

Simulating ZF, MMSE, MLD Receivers in a Rayleigh Fading, MIMO Environment

Project #2

Intelligent Communication Systems (ICS) Lab.
노용재

Winter Intern Seminar (2023-1)

Contents

1	Implementation	1
1.1	ZF(Zero-forcing)	2
1.2	MMSE(Minimum Mean Square Error)	3
1.3	MLD(Maximum Likelihood Detection)	4
1.3.1	Creating All Possible Signal Combinations	4
1.3.2	Solving For Minimum Euclidean Distance	5
2	결과 및 분석	6
2.1	Simulation Result	6
3	미해결 & 추가연구 필요 내용	9
4	Entire Code	9

1 Implementation

다음의 조건을 만족하는 환경에 해당한다.

$$N_t \leq N_r \quad (1)$$

```
1 placeholder = 4
2
3 % Environment Variables
4 M = placeholder
5 Nt = placeholder
6 Nr =placeholder
7 NormalizationFactor = sqrt(2/3*(M-1)*Nt);
8
9 % Signal Generation
10 SignalSequence = randi([0 M-1], Nt, 1);
11 SignalBinary = de2bi(SignalSequence, log2(M), 'left-msb');
12 SymbolSequence = qammod(SignalSequence, M) / NormalizationFactor;
13
14 % Noise (n) Generation
15 NoiseSequence = (randn(Nt, 1) + 1j * randn(Nt, 1)) / sqrt(2);
```

```
16 H = (randn(Nr, Nt) + 1j * randn(Nr, Nt)) ./ sqrt(2);
```

Normalization Factor를 $\sqrt{\frac{2}{3}(M-1)N_t}$ 로 설정한 이유는 하나의 trasmitter가 $\frac{1}{N_t}W$ 의 전력을 갖도록 하기 위해서이다.

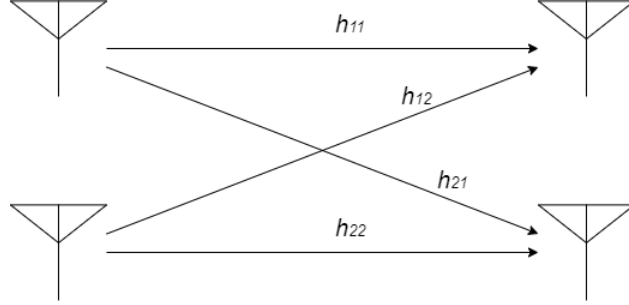


Figure 1

$$H = \begin{bmatrix} h_{11} & \dots & h_{1N_T} \\ \vdots & \ddots & \vdots \\ h_{NR1} & \dots & h_{NRN_T} \end{bmatrix} \quad (2)$$

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.

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

이때, W_{ZF} 를 H 의 right pseudo-inverse라고 할 수 있다.

Matlab에는 Moore Penrose Pseudo Inverse를 구할 수 있는 **pinv(H)** 함수가 존재한다.

$ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence/\sqrt{EsN_0}$ 의 경우:

```
1 NormalizationFactor = sqrt(2/3*(M-1) * Nt);
2
3 w_zf = pinv(H); % pinv(H) = inv(H' * H) * H'
4 DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence;
5
6 DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF*
   NormalizationFactor, M);
7 DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');
8
9 BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
10 SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');
```

$ReceivedSymbolSequence = \sqrt{EsN_0} * H * SymbolSequence + NoiseSequence$ 의 경우:

```

1 NormalizationFactor = sqrt(2/3*(M-1)*Nt);
2
3 w_zf = NormalizationFactor /sqrt(EsN0) * pinv(H); % pinv(H) = inv(H' * H)
   * H';
4 DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence;
5
6 DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF, M);
7 DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');
8
9 BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
10 SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');

```

1.2 MMSE(Minimum Mean Square Error)

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

$ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence / \sqrt{EsN_0}$ 의 경우:

```

1 NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt
2
3 w_mmse = NormalizationFactor * inv(H' * H + Nt / EsN0 * eye(Nt)) * H';
4 DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence;
5
6 DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M);
7 DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-
   msb');
8
9 BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
10 SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all')
   ;

