w_zf = pinv(H); % pinv(H) = inv(H' * H) * H'
DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence;

DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF*
    NormalizationFactor, M);
DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');
```

ReceivedSymbolSequence = sqrt(EsN0(indx_EbN0)) * H * SymbolSequence + NoiseSequence

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

w_zf = NormalizationFactor /sqrt(EsN0) * pinv(H); % pinv(H) = inv(H' * H) * H'
DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence;

DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF, M);
DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');
```

1.2 MMSE(Minimum Mean Square Error)

$$W_{MMSE} = \sqrt{\frac{N_t}{E_s}}(H^H + \frac{N_t}{\rho})^{-1}H^H \tag{4}$$

ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0))

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

w_mmse = NormalizationFactor * inv(H' * H + Nt / EsN0 * eye(Nt)) * H';
DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence;

DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M);
DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all');
```

ReceivedSymbolSequence = sqrt(EsN0(indx_EbN0)) * H * SymbolSequence + NoiseSequence

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

w_mmse = NormalizationFactor / sqrt(EsN0) * inv(H' * H + Nt / EsN0 * eye(Nt)) * H';
DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence; % Detection (Zero-Forcing: y / h)

DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M); % Detection
DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all');
```

1.3 MLD(Maximum Likelihood Detection)

1.3.1 Creating All Possible Signal Combinations

```
% Creating Matrix for all possible combinations of signals (M^Nt possible combinations)
AllNumbers = de2bi([0:M^Nt-1], Nt*log2(M), 'left-msb');
Candidates = zeros(M^Nt, Nt);
for ii = 1 : M^Nt
    for jj = 1 : Nt
        Candidates(ii,jj) = bi2de(AllNumbers(ii,log2(M)*(jj-1)+1:log2(M)*jj), 'left-msb');
    end
end
Candidates = qammod(Candidates',M) / NormalizationFactor;
```

*AllNumbers*는 각 열마다 0부터 $M^{N_t}-1$를 이진수로 나태낸 matrix이다.

```
AllNumbers =

     0     0     0     0
     0     0     0     1
     0     0     1     0
     0     0     1     1
     0     1     0     0
     0     1     0     1
     0     1     1     0
     0     1     1     1
     1     0     0     0
     1     0     0     1
     1     0     1     0
     1     0     1     1
     1     1     0     0
     1     1     0     1
     1     1     1     0
     1     1     1     1
```

Figure 2: $M = 4,\ N_t = 2$

Figure 2은 $M = 4$, $N_t = 2$일 때의 *AllNumbers*의 예시이다. *AllNumbers*의 각 열을 $\log_2 M$ 숫자의 묶음이 N_t개로 있는 것으로 생각한다. 그렇다면, 각 열은 가능한 하나의 signal combination으로 볼 수 있다. *Candidates*의 dimension은 $M^{N_t} \times N_t$이다. 이는 N_t개의 signal로 만들 수 있는 M^{N_t}개의 signal combination을 나타낸다.

1.3.2 Solving For Minimum Euclidean Distance

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

ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0))

```
% results in Nt x M^Nt, each column representing each candidate symbol combination
EuclideanDistance = abs(ReceivedSymbolSequence * ones(1,M^Nt) - H*Candidates).^2;
[val, idx] = min(sum(EuclideanDistance, 1));
```

```
DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'),log2(M),[])';
DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
```

ReceivedSymbolSequence = sqrt(EsN0(indx_EbN0)) * H * SymbolSequence + NoiseSequence

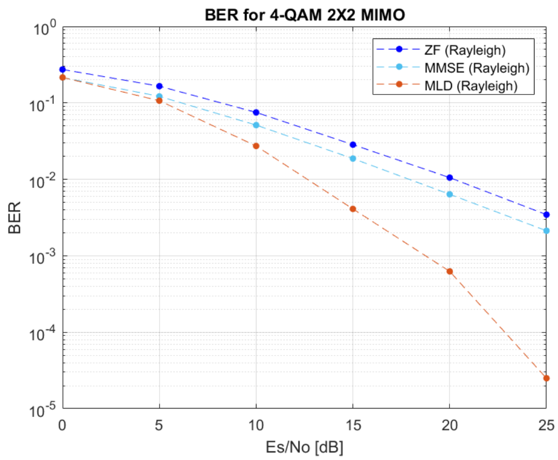
```
% 'EuclideanDistance' results in Nt x M^Nt, each column representing each candidate symbol combination
EuclideanDistance = abs(ReceivedSymbolSequence/sqrt(EsN0) * ones(1,M^Nt) - H*Candidates).^2;
[val, idx] = min(sum(EuclideanDistance, 1));

DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'),log2(M),[])';
DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');

BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
```

