ECE131A: Project (MATLAB): Simulation and Analysis of Error in a Binary Transmission Channel

Introduction

The goal of this project is to investigate the average probabilities of error in a simple binary data communication system. In this case we will consider a single bit transmission channel, in which binary data bits are sent in succession to form a binary data vector. We will consider a signal of S volts to be transmission of an information bit "1" (denote this as hypothesis H_1) and a signal of -S volts to be the transmission of a of an information bit "0" (denote this as hypothesis H_0). This communication system is an additive channel modeled by the equation:

$$X = \begin{cases} S + N, H_1 = \text{"1"} \\ -S + N, H_0 = \text{"0"} \end{cases}$$

Where S is the sent signal sequence in the form of a data vector, N is the noise vector added in transmission, and X is the received data vector. Here we will model the noise in signal transmission to be an generated vector with a specified symmetric zero mean probability distribution $f_N(x)$ that is added to the sent binary data vector. This results the received data vector to take on the probability distribution of the noise that is shifted by the magnitude of the sent signal: $f_N(x + S)$ or $f_N(x - S)$

Assuming the probabilities of either a "0" or "1" being sent are approximately equal, (i.e. $P(H_0) = P(H_1) = \frac{1}{2}$) the receiver will operate under a decision rule setting the threshold to 0 for each received data signal x. As such:

if $x \geq 0$, decide H_1 was sent;

if $x < 0$, decide H_0 was sent.

Due to the addition of the noise vector, it is possible the noise vector will distort the particular sent signal enough such that the decision rule will fail and the receiver will erroneously decide that H_1 was sent when the original signal was sending H_0, or vice versa. In this project, we will explore the average rate of error $P(e)$ in this transmission channel given two different probability distributions of noise, as well as different signal to noise ratios.

1 Methods of Computation

This project will utilize two methods of numerically evaluating the probability of error:

1. Analytically evaluating the probability of error by computing the cumulative distribution function of the shifted probability distribution of the received noise.

2. Creating a Monte Carlo simulation of the binary transmission channel using random number generation, and recording the proportion of instances where the decision rule resulted in an error.

2 Analytical Evaluation of $P(e)$

We will begin by defining the average probability of error as:

$$\begin{aligned} P(e) &= P(e|H_1)P(H_1) + P(e|H_0)P(H_0) \\ &= P(e|H_1)\frac{1}{2} + P(e|H_0)\frac{1}{2} \end{aligned} \tag{1}$$

Because we have assumed the noise pdf $f_N(n)$ is a symmetric function, $P(e|H_1)$ is equal to $P(e|H_0)$ and the average probability of error can be expressed as:

$$P(e) = P(e|H_1) = P(e|H_0) \tag{2}$$

We will also define the Signal to Noise Ratio (SNR(dB)) as follows:

$$SNR(dB) = 10\log_{10}(\frac{S^2}{\sigma^2}) = 20\log_{10}(\frac{S}{\sigma}) \tag{3}$$

This equation can be rewritten in terms of SNR(dB) as:

$$\frac{S}{\sigma} = 10^{\frac{SNR(dB)}{20}} \tag{4}$$

2.1 Cumulative Distribution of Received Signals

In order to analytically evaluate $P(e|H_1)$ or $P(e|H_0)$ we will need to integrate the probability distribution functions of the received signals. This will give us the cumulative distribution function of the received noise. Provided these are evaluated using the appropriate bounds, the resulting value will be the desired probability of error.

The probability of error when H_1 was sent:

$$P(e|H_1) = \int_{-\infty}^{0} f_N(x - S)dx \tag{5}$$

The probability of error when H_0 was sent:

$$P(e|H_0) = \int_{0}^{\infty} f_N(x + S)dx \tag{6}$$

The bounds are such that the integral will compute the probability that the received signal will fall outside the threshold of the decision rule (i.e. a negative voltage when H_1 was sent or a positive voltage when H_0 was sent. Note that by Equation (2) we will only need to evaluate one of the above cases to obtain $P(e)$.

2.2 Gaussian Noise Distribution

The first case we will consider is that where the probability distribution of the noise vector follows a normal, or Gaussian curve. The Gaussian probability distribution function (PDF) is given by the function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{7}$$

In order to model $P(e)$ in the case of a Gaussian noise distribution, we will need to compute the integral of the shifted Gaussian PDF. In this case the noise is zero mean, so we will disregard μ. Using Equation (5), we obtain the equation:

$$P(e|H_1) = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-S)^2}{2\sigma^2}} dx \tag{8}$$

To obtain the Bit Error Rate (BER) vs. SNR(dB) curve, we will use Equation (4) to obtain the above Equation (8) in terms of SNR(dB). In this case $\sigma = 1$, so we will simplify this as well.

$$P(e|H_1) = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{\frac{(x-10^{\frac{SNR(dB)}{20}})^2}{2}} dx \tag{9}$$

Evaluating this equation for different values of SNR and plotting the results will give the analytical BER vs. SNR curve for the case of a Gaussian noise distribution.