```

$ReceivedSymbolSequence = \sqrt{EsN_0} * H * SymbolSequence + NoiseSequence$ 의 경우:

```

1 Nt = size(H,1);
2 NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt
3
4 w_mmse = NormalizationFactor / sqrt(EsN0) * inv(H' * H + Nt / EsN0 * eye(
   Nt)) * H';
5 DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence; % Detection
   (Zero-Forcing: y / h)
6
7 DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M); %
   Detection
8 DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-
   msb');
9
10 BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
11 SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all')
   ;

```

1.3 MLD(Maximum Likelihood Detection)

1.3.1 Creating All Possible Signal Combinations

```

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

```

*AllNumbers*는 각 열마다 0부터 $M^{N_t} - 1$ 를 이진수로 나타낸 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_t 개로 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_t 개의 signal로 만들 수 있는 M^{N_t} 개의 signal combination을 나타낸다.

1.3.2 Solving For Minimum Euclidean Distance

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

$ReceivedSymbolSequence = H * SymbolSequence + NoiseSequence / \sqrt{EsN0}$ 의 경우:

```

1 % 'EuclideanDistance' results in Nt x M^Nt, each column representing each
  candidate symbol combination
2 EuclideanDistance = abs(ReceivedSymbolSequence * ones(1,M^Nt) - H*
  Candidates).^2;
3 [val, idx] = min(sum(EuclideanDistance, 1));
4
5 DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'), log2(M)
  , []);
6 DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');
7
8 BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
9 SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
```

$ReceivedSymbolSequence = \sqrt{EsN0} * H * SymbolSequence + NoiseSequence$ 의 경우:

```

1 % 'EuclideanDistance' results in Nt x M^Nt, each column representing each
  candidate symbol combination
2 EuclideanDistance = abs(ReceivedSymbolSequence/sqrt(EsN0) * ones(1,M^Nt) -
  H*Candidates).^2;
3 [val, idx] = min(sum(EuclideanDistance, 1));
4
5 DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'), log2(M)
  , []);
6 DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');
7
8 BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
9 SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
```

