OFDM Project

Scenario A: Even and Odd Frequencies Split Between Users

Generating Transmitted Signals

Generate the random bits for each user.

```
clear;
bits1 = randi([0 1], 1, 1024);
bits2 = randi([0 1], 1, 1024);
```

We then assign users one and two with even and odd frequencies respectively. We then assign a 1024-point FFT $X_1[k]$, which is only non-zero at even indices, to user one. We similarily assign $X_2[k]$, which is even only at odd indices, to user two. Since MATLAB's arrays are one-indexed, in implementation, user one's values are assigned to odd indices, and user two's values are assigned to even indices.

```
% DFT coefficients for x1[n]
pairs1 = reshape(bits1, 2, []);
X1 = zeros(1, 1024);
for k = 1:size(pairs1, 2)
    if isequal(pairs1(:, k)', [0 0])
        X1(2*k-1) = -1 - 1j;
    elseif isequal(pairs1(:, k)', [0 1])
        X1(2*k-1) = -1 + 1j;
    elseif isequal(pairs1(:, k)', [1 1])
        X1(2*k-1) = 1 + 1j;
    else
        X1(2*k-1) = 1 - 1j;
    end
end

clear pairs1
% DFT coefficients for x2[n]
pairs2 = reshape(bits2, 2, []);
X2 = zeros(1, 1024);
for k = 1:size(pairs2, 2)
    if isequal(pairs2(:, k)', [0 0])
        X2(2*k) = -1 - 1j;
    elseif isequal(pairs2(:, k)', [0 1])
        X2(2*k) = -1 + 1j;
    elseif isequal(pairs2(:, k)', [1 1])
        X2(2*k) = 1 + 1j;
    else
        X2(2*k) = 1 - 1j;
    end
end
clear pairs2
```

Generating Received Signal

We first generate the channels $h_1[n]$ and $h_2[n]$, both of which are of duration 30.

```
dur = 30;
h1 = randn(1, dur) + randn(1, dur) * 1j;
h2 = randn(1, dur) + randn(1, dur) * 1j;
```

We then compute the 1024-point circular convolution to get the uncontaminated signal.

```
clean = ifft(X1.*fft(h1, 1024)+X2.*fft(h2, 1024), 1024);
```

Then, we add noise to the original signal.

```
v = randn(1, size(clean, 2))*sqrt(0.1);
```

Get the received signal

```
y = clean + v;
clear v;
clear clean;
```

Reconstructing the Original Signals

The non-zero values of $X_1[k]$ and $X_2[k]$ can only take on distinct 4 distinct values. We also know that for any k, only one of $X_1[k]$ and $X_2[k]$ can be non-zero. Therefore, for each value of $Y[k]$, if $k$is odd, then we choose the symbol $s_1 \in S$ that minimizes $|s_1H_1[k] - Y[k]|$ and assign that to $X_1[k]$. Similarily, if $k$is even, we choose the symbol $s_2 \in S$ that minimizes $|s_2H_2[k] - Y[k]|$ and assign that to $X_2[k]$.

We first define a set of symbols.

```
symb = [-1-1j, -1+1j, 1+1j, 1-1j];
```

We then iterate through with the algorithm described earlier.

```
% Define variables to hold reconstructed signals
X1_reconstructed = zeros(1, 1024);
X2_reconstructed = zeros(1, 1024);

% Compute the FFT of y[n], h_1[n], and h_2[n]
Y = fft(y, 1024);
H1 = fft(h1, 1024);
H2 = fft(h2, 1024);

% Iterate through received signal
for k=1:length(Y)
    if mod(k, 2) == 0
        min = Inf;
        for m = 1:length(symb)
            diff = abs(Y(k)-H2(k)*symb(m));
            if diff < min
                min = diff;
                X2_reconstructed(k) = symb(m);
            end
        end
    else
        min = Inf;
        for m = 1:length(symb)
            diff = abs(Y(k)-H1(k)*symb(m));
            if diff < min
                min = diff;
                X1_reconstructed(k) = symb(m);
            end
        end
    end
end
```

Identify Errors

Compare the extracted signals and original signals, and get the approximate probability of error.

```
num_diff_1 = sum(X1(1:2:end)~=X1_reconstructed(1:2:end));
num_diff_2 = sum(X2(2:2:end)~=X2_reconstructed(2:2:end));
```

When computing the error probability, in the frequency domain, we only have 512 non-zero values for X_1 and X_2 respectively, and they are spaced by zeros; these zeros are not considered in the decoding process. Therefore, we evaluate the probability over $2 \cdot 512 = 1024$ points.

```
Pe = (num_diff_1+num_diff_2) / 1024;
disp("Error Probability: " + Pe);
```

```
Error Probability: 0.33496
```

```
clear;
```

Noise Variance and Error Probability

We copy the above code into a function onto another file that takes in noise variance σ^2 and returns error probability P_e.

```
cd /Users/kyle/Library/'Mobile Documents'/com~apple~CloudDocs/'Course
Files'/'Fall 2024'/'ECE 161A'/MATLAB/'OFDM Project'/;
```

Define logarithmic range.

```
min_exp = -5;
max_exp = 1;
num_points = 500;
```

Get variances.

```
variances = logspace(min_exp, max_exp, num_points);
```

Generate channels.

