Lab 2b - 4x4 MIMO System

Pranavi Boyalakuntla
Jonathan Kelley

I. Introduction

Multiple antennas help support multiple users sending streams of data in the same frequency band. We see this with technologies like IoT and bluetooth where multiple devices frequently communicate in close range. Systems like these are called "multiple input multiple output" (MIMO) systems.

In this lab, we were given a simulated 4x4 MIMO channel in MATLAB to pass constructed data through. If we were to implement this in hardware, this system might look like Fig. 1.

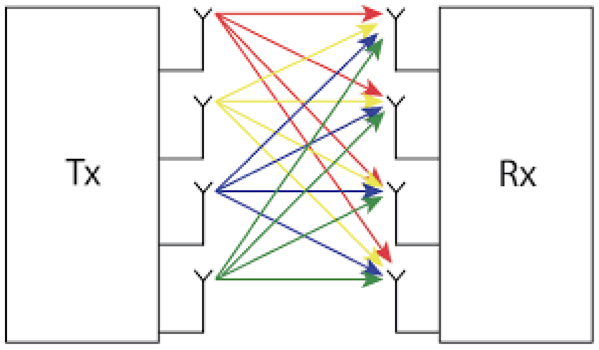

Fig. 1: A visual representation of a 4x4 MIMO system.

In Fig. 1, we can see that each of the receivers is "hearing" a combination of all of the signals sent in this system. We processed the received data at each receiver for the original signal sent. In order to do this, we considered two cases.

Case 1: We assume we do not know the channels between transmitter and receiver.

Case 2: We assume we do know the channels between transmitter and receiver.

We assumed a flat fading and timing synchronized model. A flat fading model assumes that the signal fades evenly across all frequencies. A timing synchronized model assumes that the cosine signals at the transmitters and receivers are synchronized. These assumptions are reasonable because we are simulating the channel.

II. Process

There are 16 channels in a 4x4 MIMO system because each transmitter sends signals to each receiver. We can send training signals through the channels and watch how the signal changes once it is received to estimate channel responses. To implement this, we sent a known signal from one transmitter at a time and compared it to what was received on all four receivers.

Eqn. 1 shows the calculation of the channel response, h, by dividing the received signal, y, with the sent signal, x.

$$h = y./x \tag{1}$$

From the 16 channels we estimated, we formed the H matrix. Eqn. 2 shows the structure of the H matrix where the rows increment by transmitter and the columns increment by receiver.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} \tag{2}$$

Eqn. 3 and Eqn. 4 illustrate how the channels changed the transmitted signals, x, to the received signals, y.

$$y = Hx \tag{3}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{4}$$

A. Accounting for delay

Taking the cross correlation of the received signal with the sent signal allows us to find the point where the two signals are most correlated. We found this point to be the start of the received signal, so we did not account for any time delay.

B. MMSE Decoder

Once we have found the channel properties through the training signal signal approach discussed earlier, we can implement the Minimum Mean Square Error (MMSE) decoder algorithm. The MMSE decoder avoids causing noise amplification if the minimum singular value of H is too small.

To form our guess as to what was originally sent, $\hat{x}$, we apply a weight vector, w, to the received signal, y, in Eqn. 5. The MMSE decoder algorithm can help find this weight vector.

$$\hat{x} = w^H y \tag{5}$$

The condition number is the ratio between the largest and smallest singular values in the H matrix. Linear receivers (including MMSE Receivers) are sensitive to large condition numbers. In order to account for this, we can account for the variance when nothing is being transmitted. This is known as the regularization term.

Our goal with the MMSE receiver is to minimize the MMSE term given by the formula in Eqn. 6 assuming the variance, λ, is greater than 0 to form our guess as to what was originally sent, $\hat{x}$.

$$\hat{\mathbf{x}} = \text{argmin}_{\mathbf{x}} \, \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2 + \lambda\|\mathbf{x}\|^2 \tag{6}$$

The receiver implementation can be simplified to the matrix product in Eqn. 7.

$$\hat{x} = \left(\boldsymbol{H}^H + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{H}^H \boldsymbol{y} \tag{7}$$

Eqn. 7 looks suprisingly like Eqn. 5. We can define the weights matrix , w, as shown in Eqn. 8 where I is the identity matrix.

$$w^H = \left(\boldsymbol{H}^H + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{H}^H \tag{8}$$

We can solve for $\hat{x}$ with the calculated weights matrix using Eqn. 5.