$$ReceivedSymbolSequence = [y_1 \dots y_{N_R}]^T \quad (6)$$

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

2 결과 및 분석

2.1 Simulation Result

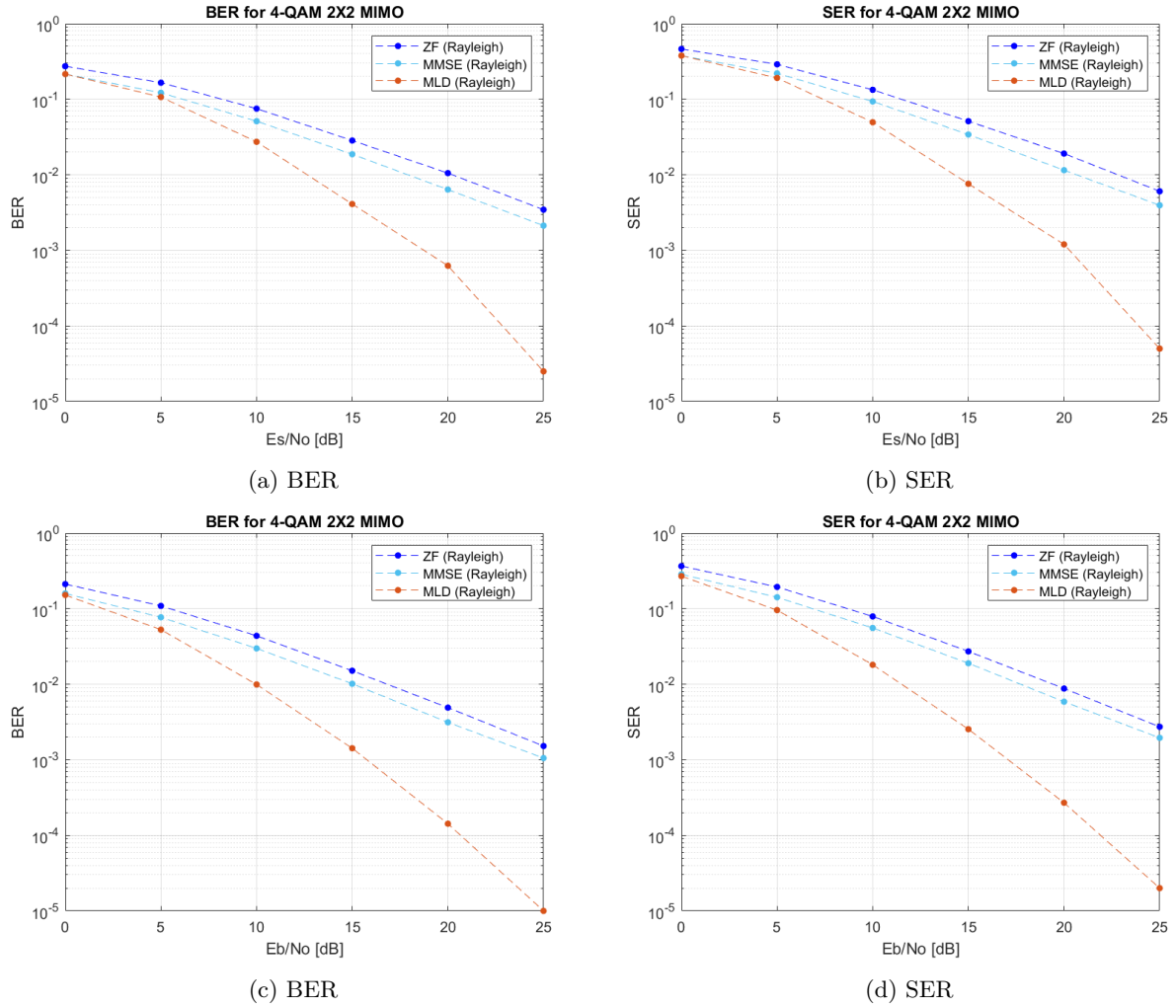
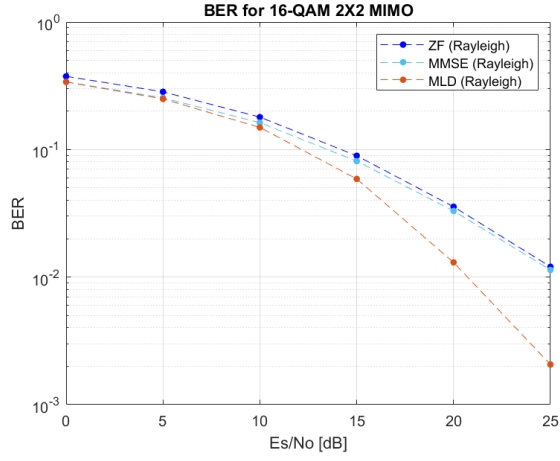
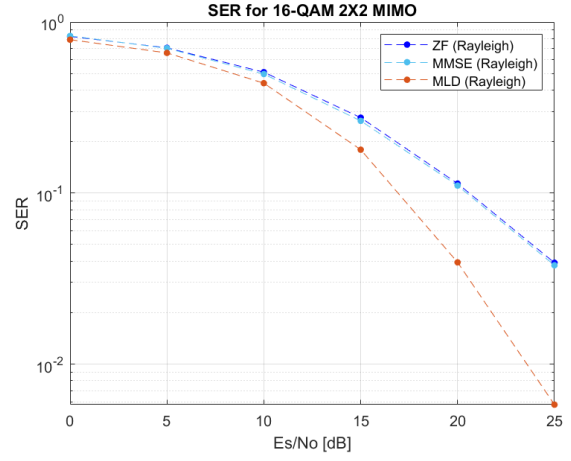