$$ReceivedSymbolSequence = [y_1 \ \ \dots \ \ y_{N_R}]^T \tag{6}$$

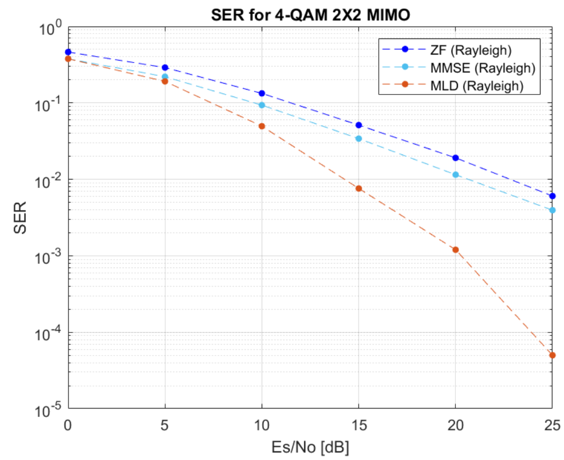
*sum(EuclideanDistance, 1)*는 $\|y - Hs\|_F^2$을 의미한다.

2 결과 및 분석

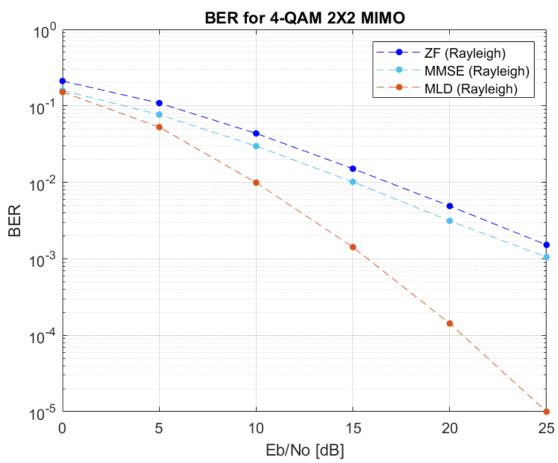
2.1 Simulation Result



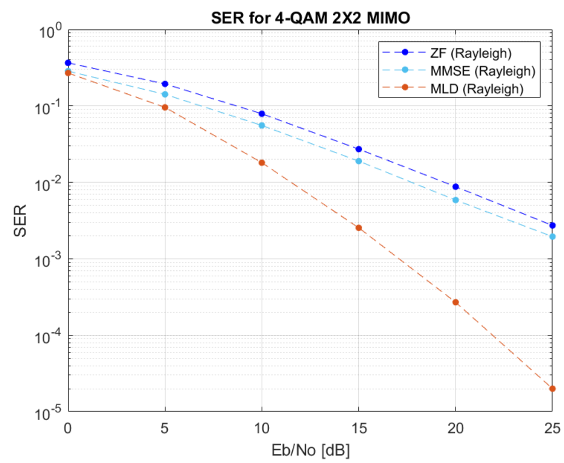
(a) BER



(b) SER

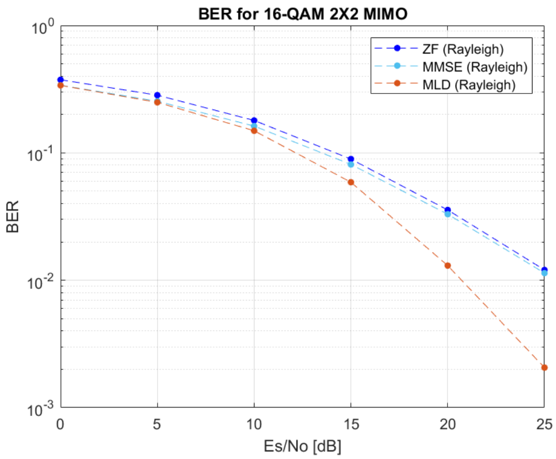


(c) BER

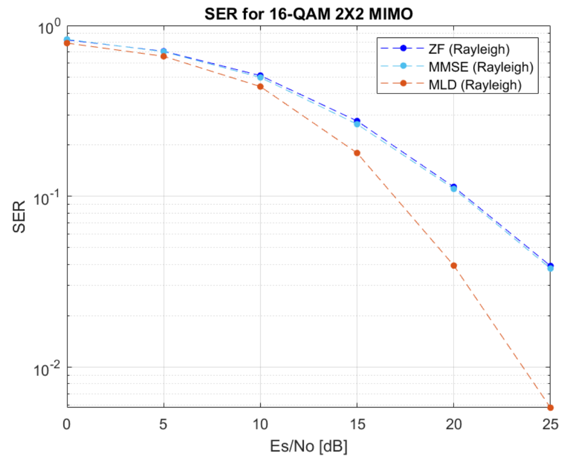


(d) SER

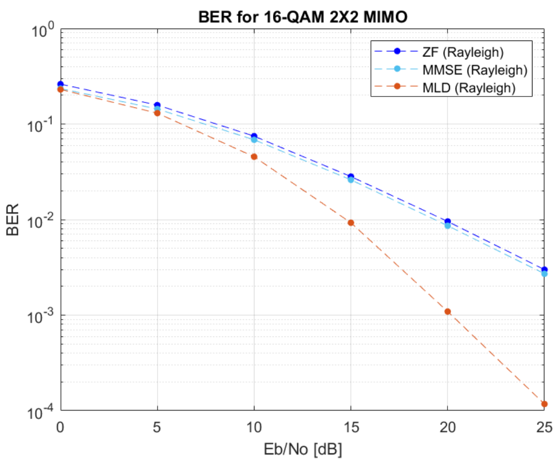
Figure 3: 4-QAM 2×2 MIMO



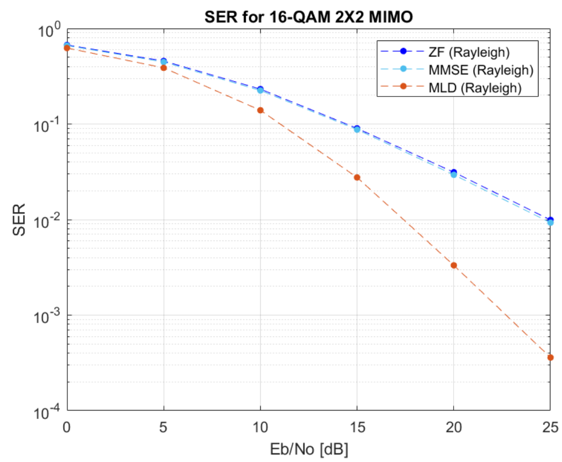
(a) BER



(b) SER

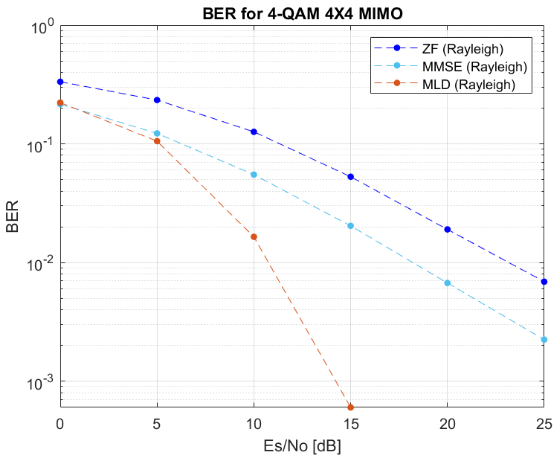


(c) BER

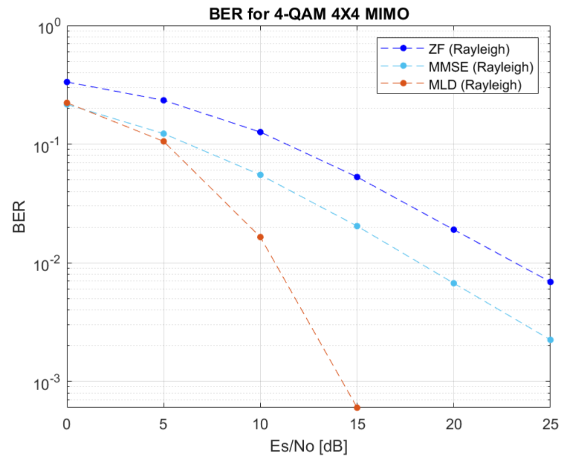


(d) SER

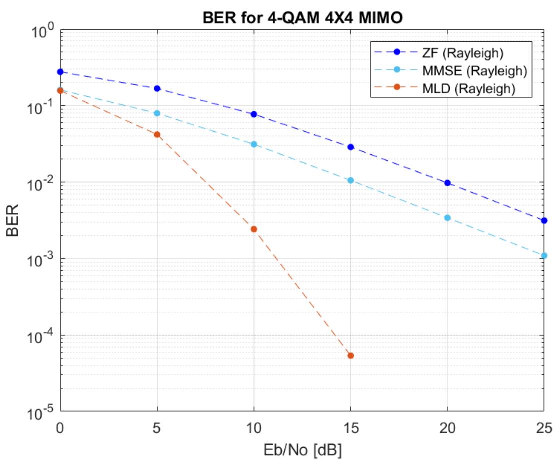
Figure 4: 16-QAM 2×2 MIMO



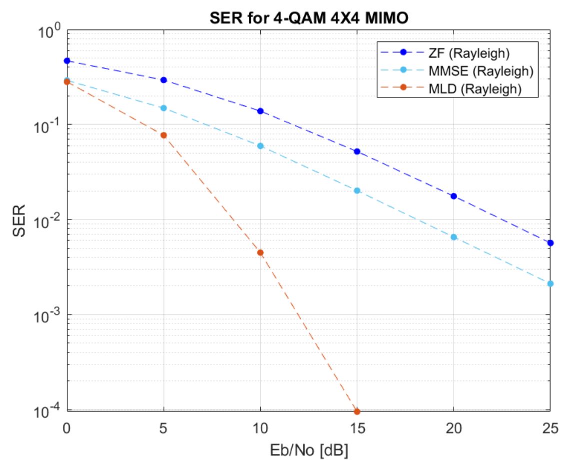
(a) BER



(b) SER

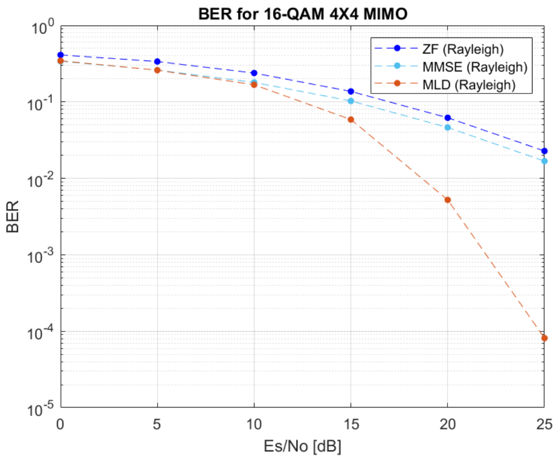


(c) BER

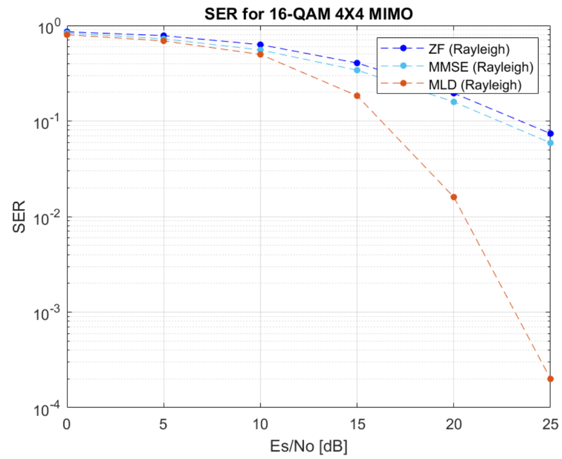


(d) SER

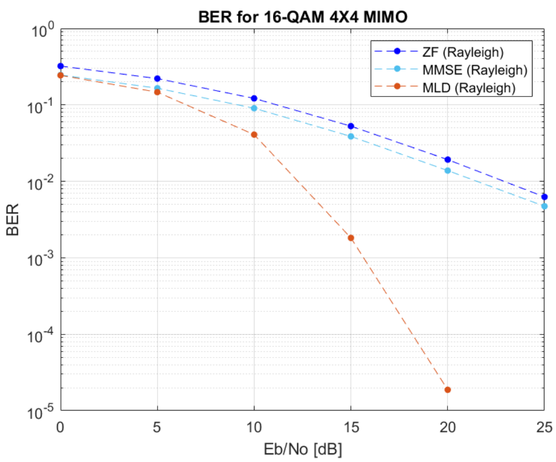
Figure 5: 4-QAM 4×4 MIMO



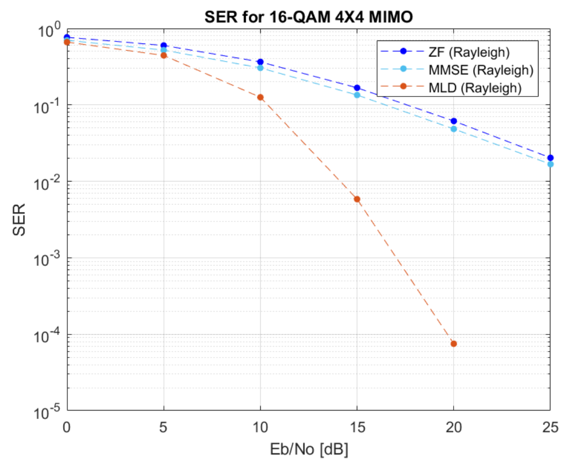
(a) BER



(b) SER



(c) BER



(d) SER

Figure 6: 16-QAM 4×4 MIMO

3 미해결 & 추가연구 필요 내용

- $W_{ZF} = (H^H H)^{-1} H^H$, transpose를 한다고 하더라도 동일한 결과를 가져오는 것 아닌가? 왜 ZF에서 transpose가 아닌 hermitian transpose를 사용한건가?