C. Explaining Singular Value Decomposition (SVD)

Let $\vec{v_1}$ and $\vec{v_2}$ define the basis for a space. This means that these vectors are orthogonal to each other and unit vectors.

Let's apply a matrix transformation M to each of these space basis vectors as shown in Eqn. 9.

$$\begin{aligned} \vec{V_1} &= M\vec{v_1} \\ \vec{V_2} &= M\vec{v_2} \end{aligned} \tag{9}$$

By the definition of a vector, we can say that $\vec{V_1}$ and $\vec{V_2}$ are equal to some unit vector with a scaling factor applied to it as shown in Eqn. 10.

$$\begin{aligned} \vec{V_1} &= M\vec{v_1} = \vec{u_1}\sigma_1 \\ \vec{V_2} &= M\vec{v_2} = \vec{u_2}\sigma_2 \end{aligned} \tag{10}$$

Let there be a vector $\vec{x}$ that can be broken down into $\vec{v_1}$ and $\vec{v_2}$ directional components are shown in Eqn.11.

$$\vec{x} = (\vec{x} \cdot \vec{v_1})\vec{v_1} + (\vec{x} \cdot \vec{v_2})\vec{v_2} \tag{11}$$

Let's left multiply both sides by Mas shown in Eqn. 12.

$$M\vec{x} = (\vec{x} \cdot \vec{v_1})M\vec{v_1} + (\vec{x} \cdot \vec{v_2})M\vec{v_2} \tag{12}$$

We simplify further as shown in Eqn. 13.

$$M\vec{x} = (\vec{x} \cdot \vec{v_1})\vec{u_1}\sigma_1 + (\vec{x} \cdot \vec{v_2})\vec{u_1}\sigma_2 \tag{13}$$

Since the dot product is commutative and the terms are scalar, we perform the following manipulations shown in Eqns. 14.

$$\begin{aligned} M\vec{x} &= \vec{u_1}\sigma_1\vec{v_1^T}\vec{x} + \vec{u_2}\sigma_2\vec{v_2^T}\vec{x} \\ M &= \vec{u_1}\sigma_1\vec{v_1^T} + \vec{u_2}\sigma_2\vec{v_2^T} \end{aligned} \tag{14}$$

In Eqn. 15, we can combine terms and come to the final equation for SVD. If we were to apply $M$to any vector, it would first be rotated to the original space basis using V. Then it would be scaled using diagonal matrix, Σ. Lastly, it would be rotated again using matrix U.

$$\begin{aligned} M &= \begin{bmatrix} \vec{u_1} & \vec{u_2} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} \vec{v_1^T} \\ \vec{v_2^T} \end{bmatrix} \\ \text{svd}(M) &= U\Sigma V^T \end{aligned} \tag{15}$$

D. Implementing SVD

If we know the channel properties before transmitting, we can use SVD to send and receive data. We start by applying the SVD algorithm to the channel matrix, H, as shown in Eqn. 16.

$$\begin{aligned} y &= Hx \\ y &= U\Sigma V^T x \end{aligned} \tag{16}$$

Both $U$and $V^T$are rotation matrices. In the case of transmitting data over air, V^T rotates to the radio channel path coordinate system and Urotates to the antenna coordinate system. Σ is a diagonal matrix where each value corresponds to a significant radio channel path.

As the signal is being sent from the transmitter, it is being rotated by V^T. During transmission, the signal is being scaled by Σ. As the signal is being received, it is being rotated by U. We can use this property to our advantage as we know how different stages of the transmission process change the signal.

We can precode the signal with V as shown in Eqn. 17.

$$\begin{aligned} y &= U\Sigma V^T(Vx) \\ y &= U\Sigma I x \\ y &= U\Sigma x \end{aligned} \tag{17}$$

If we left multiply both sides with U^T we are left with Σx which is just a scaled version of x as shown in Eqn. 18. This tells us that we can left multiply the received signal with the transpose of the antenna coordinate system to get a scaled version of our original signal. In simpler terms, we are "tricking" our channel into sending information in the initial coordinate system and then undoing any coordinate system rotations done by the receiver.

$$\begin{aligned} U^T y &= U^T U\Sigma x \\ U^T y &= I\Sigma x \\ U^T y &= \Sigma x \end{aligned} \tag{18}$$

III. Implementation and Results

Fig. 2 shows the layout of the MMSE decoder implementation in software.