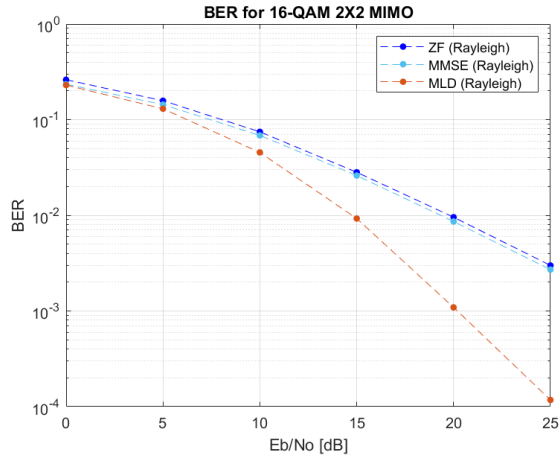
Figure 3: 4-QAM 2×2 MIMO



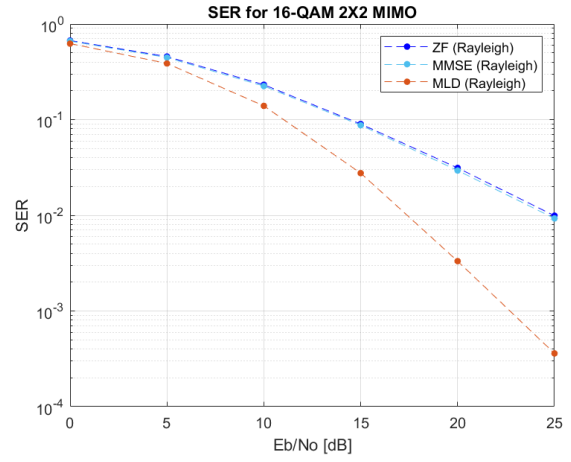
(a) BER



(b) SER

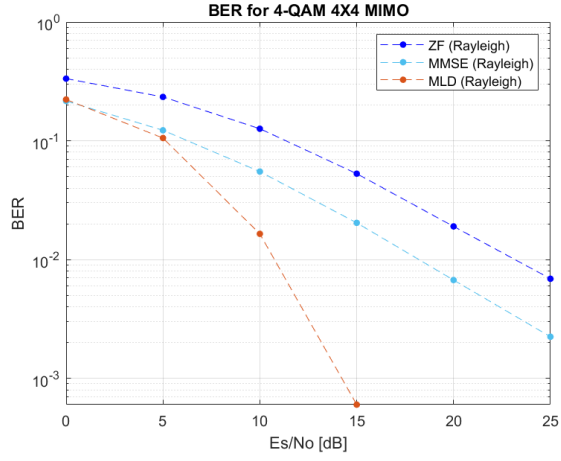


(c) BER

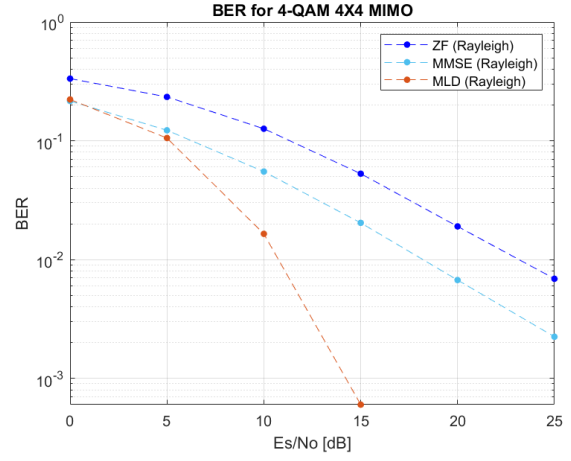


(d) SER

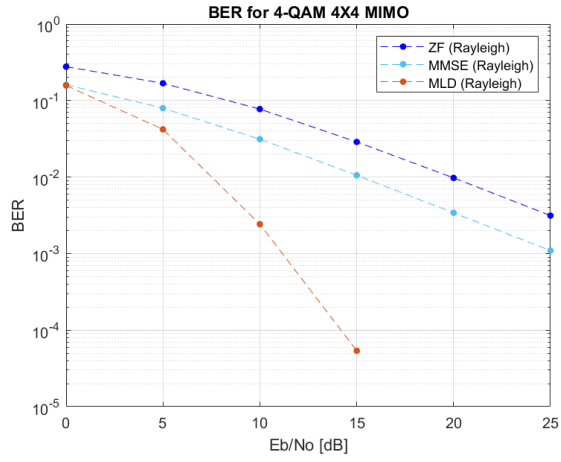
Figure 4: 16-QAM 2×2 MIMO



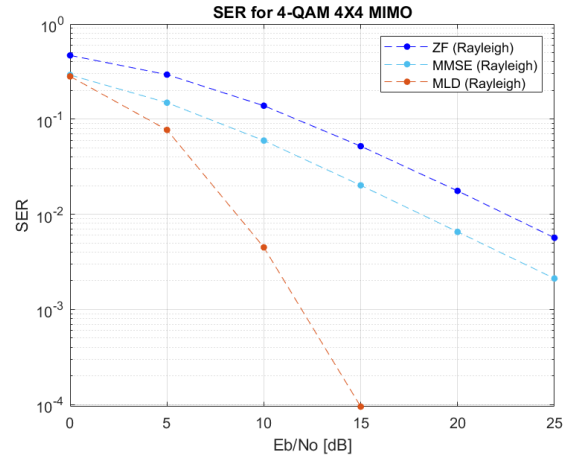
(a) BER



(b) SER

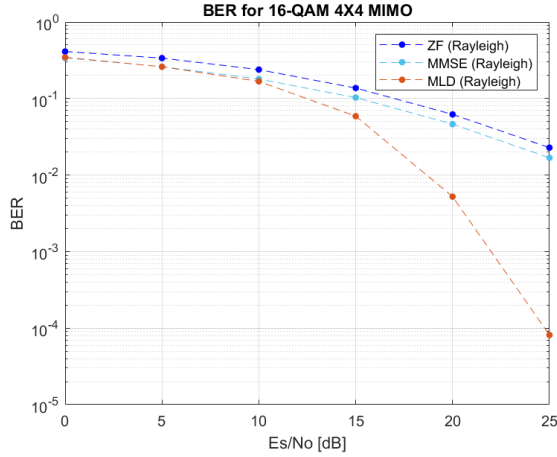


(c) BER

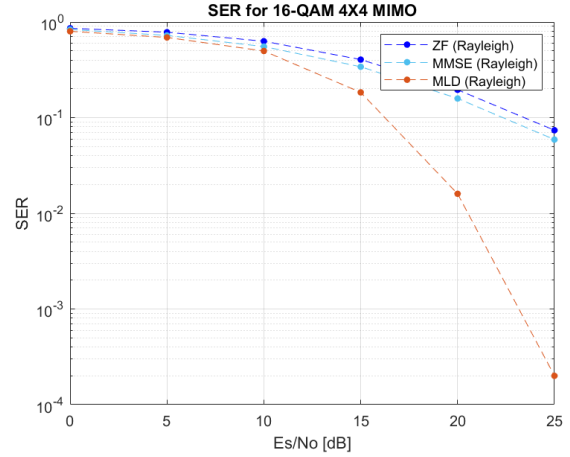


(d) SER

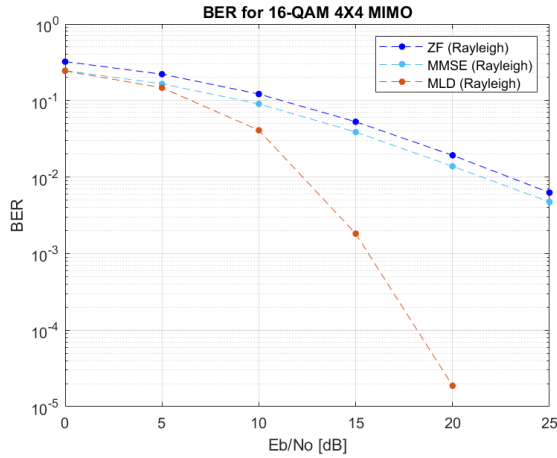