4 Entire Code [1]

4.1 ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0))

main.m

```
close all
clear all
clc

% Environment Varible
```

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

```matlab
M = 16
Nt = 4
Nr = 4
NumberIteration = 10^4;

% Simulation
LengthBitSequence = Nt * log2(M); % log2(M) bits per signal
LengthSignalSequence = Nt;

% EbN0_dB = 0:5:25;
% EbN0 = db2pow(EbN0_dB);
%
% EsN0 = EbN0 * log2(M);
% EsN0_db = pow2db(EsN0);

EsN0_dB = 0:5:25;
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));

BitErrorCount_MMSE = zeros(1, length(EsN0_dB));
SignalErrorCount_MMSE = zeros(1, length(EsN0_dB));

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

FivePercent = ceil(NumberIteration/20);
for iTotal = 1 : NumberIteration
    if mod(iTotal-100, FivePercent)==0
        tic
    end
    % Bit Generation
    SignalSequence = randi([0 M-1], Nt, 1);
    SignalBinary = de2bi(SignalSequence, log2(M), 'left-msb');
    SymbolSequence = qammod(SignalSequence, M) / NormalizationFactor;

    NoiseSequence = (randn(Nr, 1) + 1j * randn(Nr, 1)) / sqrt(2); % Noise (n) Generation