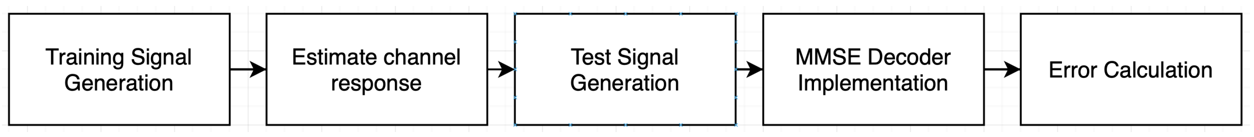
Training Signal Generation
Estimate channel response
Test Signal Generation
MMSE Decoder Implementation
Error Calculation

Fig. 2: Software structure for the MMSE decoder implementation.

To generate the training signal we used 600 characters of generic "Lorem Ipsum Dolor" text. This test data was converted from ASCII to binary. This was then sent across the channels to estimate the channel responses in the method described earlier.

To generate the test signal where all four transmitters would be sending signals at the same time, we used seperate generic text found on the internet. We converted this to binary. Per intial specifications, we started with a pulse width of 40 and saw no error. To increase the chances of catching error, we sent the binary with a pulse width of 10,000 samples per bit. Overall, we sent 10M samples on each channel, for a total of 40M samples.

We, then, sent the signal through the channel and implemented an MMSE decoder. Once the data was received, we pulled the bits to low or high and calculated the final error.

Fig. 3 shows the layout of the SVD receiver in software.

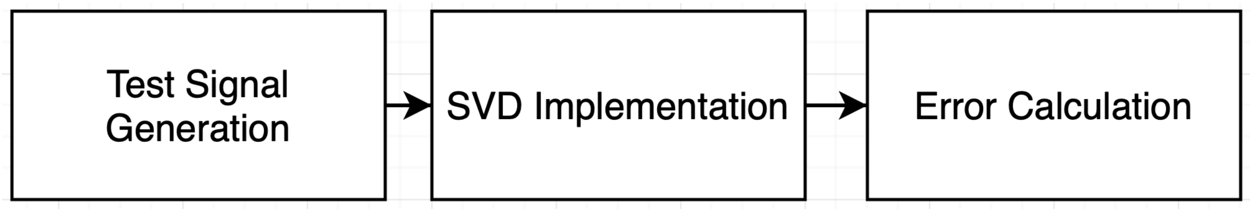

Fig. 3: Software structure for the SVD receiver.

We generated the test signal for the SVD receiver implementation in the same way as the MMSE decoder implementation. Then, we precoded the signal, sent it through the channel, and processed the signal after. Finally, we calculated the error.

Let's look at the results from the MMSE decoder first. In Fig. 4, you can see the first 50 samples of the sent and received signals on all transmitters and receivers before any post-processing was applied. The left y-axes show the scale of the received signal and the right y-axes show the scale of the sent signal. There is a significant loss in power through transmission.