Figure 5: 4-QAM 4×4 MIMO



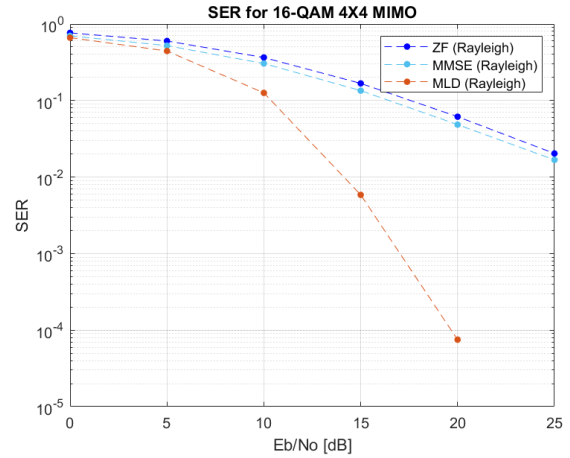
(a) BER



(b) SER



(c) BER



(d) SER

Figure 6: 16-QAM 4×4 MIMO

3 미해결 & 추가연구 필요 내용

- ZF receiver에서 $W_{ZF} = (H^T H)^{-1} H^T$ 와 같이 일반 transpose를 사용하여 W_{ZF} 를 구한다 하더라도 결과는 동일하다.
그렇다면 왜 Zero-forcing receiver에서 transpose가 아닌 complex transpose를 사용한건가?
Complex transpose를 사용함으로써 어떠한 이점이 있는가?

4 Entire Code ¹

$$ReceivedSymbolSequence = \sqrt{EsN0} * H * SymbolSequence + NoiseSequence \quad (7)$$