    H = (randn(Nr, Nt) + 1j * randn(Nr, Nt)) ./ sqrt(2); % Receiver x Transmitter
    for indx_EbN0 = 1 : length(EsN0)
        % Received Signal (y = hs + n) Generation
        ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0)); % log2(M)x1 matrix

        % MLD Receiver
        [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_mld(ReceivedSymbolSequence, SignalSequence, SignalBinary,  M, H);
        BitErrorCount_MLD(indx_EbN0) = BitErrorCount_MLD(indx_EbN0) +
```

```
            BitErrorCount_tmp;
        SignalErrorCount_MLD(indx_EbN0) = SignalErrorCount_MLD(indx_EbN0)
            + SignalErrorCount_tmp;

        % ZF Receiver
        [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_zf(
            ReceivedSymbolSequence, SignalSequence, SignalBinary, M, H);
        BitErrorCount_ZF(indx_EbN0) = BitErrorCount_ZF(indx_EbN0) +
            BitErrorCount_tmp;
        SignalErrorCount_ZF(indx_EbN0) = SignalErrorCount_ZF(indx_EbN0) +
            SignalErrorCount_tmp;

        % MMSE Receiver
        [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_mmse(
            ReceivedSymbolSequence, SignalSequence, SignalBinary, M, H,
            EsN0(indx_EbN0));
        BitErrorCount_MMSE(indx_EbN0) = BitErrorCount_MMSE(indx_EbN0) +
            BitErrorCount_tmp;
        SignalErrorCount_MMSE(indx_EbN0) = SignalErrorCount_MMSE(indx_EbN0
            ) + SignalErrorCount_tmp;
    end
    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

% Error Count to Ratio
SER_MLD = SignalErrorCount_MLD / (LengthSignalSequence * NumberIteration);
BER_MLD = BitErrorCount_MLD / (LengthBitSequence * NumberIteration);

SER_ZF = SignalErrorCount_ZF / (LengthSignalSequence * NumberIteration);
BER_ZF = BitErrorCount_ZF / (LengthBitSequence * NumberIteration);

SER_MMSE = SignalErrorCount_MMSE / (LengthSignalSequence * NumberIteration
    );
BER_MMSE = BitErrorCount_MMSE / (LengthBitSequence * NumberIteration);

% Plot
figure()

semilogy(EsN0_dB, BER_ZF, 'b.--', 'MarkerSize', 15);
hold on
semilogy(EsN0_dB, BER_MMSE, '.--', 'Color', "#4DBEEE", 'MarkerSize', 15);
semilogy(EsN0_dB, BER_MLD, '.--','Color', '#D95319', 'MarkerSize', 15);
ylabel('BER');
title(sprintf("BER for %d-QAM %dX%d MIMO", M, Nr, Nt));
grid on
legend('ZF (Rayleigh)', 'MMSE (Rayleigh)', 'MLD (Rayleigh)');
xlabel('Es/No [dB]');