2.3 Laplacian Noise Distribution

The next case we will consider is that where the probability distribution of the noise vector follows a Laplacian function. The Laplacian PDF is given by the function:

$$f(x) = (\frac{\alpha}{2})e^{-\alpha|x|} \tag{10}$$

Where $\sigma^2 = (\frac{2}{\alpha^2})$. Because we will consider the standard deviation σ to be equal to 1, then $\alpha = \sqrt{2}$. In order to model $P(e)$ in the case of a Laplacian noise distribution, we will need to compute the integral of the shifted Laplacian PDF. Here we will evaluate the case of $P(e|H_1)$ for all negative values of x. Therefore we will substitute the absolute value $|x|$ for $-x$. This gives us:

$$f(x) = (\frac{\alpha}{2})e^{\alpha x}$$

Next, using Equation (5), we obtain:

$$P(e|H_1) = \int_{-\infty}^{0} \frac{\sqrt{2}}{2} e^{\sqrt{2}(x-S)} dx \tag{11}$$

To obtain the BER vs. SNR(dB) curve, we will use Equation (4) to obtain the above equation (11) in terms of SNR(dB). In this case $\sigma = 1$, so we will simplify this as well.

$$P(e|H_1) = \int_{-\infty}^{0} \frac{\sqrt{2}}{2} e^{\sqrt{2}(x-10^{\frac{SNR(dB)}{20}})} dx \tag{12}$$

Evaluating this equation for different values of SNR and plotting the results will give the analytical BER vs. SNR curve for the case of a Gaussian noise distribution.

3 $P(e)$ by Monte Carlo Simulation

Next we will discuss the process of computing the average rate of error for different signal to noise ratios using a Monte Carlo simulation. This method involves utilizing MATLAB to generate sequences of random numbers as well as sequences of numbers in a specified probability distribution. The random vectors representing the binary signal transmissions and the noise are generated according to the desired probability distribution, summed and run through the decision rule of the receiver to generate a received data. Next, the received data is compared to the sent data to determine the proportion of error for the particular error. The above process can be defined as a function in MATLAB, and it represents a single trial, or data point in the SNR vs. BER curve. This function can be utilized to generate more data points through numerous trials with different signal to noise ratios. Ultimately when enough data points are gathered, we are able to plot a BER vs. SNR(dB) curve similar to that which we obtained analytically.

3.1 Case A: Gaussian Noise Distribution

Once again, we will first consider the case in which the noise follows a zero mean Gaussian probability distribution with unit variance. The process of executing a trial in for this case is as follows:

1. We begin by generating a data vector following a Gaussian distribution.

2. Next, generate a random, uniformly distributed vector of numbers between 0 and 1.

3. Convert the uniformly distributed vector to a binary vector. This will convert values less than 0.5 in the vector to 0, and values greater than or equal to 0.5 to 1.

4. Convert the binary vector to a vector containing ±S values. This is the strength of the signal and will be proportional to the signal to noise ratio.

5. Sum the two vectors (binary $\pm S$ and Gaussian) elementwise to simulate the process of transmission. This yields your received vector.

6. We now evaluate $P(e|H_1)$, therefore we multiply the received data vector into the originally generated binary data vector to return only the bits where +S was sent.

7. Now we sum the elements of the vector that were negative (i.e. would have resulted in an error with the given decision rule) and divide that value with the sum of all non-zero elements in the vector to give the simulated proportion of error.

The following constitutes a single trial for a specific value of S. This process is repeated numerous times with various signal strengths (values of S) to yield the simulated BER vs. SNR(dB) curve for the case of Gaussian noise.

3.2 Case B: Laplacian Noise Distribution

Next, we consider the simulation of the same process with a zero mean Laplacian noise distribution with unit variance. The process is slightly more complex in this case due to the fact that there is no predefined function in MATLAB to generate a vector following a Laplacian distribution. Therefore we must utilize a method known as inverse transform sampling, in which we transform a uniformly distributed vector from 0 to 1 by inputting its elements into the inverse CDF of the desired distribution. Consequently, we now find it necessary to derive the inverse CDF of the Laplacian distribution:

To obtain the inverse CDF of the Laplacian distribution, we will begin with the PDF, which we are given in Equation (10). Following the definition of CDF, we apply an integration across $(-\infty, \infty)$ to the PDF to obtain the following:

$$f(x) = \int_{-\infty}^{\infty} \frac{\sqrt{2}}{2} e^{-\sqrt{2}|x|} dx$$

Next, we apply the definition of an absolute value to split the integral and evaluate:

Case 1: $x > 0$

$$\begin{aligned} f(x) &= \int_{-\infty}^{0} \frac{\sqrt{2}}{2} e^{\sqrt{2}n} dn + \int_{0}^{x} \frac{\sqrt{2}}{2} e^{-\sqrt{2}n} dn \\ &= \frac{\sqrt{2}}{2} \int_{0}^{\infty} e^{-\sqrt{2}n} dn + \frac{\sqrt{2}}{2} \int_{0}^{x} e^{-\sqrt{2}n} dn \\ &= -\frac{1}{2} \left[e^{\sqrt{2}n} \right]_{0}^{\infty} - \frac{1}{2} \left[e^{\sqrt{2}n} \right]_{0}^{x} \\ &= -\frac{1}{2} e^{-\sqrt{2}x} + 1 \end{aligned} \tag{13}$$

Case 2: $x \leq 0$

$$\begin{aligned} f(x) &= \int_{-\infty}^{x} \frac{\sqrt{2}}{2} e^{\sqrt{2}x} dx \\ &= \frac{\sqrt{2}}{2} \int_{-x}^{\infty} e^{-\sqrt{2}x} dx \\ &= -\frac{1}{2} \left[e^{-}\sqrt{2}n \right]_{-x}^{\infty} \\ &= \frac{1}{2} e^{\sqrt{2}x} \end{aligned} \tag{14}$$

We have now established that the CDF of the Laplace distribution takes the form:

$$F(x) = \begin{cases} -\frac{1}{2} e^{-\sqrt{2}x} + 1, x > 0 \\ \frac{1}{2} e^{\sqrt{2}x}, x \leq 0 \end{cases}$$

Now, following the inverse method $F(F^{-1}(x)) = x$, we will take the inverse of both cases to obtain the inverse CDF.

Case 1: $x > 0$

$$\begin{aligned} -\frac{1}{2} e^{-\sqrt{2}F^{-1}(x)} + 1 &= x \\ e^{-\sqrt{2}F^{-1}(x)} &= 2 - 2x \\ -\sqrt{2}F^{-1}(x) &= ln(2 - 2x) \\ F^{-1}(x) &= -\frac{1}{\sqrt{2}} ln(2 - 2x) \end{aligned} \tag{15}$$

Case 2: $x \leq 0$

$$\begin{aligned} \frac{1}{2} e^{\sqrt{2}F^{-1}(x)} &= x \\ e^{\sqrt{2}F^{-1}(x)} &= 2x \\ \sqrt{2}F^{-1}(x) &= ln(2x) \\ F^{-1}(x) &= \frac{1}{\sqrt{2}} ln(2x) \end{aligned} \tag{16}$$

We have now found that the inverse CDF of the Laplace distribution takes the form:

$$F^{-1}(x) = \begin{cases} -\frac{1}{\sqrt{2}} ln(2 - 2x), x > 0 \\ \frac{1}{\sqrt{2}} ln(2x), x \leq 0 \end{cases} \tag{17}$$

This piecewise function can also be rewritten in terms of the sign function as follows:

$$F^{-1}(x) = -\frac{1}{\sqrt{2}} sign(x - \frac{1}{2}) ln(1 - 2|x - \frac{1}{2}|) \tag{18}$$

Now that we have obtained the inverse Laplacian CDF, we are now able to proceed with the simulation. The process of executing a trial in for this case is as follows:

1. Begin by generating a random vector following a uniform distribution across the range (0,1).

2. In order to obtain a random vector following the zero mean, unit variance Laplacian distribution, we will plug the newly generated uniform vector into the inverse Laplacian CDF. This is a method known as inverse transform sampling.

3. Generate another random vector following a uniform distribution across the range (0,1).

4. Convert the uniformly distributed vector to a binary vector. This will convert values less than 0.5 in the vector to 0, and values greater than or equal to 0.5 to 1.

5. Convert the binary vector to a vector containing ±S values. This is the strength of the signal and will be proportional to the signal to noise ratio.

6. Sum the two vectors (binary $\pm S$ and Laplacian) elementwise to simulate the process of transmission. This yields your received vector.

7. We now evaluate $P(e|H_1)$, therefore we multiply the received data vector into the originally generated binary data vector to return only the bits where +S was sent.

8. Now we sum the elements of the vector that were negative (i.e. would have resulted in an error with the given decision rule) and divide that value with the sum of all non-zero elements in the vector to give the simulated proportion of error.