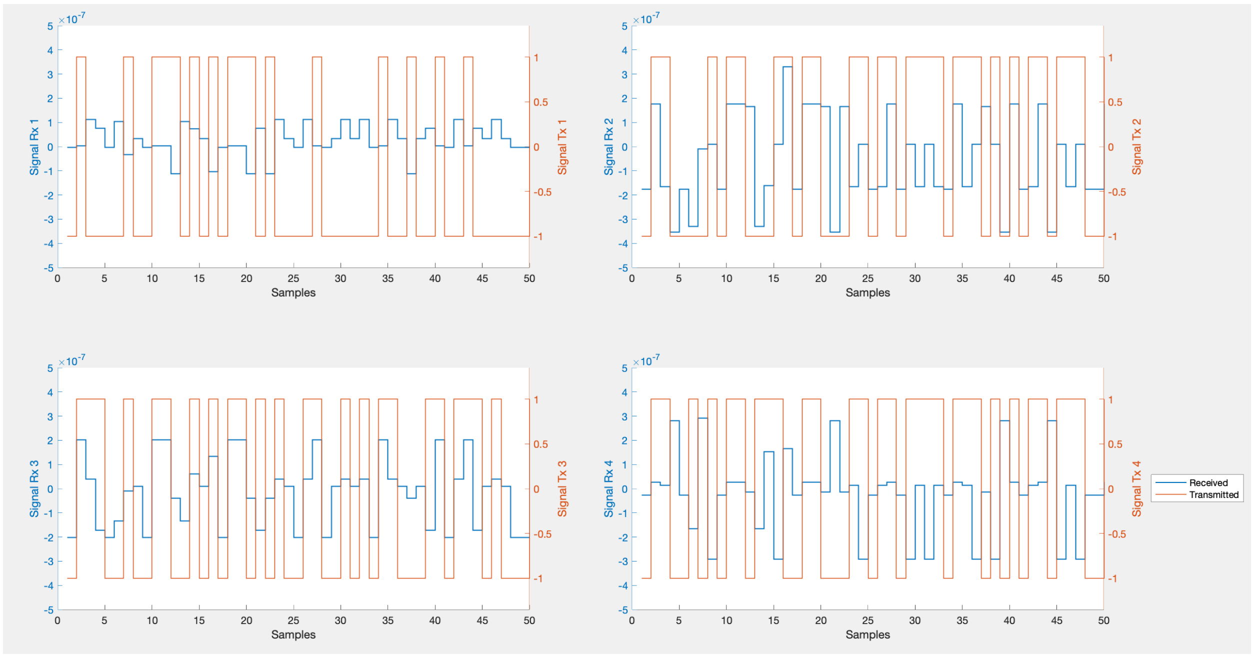

Fig. 4: Sent and received signals using the MMSE method without postprocessing.

In Fig. 5, you can see the first 50 samples of the sent and received signals on all transmitters and receivers after the MMSE receiver was applied. There was a 0% error rate with 40M samples. Using MATLAB's builtin SNR function, the SNR calculated was 143 dB.

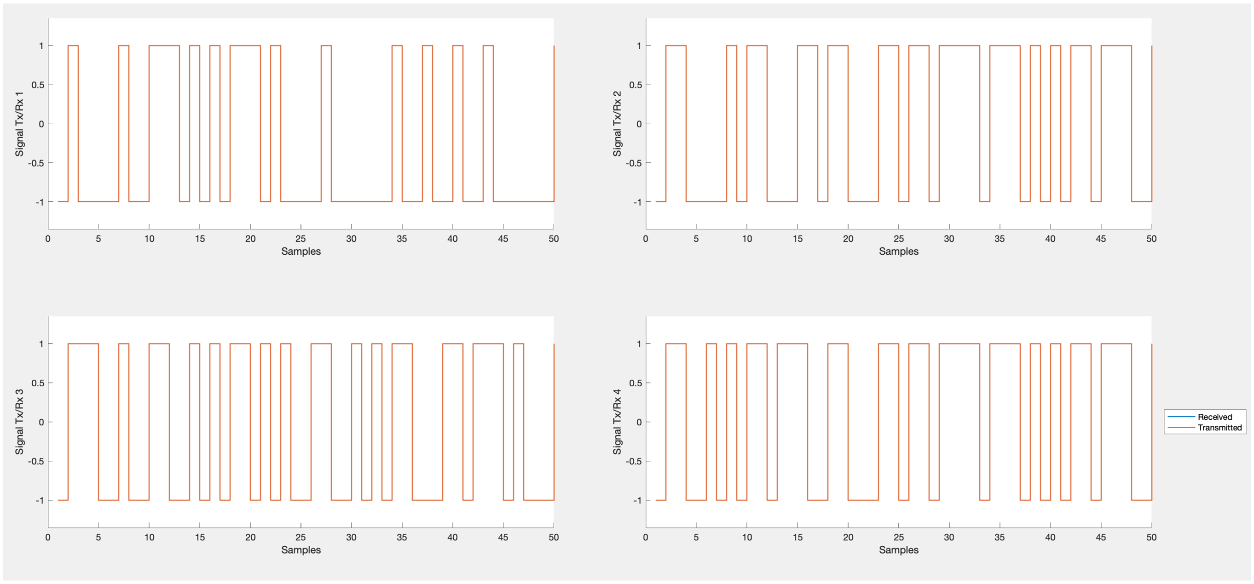

Fig. 5: Sent and received signals using the MMSE method with postprocessing.