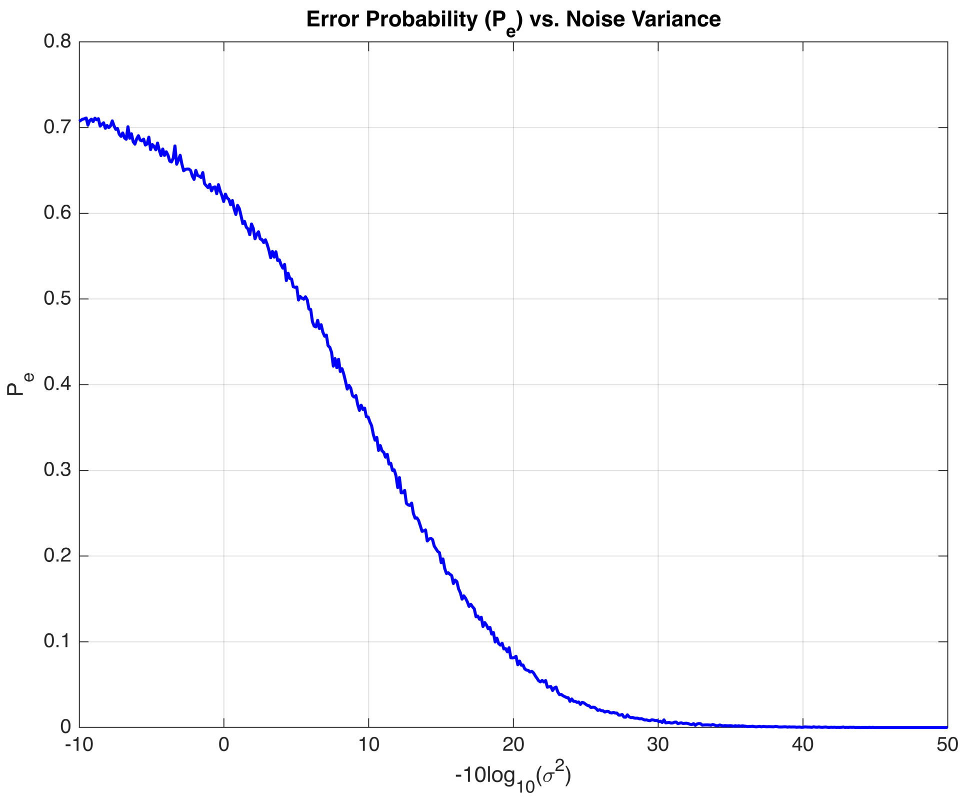
```
dur = 30;
h1 = randn(1, dur) + randn(1, dur) * 1j;
h2 = randn(1, dur) + randn(1, dur) * 1j;
```

Compute P_e. For each variance, we do 10 runs and compute the average probability.

```
Pe = zeros(size(variances));
for k = 1:length(Pe)
    p = 0;
    for m = 1:10
        p = p + ErrorProbabilityA(variances(k), h1, h2);
    end
    Pe(k) = p/10;
end
```

Plot the error probability vs noise variance.

```
figure;
x_axis = -10*log10(variances);
plot(x_axis, Pe, 'b-', 'LineWidth', 1.5);
grid on;
xlabel('-10log_{10}(\sigma^{2})');
ylabel('P_e');
title('Error Probability (P_{e}) vs. Noise Variance');
```



```
clear;
```

We can see that as the variance decreases, meaning that $-10\log_{10}(\sigma^2)$ increases, P_e tends to 0. Thus, as the variance tends to 0, we have a perfect signal extraction.