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

main.m

```
1 close all
2 clear all
3 clc
4
5 % Environment Variable
6 M = 4
7 Nt = 2
8 Nr = 2
9 NumberIteration = 10^3;
10
11 % Simulation
12 LengthBitSequence = Nt * log2(M); % log2(M) bits per signal
13 LengthSignalSequence = Nt;
14
15 % EbN0_dB = 0:5:25;
16 % EbN0 = db2pow(EbN0_dB);
17 %
18 % EsN0 = EbN0 * log2(M);
19 % EsN0_db = pow2db(EsN0);
20
21 EsN0_dB = 0:5:25;
22 EsN0 = db2pow(EsN0_dB);
23
24 EbN0 = EsN0 / log2(M);
25 EbN0_dB = pow2db(EbN0);
26
27 BitErrorCount_ZF = zeros(1, length(EsN0_dB));
28 SignalErrorCount_ZF = zeros(1, length(EsN0_dB));
29 BitErrorCount_MLD = zeros(1, length(EsN0_dB));
30 SignalErrorCount_MLD = zeros(1, length(EsN0_dB));
31
32 BitErrorCount_MMSE = zeros(1, length(EsN0_dB));
33 SignalErrorCount_MMSE = zeros(1, length(EsN0_dB));
34
35 NormalizationFactor = sqrt(2/3*(M-1)*Nt);
36
37 FivePercent = ceil(NumberIteration/20);
38 for iTot = 1 : NumberIteration
39     if mod(iTot-100, FivePercent)==0
40         tic
41     end
42     % Bit Generation
43     SignalSequence = randi([0 M-1], Nt, 1);
44     SignalBinary = de2bi(SignalSequence, log2(M), 'left-msb');
45     SymbolSequence = qammod(SignalSequence, M) / NormalizationFactor;
46
47     NoiseSequence = (randn(Nr, 1) + 1j * randn(Nr, 1)) / sqrt(2); % Noise (n) Generation
48     H = (randn(Nr, Nt) + 1j * randn(Nr, Nt)) ./ sqrt(2); % Receiver x Transmitter
49     for indx_EbN0 = 1 : length(EsN0)
50         % Received Signal (y = hs + n) Generation
51         ReceivedSymbolSequence = sqrt(EsN0(indx_EbN0)) * H * SymbolSequence +
52             NoiseSequence; % log2(M)x1 matrix
53
54         % MLD Receiver
55         [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_mld(ReceivedSymbolSequence,
56             SignalSequence, SignalBinary, M, H, EsN0(indx_EbN0));
57         BitErrorCount_MLD(indx_EbN0) = BitErrorCount_MLD(indx_EbN0) + BitErrorCount_tmp;
58         SignalErrorCount_MLD(indx_EbN0) = SignalErrorCount_MLD(indx_EbN0) +
59             SignalErrorCount_tmp;
60
61         % ZF Receiver
```

```

59     [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_zf(ReceivedSymbolSequence,
60         SignalSequence, SignalBinary, M, H, EsNO(indx_EbNO));
61     BitErrorCount_ZF(indx_EbNO) = BitErrorCount_ZF(indx_EbNO) + BitErrorCount_tmp;