Now, let's transition to the SVD receiver. In Fig. 6, you can see the first 50 samples of the sent and received signals on all transmitters and receivers before any post-processing was applied. The left y-axes show the scale of the received signal and the right y-axes show the scale of the sent signal. By this point, this signal has already been precoded with the V matrix.

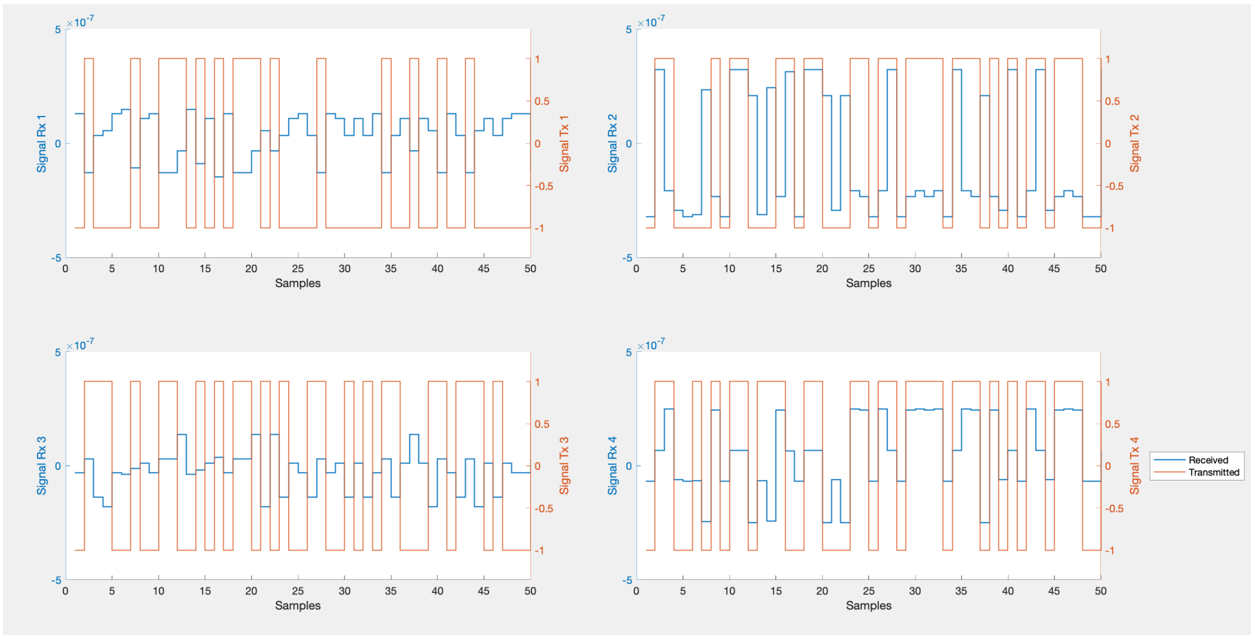

Fig. 6: Sent and received signals using the SVD method without postprocessing.

In Fig. 7, you can see the first 50 samples of the sent and received signals on all transmitters and receivers after the SVD receiver was applied. There was a 0% error rate with 40M samples. Using MATLAB's builtin SNR function, the SNR calculated was 139 dB.