Scenario B: Both Users Use All Frequencies

Generating Transmitted Signals

Generate the random bits for each user.

```
clear;
bits1 = randi([0 1], 1, 2048);
bits2 = randi([0 1], 1, 2048);
```

Generate 1024 DFT coefficients using QPSK modulation.

```
% DFT coefficients for x1[n]
pairs1 = reshape(bits1, 2, []);
X1 = zeros(1, size(pairs1, 2));
for k = 1:size(pairs1, 2)
    if isequal(pairs1(:, k)', [0 0])
        X1(k) = -1 - 1j;
    elseif isequal(pairs1(:, k)', [0 1])
        X1(k) = -1 + 1j;
    elseif isequal(pairs1(:, k)', [1 1])
```

```
        X1(k) = 1 + 1j;
    else
        X1(k) = 1 - 1j;
    end
end
clear pairs1;

% DFT coefficients for x2[n]
pairs2 = reshape(bits2, 2, []);
X2 = zeros(1, size(pairs2, 2));
for k = 1:size(pairs2, 2)
    if isequal(pairs2(:, k)', [0 0])
        X2(k) = -1 - 1j;
    elseif isequal(pairs2(:, k)', [0 1])
        X2(k) = -1 + 1j;
    elseif isequal(pairs2(:, k)', [1 1])
        X2(k) = 1 + 1j;
    else
        X2(k) = 1 - 1j;
    end
end
clear pairs2;
```

Obtain the transmitted signals $x_1[n]$ and $x_2[n]$.

```
x1 = ifft(X1);
x2 = ifft(X2);
```

Generating Received Signal

We first generate the channels $h_1[n]$ and $h_2[n]$, both of which are of duration 30.

```
dur = 30;
h1 = randn(1, dur) + randn(1, dur) * 1j;
h2 = randn(1, dur) + randn(1, dur) * 1j;
```

We then compute the 1024-point circular convolution to get the uncontaminated signal.

```
clean = ifft(X1.*fft(h1, 1024)+X2.*fft(h2, 1024), 1024);
```

Then, we add noise to the original signal.

```
v = randn(1, size(clean, 2))*sqrt(0.1);
```

Get the received signal

```
y = clean + v;
clear v;
clear clean;
```

Reconstructing the Original Signals

$X_1[k]$ and $X_2[k]$ each can only take on 4 distinct values. We know the channel FFTs. Let $(s_1, s_2) \in S^2$. For each $Y[k]$, we define the equation of two unknowns

$$W[k] = s_1 H_1[k] + s_2 H_2[k].$$

Then, we find the pair (s_1, s_2) that minimizes $|Y[k] - W[k]|$. This is our guess for $(X_1[k], X_2[k])$.

We first define the set of symbols.

```
symb = [-1-1j, -1+1j, 1+1j, 1-1j];
[A, B] = meshgrid(symb, symb);
s = [A(:), B(:)];
clear A;
clear B;
clear symb;
```

Then, we iterate through the received signal and minimize $|Y[k] - W[k]|$.

```
% Define variables to hold reconstructed signals
X1_reconstructed = zeros(1, 1024);
X2_reconstructed = zeros(1, 1024);

% Compute the FFT of y[n], h_1[n], and h_2[n]
Y = fft(y);
H1 = fft(h1, 1024);
H2 = fft(h2, 1024);

% Iterate through received signal
for k=1:length(Y)
    min = Inf;
    for m = 1:size(s, 1)
        s_1 = s(m, 1);
        s_2 = s(m, 2);
        w = s_1*H1(k)+s_2*H2(k);
        diff = abs(Y(k)-w);
        if diff < min
            X1_reconstructed(k) = s_1;
            X2_reconstructed(k) = s_2;
            min = diff;
        end
    end
end
```

Identify Errors

Compare the extracted signals and original signals, and get the approximate probability of error.

```
num_diff_1 = sum(X1~=X1_reconstructed);
num_diff_2 = sum(X2~=X2_reconstructed);
```

Compute probability. Both FFTs have 1024 points, so then total number of points is $2 \cdot 1024 = 2048$.

```
Pe = (num_diff_1+num_diff_2) / (2*1024);
disp("Error Probability: " + Pe);
```

```
Error Probability: 0.42725
```

```
clear;
```

Noise Variance and Error Probability

We copy the above code into a function onto another file that takes in noise variance σ^2 and returns error probability P_e.

```
cd /Users/kyle/Library/'Mobile Documents'/com~apple~CloudDocs/'Course
Files'/'Fall 2024'/'ECE 161A'/MATLAB/'OFDM Project'/;
```

Define logarithmic range.

```
min_exp = -5;
max_exp = 1;
num_points = 500;
```

Get variances.

```
variances = logspace(min_exp, max_exp, num_points);
```

Generate channels.

```
dur = 30;
h1 = randn(1, dur) + randn(1, dur) * 1j;
h2 = randn(1, dur) + randn(1, dur) * 1j;
```