```

```
figure()
semilogy(EsN0_dB, SER_ZF, 'b.--', 'MarkerSize', 15);
hold on
semilogy(EsN0_dB, SER_MMSE, '.--', 'Color', "#4DBEEE", 'MarkerSize', 15);
semilogy(EsN0_dB, SER_MLD, '.--', 'Color', '#D95319', 'MarkerSize', 15);
ylabel('SER');
title(sprintf("SER for %d-QAM %dX%d MIMO", M, Nr, Nt));
grid on
legend('ZF (Rayleigh)', 'MMSE (Rayleigh)', 'MLD (Rayleigh)');
xlabel('Es/No [dB]');
```

simulation_mld.m

```
function [BitErrorCount, SignalErrorCount] = simulate_mld(ReceivedSymbolSequence, SignalSequence, SignalBinary,  M, H)
    Nt = size(H,1);
    NormalizationFactor = sqrt(2/3*(M-1)*Nt);
    persistent Candidates
    if isempty(Candidates)
         Candidates = get_candidates(M, Nt) / NormalizationFactor;
    end
    % results in Nt x M^Nt, each column representing each candidate symbol combination
    EuclideanDistance = abs(ReceivedSymbolSequence * ones(1,M^Nt) - H*Candidates).^2;
    [val, idx] = min(sum(EuclideanDistance, 1));

    DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'),log2(M),[])';
    DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');

    BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
    SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
end

function Candidates = get_candidates(M, Nt)
    AllNumbers = de2bi([0:M^Nt-1], Nt*log2(M), 'left-msb');
    Candidates = zeros(M^Nt, Nt);
    for ii = 1 : M^Nt
        for jj = 1 : Nt
            Candidates(ii,jj) = bi2de(AllNumbers(ii,log2(M)*(jj-1)+1:log2(M)*jj), 'left-msb');
        end
    end
    Candidates = qammod(Candidates',M);
end
```

simulation_zf.m

```
function [BitErrorCount, SignalErrorCount] = simulate_zf(ReceivedSymbolSequence, SignalSequence, SignalBinary,  M, H)
    Nt = size(H,1);
    NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt
    w_zf = pinv(H); % pinv(H) = inv(H' * H) * H'
```

```
    DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence; % Detection (Zero-Forcing: y / h)

    DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF*NormalizationFactor, M); % Detection
    DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');

    BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
    SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');
end
```

simulation_mmse.m

```
function [BitErrorCount, SignalErrorCount] = simulate_mmse(ReceivedSymbolSequence, SignalSequence, SignalBinary,  M, H, EsN0)
    Nt = size(H,1);
    NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt

    w_mmse = NormalizationFactor * inv(H' * H + Nt / EsN0 * eye(Nt)) * H';
    DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence;

    DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M);
    DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-msb');

    BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
    SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all');
end
```

4.2 ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence * sqrt(1 / EsN0(indx_EbN0))

main.m

```
```

simulation_mld.m

```
```

simulation_zf.m

```
```

simulation_mmse.m

```
```