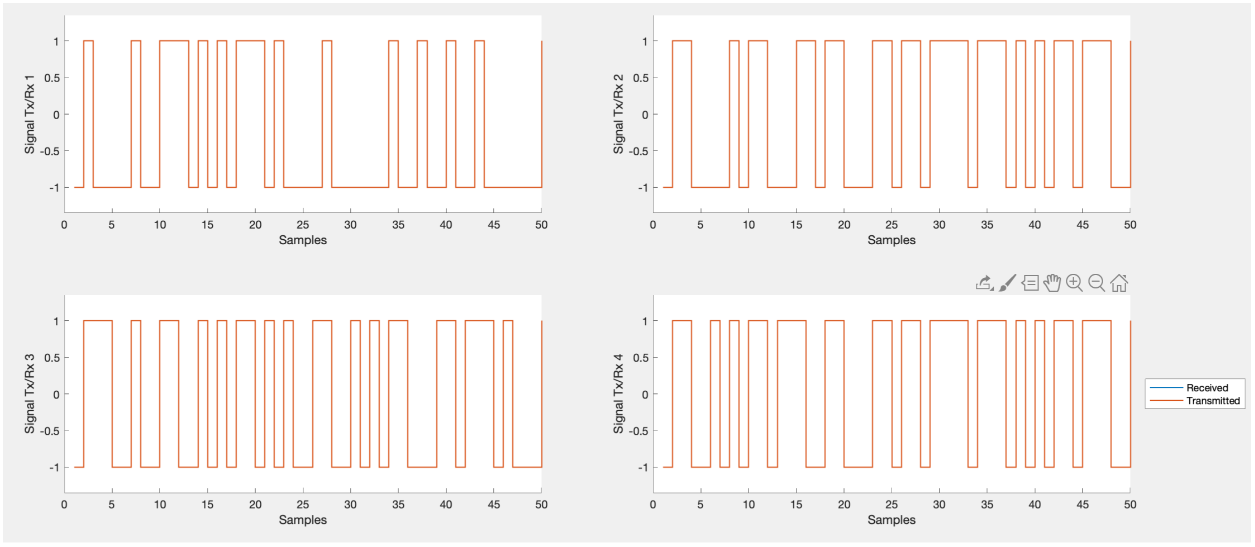

Fig. 7: Sent and received signals using the SVD method with postprocessing.

IV. Discussion and Next Steps

Our implementation for MMSE and SVD were both successful in this lab, both achieving an error rate of 0% on 40M samples with an SNR of around 140 dB. We believe that our extremely low error rate and high SNR are a result of low sample-size and near-perfect simulation conditions. While we cannot change the given `MIMOChannel4x4` function, we predict that the channel itself is not terribly noisy and that our "transmitter" is extremely powerful compared to the channel itself. If we "decrease" our transmitter power by 4 magnitudes, we begin to see more appropriate SNRs of around 60 dB.

V. Resources

[0] Code at: https://github.com/naviatolin/PoW-Labs/tree/master/Lab_2B

[2] MIMO I: spatial multiplexing and channel modeling

VI. Appendix

A. Main MMSE Implementation

```
%% Parameters
clear all;

pulse_width = 1;

data1 = 'But I must explain to you how all this mistaken idea of denouncing pleasure and p
raising pain was born and I will give you yy';
data2 = 'account of the system, and expound the actual teachings of the great explorer of
 the truth, the master-builder of human hapyy';
data3 = 'rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because
those who do not know how to pursue pleasyy';
data4 = 'encounter consequences that are extremely painful. Nor again is there anyone who
 loves or pursues or desires to obtain painyy';

train_data = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod temp
or incididunt ut labore et dolore magna aliqua. C';

data_binary = (reshape(dec2bin(train_data, 8).'-'0',1,[])') .* 2  - 1;
data_empty = zeros(strlength(train_data) * 8, 1);

%% Generate Data

empties = repelem(data_empty, pulse_width);
samples = repelem(data_binary, pulse_width);
```

```
y_empty = real(MIMOChannel4x4([empties, empties, empties, empties]));

y1 = real(MIMOChannel4x4([samples, empties, empties, empties]));
h1 = estimate_channel_response(samples', y1);

y2 = real(MIMOChannel4x4([empties, samples, empties, empties]));
h2 = estimate_channel_response(samples', y2);

y3 = real(MIMOChannel4x4([empties, empties, samples, empties]));
h3 = estimate_channel_response(samples', y3);

y4 = real(MIMOChannel4x4([empties, empties, empties, samples]));
h4 = estimate_channel_response(samples', y4);

H = [h1 h2 h3 h4];

%%
x_data1 = repelem(string_to_binvec(data1), pulse_width)';
x_data2 = repelem(string_to_binvec(data2), pulse_width)';
x_data3 = repelem(string_to_binvec(data3), pulse_width)';
x_data4 = repelem(string_to_binvec(data4), pulse_width)';
data_full = [
    x_data1
    x_data2
    x_data3
    x_data4
]';


y = real(MIMOChannel4x4(data_full));

%%

lambda = var(y_empty(1,:));
ident = lambda*eye(4,4);
w = H' * inv( ( H * H' +  ident ));

%%
x_hat_raw = w * y
x_hat = sign(round(x_hat_raw));

figure
hold on
stem(x_hat(1,1:50))
stem(x_data1(1:50))
legend('received', 'sent')
hold off
%%
error1 = calculate_error(x_hat(1,:), x_data1);
error2 = calculate_error(x_hat(2,:), x_data2);
error3 = calculate_error(x_hat(3,:), x_data3);
error4 = calculate_error(x_hat(4,:), x_data4);

%%
```

```
msg1 = binvec_to_string((x_hat(1,:) + 1) ./ 2);
msg2 = binvec_to_string((x_hat(2,:)+ 1) ./ 2);
msg3 = binvec_to_string((x_hat(3,:)+ 1) ./ 2);
msg4 = binvec_to_string((x_hat(4,:)+ 1) ./ 2);
msg = strcat(msg1', msg2', msg3', msg4')
```