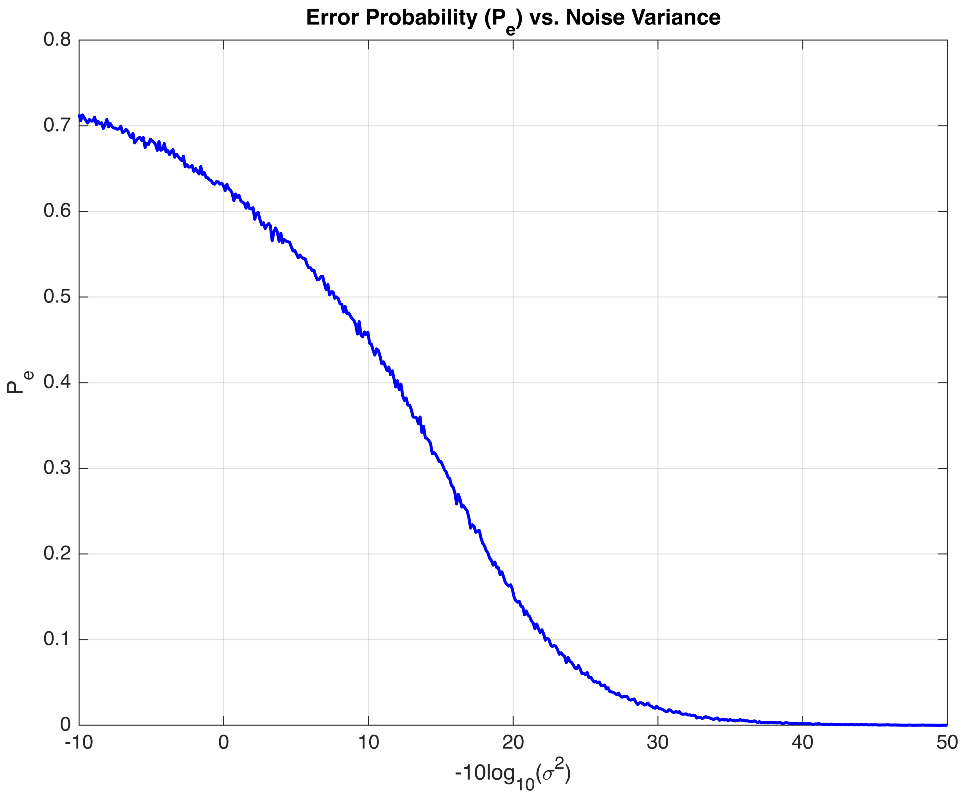
Compute P_e. For each variance, we do 10 runs and average the probability.

```
Pe = zeros(size(variances));
for k = 1:length(Pe)
    p = 0;
    for m = 1:10
        p = p + ErrorProbabilityB(variances(k), h1, h2);
    end
    Pe(k) = p/10;
end
```

Plot the error probability vs noise variance.

```
figure;
x_axis = -10*log10(variances);
```

```
plot(x_axis, Pe, 'b-', 'LineWidth', 1.5);
grid on;
xlabel('-10log_{10}(\sigma^{2})');
ylabel('P_e');
title('Error Probability (P_{e}) vs. Noise Variance');
```



```
clear;
```

As with part A, can see that as the variance decreases, meaning that $-10\log_{10}(\sigma^2)$ increases, P_e tends to 0. Thus, as the variance tends to 0, we have a perfect signal extraction.

Conclusion

We empircally tested two encoding schemes in this. In the first one (Scheme A), given two users, we assign one with to encode their bits with even frequencies and the other to encode with odd frequencies. In Scheme B, both users used all frequencies available. After both users encoded their bits, the transmitted signal sent to the receiver is corrupted with noise. We see that for both schemes, as the noise variances approaches zero, the error probabiity approaches zero. We tested variances in the range $[10^{-5}, 10]$. While both schemes started out with a probability error of around 0.7 for the highest variance closest to 10, we can see in the graph that they began decreasing as the variance decreased. For example, for the testing variance of $\sigma^2 = 0.1$, the error probability of scheme A is around 0.3, which is much lower than scheme B's 0.4.

We note that scheme A has a quicker decrease than scheme B, with a steeper transition slope to 0. This is due to the lower complexity with encoding and decoding in scheme A. When decoding, scheme A only needs to consider 4 symbols, while scheme B needs to choose from 16 symbols. The higher complexity of scheme B results in more overlap between the two users, so it is harder to decode more accurately. Therefore, scheme A is likely to perform better in terms of accuracy.

However, scheme B allows us to transmit more information, while scheme A requires that we halve it. Therefore, while scheme A may result in more accuracy, it transmits much less information than B.