62     SignalErrorCount_ZF(indx_EbNO) = SignalErrorCount_ZF(indx_EbNO) +
63         SignalErrorCount_tmp;
64
65     % MMSE Receiver
66     [BitErrorCount_tmp, SignalErrorCount_tmp] = simulate_mmse(ReceivedSymbolSequence
67         , SignalSequence, SignalBinary, M, H, EsNO(indx_EbNO));
68     BitErrorCount_MMSE(indx_EbNO) = BitErrorCount_MMSE(indx_EbNO) +
69         BitErrorCount_tmp;
70     SignalErrorCount_MMSE(indx_EbNO) = SignalErrorCount_MMSE(indx_EbNO) +
71         SignalErrorCount_tmp;
72
73     end
74
75     % Progress Keeping
76     if mod(iTotal-100, FivePercent)==0
77         ElapsedTime = toc;
78         EstimatedTime = (NumberIteration-iTotal)*ElapsedTime;
79         disp(sprintf("%d%%, estimated wait time %d minutes %d seconds", round(iTotal/
80             NumberIteration*100), floor(EstimatedTime/60), floor(mod(EstimatedTime, 60)))
81             )
82     end
83 end
84
85 % Error Count to Ratio
86 SER_MLD = SignalErrorCount_MLD / (LengthSignalSequence * NumberIteration);
87 BER_MLD = BitErrorCount_MLD / (LengthBitSequence * NumberIteration);
88
89 SER_ZF = SignalErrorCount_ZF / (LengthSignalSequence * NumberIteration);
90 BER_ZF = BitErrorCount_ZF / (LengthBitSequence * NumberIteration);
91
92 SER_MMSE = SignalErrorCount_MMSE / (LengthSignalSequence * NumberIteration);
93 BER_MMSE = BitErrorCount_MMSE / (LengthBitSequence * NumberIteration);
94
95 % Plot
96 figure()
97
98 semilogy(EsNO_dB, BER_ZF, 'b.--', 'MarkerSize', 15);
99 hold on
100 semilogy(EsNO_dB, BER_MMSE, 'b.--', 'Color', '#4DBEEE', 'MarkerSize', 15);
101 semilogy(EsNO_dB, BER_MLD, 'b.--', 'Color', '#D95319', 'MarkerSize', 15);
102 ylabel('BER');
103 title(sprintf("BER for %d-QAM %dX%d MIMO", M, Nr, Nt));
104 grid on
105 legend('ZF (Rayleigh)', 'MMSE (Rayleigh)', 'MLD (Rayleigh)');
106 xlabel('Es/No [dB]');
107
108 figure()
109 semilogy(EsNO_dB, SER_ZF, 'b.--', 'MarkerSize', 15);
110 hold on
111 semilogy(EsNO_dB, SER_MMSE, 'b.--', 'Color', '#4DBEEE', 'MarkerSize', 15);
112 semilogy(EsNO_dB, SER_MLD, 'b.--', 'Color', '#D95319', 'MarkerSize', 15);
113 ylabel('SER');
114 title(sprintf("SER for %d-QAM %dX%d MIMO", M, Nr, Nt));
115 grid on
116 legend('ZF (Rayleigh)', 'MMSE (Rayleigh)', 'MLD (Rayleigh)');
117 xlabel('Es/No [dB]');