B. Main SVD Implementation

```
%% Parameters
clear all;

pulse_width = 1;

data1 = 'But I must explain to you how all this mistaken idea of denouncing pleasure and p
raising pain was born and I will give you yy';
data2 = 'account of the system, and expound the actual teachings of the great explorer of
 the truth, the master-builder of human hapyy';
data3 = 'rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because
those who do not know how to pursue pleasyy';
data4 = 'encounter consequences that are extremely painful. Nor again is there anyone who
 loves or pursues or desires to obtain painyy';

train_data = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod temp
or incididunt ut labore et dolore magna aliqua. C';

data_binary = (reshape(dec2bin(train_data, 8).'-'0',1,[])') .* 2  - 1;
data_empty = zeros(strlength(train_data) * 8, 1);

%% Generate Data

empties = repelem(data_empty, pulse_width);
samples = repelem(data_binary, pulse_width);

y_empty = real(MIMOChannel4x4([empties, empties, empties, empties]));

y1 = real(MIMOChannel4x4([samples, empties, empties, empties]));
h1 = estimate_channel_response(samples', y1);

y2 = real(MIMOChannel4x4([empties, samples, empties, empties]));
h2 = estimate_channel_response(samples', y2);

y3 = real(MIMOChannel4x4([empties, empties, samples, empties]));
h3 = estimate_channel_response(samples', y3);

y4 = real(MIMOChannel4x4([empties, empties, empties, samples]));
h4 = estimate_channel_response(samples', y4);

H = [h1 h2 h3 h4];

%%
[U, S, V] = svd(H);
```

```
x_data1 = repelem(string_to_binvec(data1), pulse_width)';
x_data2 = repelem(string_to_binvec(data2), pulse_width)';
x_data3 = repelem(string_to_binvec(data3), pulse_width)';
x_data4 = repelem(string_to_binvec(data4), pulse_width)';
data_full = [
    x_data1
    x_data2
    x_data3
    x_data4
]';

precoded_data = V * data_full';

y = real(MIMOChannel4x4(precoded_data));

%% SVD Implementation

y_precoded_transpose = U' * y;
x_hat = sign(y_precoded_transpose);

figure
hold on
stem(x_hat(1,1:50))
stem(x_data1(1:50))
legend('received', 'sent')
hold off

%%
msg1 = binvec_to_string((x_hat(1,:) + 1) ./ 2);
msg2 = binvec_to_string((x_hat(2,:)+ 1) ./ 2);
msg3 = binvec_to_string((x_hat(3,:)+ 1) ./ 2);
msg4 = binvec_to_string((x_hat(4,:)+ 1) ./ 2);
msg = strcat(msg1', msg2', msg3', msg4')
```

C. Extra Functions

```
function OUT = string_to_binvec(S)
    OUT = (reshape(dec2bin(S, 8).'-'0',1,[])') .* 2  - 1;
end
```

```
function H = estimate_channel_response(samples, response)
% samples: n x 1
% response: n x 4


% Just one column of H
% Make sure to run this fn for each training vector to assemble a full 4x4
```

```
H = [
 mean(response(1,:) ./ samples)  % h11
 mean(response(2,:) ./ samples)  % h21
 mean(response(3,:) ./ samples)  % h31
 mean(response(4,:) ./ samples)  % h41
];

end
```

```
function str = binvec_to_string(binary)
    str = char(bin2dec(reshape(char(binary+'0'), 8,[]).'))
end
```

```
function error = calculate_error(data_hat, data)
%     error_num = sum(abs(reshape(data' - data_hat, 1, [])))
    error_num = sum(sign(abs(data - data_hat)));
    error = error_num/(length(data));
end
```