```

simulation_mld.m

```
1 function [BitErrorCount, SignalErrorCount] = simulate_mld(ReceivedSymbolSequence,
2   SignalSequence, SignalBinary, M, H, EsN0)
3   Nt = size(H,1);
4   NormalizationFactor = sqrt(2/3*(M-1)*Nt);
5
6   % Candidates gets processed only once
7   persistent Candidates
8   if isempty(Candidates)
9       Candidates = get_candidates(M, Nt) / NormalizationFactor;
10  end
11
12  % 'EuclideanDistance' results in Nt x M^Nt, each column representing each candidate
13  % symbol combination
14  EuclideanDistance = abs(ReceivedSymbolSequence/sqrt(EsN0) * ones(1,M^Nt) - H*
15  Candidates).^2;
16  [val, idx] = min(sum(EuclideanDistance, 1));
17
18  DetectedBinary_MLD = reshape(de2bi(idx-1, log2(M)*Nt, 'left-msb'),log2(M),[]);
19  DetectedSequence_MLD = bi2de(DetectedBinary_MLD, 'left-msb');
20
21  BitErrorCount = sum(SignalBinary~=DetectedBinary_MLD, 'all');
22  SignalErrorCount = sum(SignalSequence~=DetectedSequence_MLD, 'all');
23 end
24
25 function Candidates = get_candidates(M, Nt)
26   AllNumbers = de2bi([0:M^Nt-1], Nt*log2(M), 'left-msb');
27   Candidates = zeros(M^Nt, Nt);
28   for ii = 1 : M^Nt
29       for jj = 1 : Nt
30           Candidates(ii,jj) = bi2de(AllNumbers(ii,log2(M)*(jj-1)+1:log2(M)*jj), 'left-
31           msb');
```

simulation_zf.m

```
1 function [BitErrorCount, SignalErrorCount] = simulate_zf(ReceivedSymbolSequence,
2   SignalSequence, SignalBinary, M, H, EsN0)
3   Nt = size(H,1);
4   NormalizationFactor = sqrt(2/3*(M-1)*Nt); % size(H,1) = Nt
5   w_zf = (NormalizationFactor) / sqrt(EsN0) * pinv(H); % pinv(H) = inv(H' * H) * H
6   DetectedSymbolSequence_ZF = w_zf * ReceivedSymbolSequence; % Detection (Zero-
7   Forcing: y / h)
8   DetectedSignalSequence_ZF = qamdemod(DetectedSymbolSequence_ZF, M); % Detection
9   DetectedBinary_ZF = de2bi(DetectedSignalSequence_ZF, log2(M), 'left-msb');
10  BitErrorCount = sum(SignalBinary~=DetectedBinary_ZF, 'all');
11  SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_ZF, 'all');
12 end
```

simulation_mmse.m

```
1 function [BitErrorCount, SignalErrorCount] = simulate_mmse(ReceivedSymbolSequence,
2   SignalSequence, SignalBinary, M, H, EsN0)
3   Nt = size(H,1);
4   NormalizationFactor = sqrt(2/3*(M-1) * Nt); % size(H,1) = Nt
5   w_mmse = NormalizationFactor / sqrt(EsN0) * inv(H' * H + Nt / EsN0 * eye(Nt)) * H';
6   DetectedSymbolSequence_MMSE = w_mmse * ReceivedSymbolSequence; % Detection (Zero-
7   Forcing: y / h)
8   DetectedSignalSequence_MMSE = qamdemod(DetectedSymbolSequence_MMSE, M); % Detection
9   DetectedBinary_MMSE = de2bi(DetectedSignalSequence_MMSE, log2(M), 'left-msb');
10  BitErrorCount = sum(SignalBinary~=DetectedBinary_MMSE, 'all');
11  SignalErrorCount = sum(SignalSequence~=DetectedSignalSequence_MMSE, 'all');
12 end
```