The following constitutes a single trial for a specific value of S. This process is repeated numerous times with various signal strengths (values of S) to yield the simulated BER vs. SNR(dB) curve for the case of Laplacian noise.

4 Results and Plots

Both the analytical and simulated BER vs. SNR(dB) were plotted in MATLAB through use of the `semilogy()` function. This resulted in a plot resembling a waterfall, with a high plateau and rapid drop towards the right end of the graph. In both cases, it appears the proportion of error becomes negligible past a signal to noise ratio of over 5-10 dB. We also notice the proportion of error is too high to be feasible until the SNR is above 1 dB.

Comparing the analytical and simulated plots, we can see that they are very similar, and are therefore both valid methods of computing the proportion of error. The simulated BER vs. SNR(dB) plot is rougher and deviates slightly from the analytical plot, which is to be expected due to the nature of random number generation.

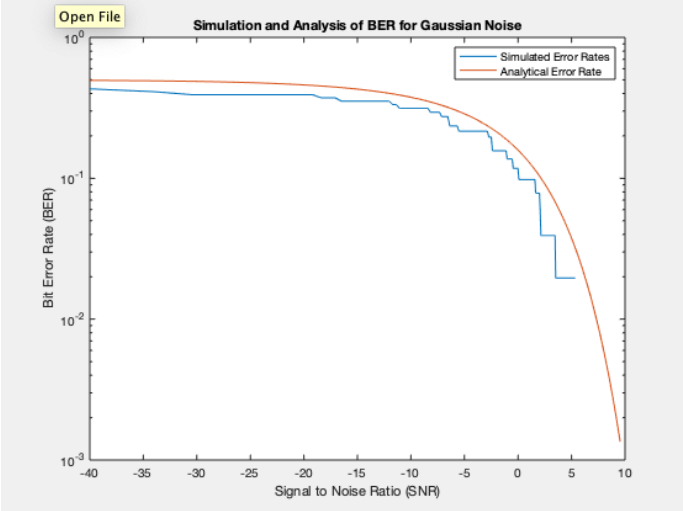

Figure 1: BER vs. SNR(dB) curve for case of Gaussian noise

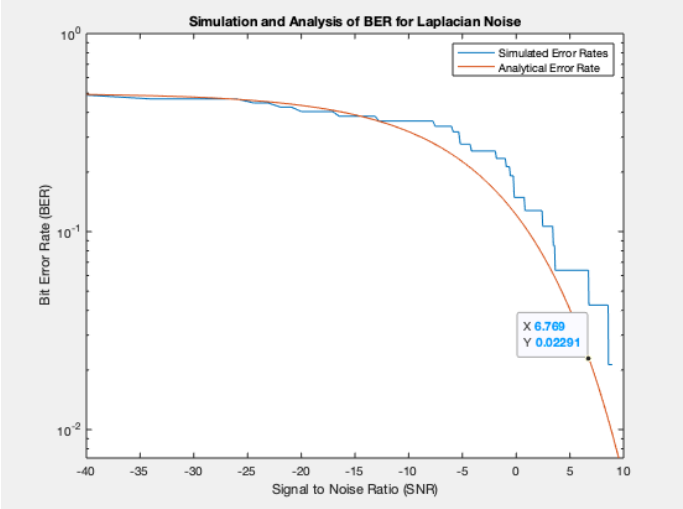

Figure 2: BER vs. SNR(dB) curve for case of Laplacian noise

5 Listing of MATLAB Code (Appendix)

5.1 Case A Simulation

```
function Pe = Case_A(S)
    M = 100;
    randn('seed',2000);
    %Generate Gaussian noise vector with 100 elements
    N = randn(1,M);
    %Generate uniform random vector with 100 elements
    rand('seed',2001);
    S1 = rand(1,M);
    %Convert uniform vector to binary vector
    S2 = S1 >= 0.5;
    %Convert binary vector to signal vector
```

```
    SS = S*(2*S2-1);
    %Sum to simulate transmission
    X = SS + N;
    %Only record when H1 was sent
    XX = X .* S2;
    X1 = XX <0;
    M1 = sum(S2);
    X11 = sum(X1);
    %Compute proportion of error
    Pe = X11/M1;
end
```

5.2 Case B Simulation

```
function Pe = Case_B(S)
    M = 100;
    rand('seed',2002);
    %Generate uniform random vector with 100 elements
    U = rand(1,M);
    %{
    To generate Laplacian noise vector we use inverse cdf function
    F(U) = -(1/(2^0.5))sgn(U-0.5)ln(1-2|U-0.5|)
    where U is the uniform random vector
    %}
    N = -(1/(2^0.5)) * sign(U-0.5) .* log(1-2.*abs(U-0.5));
    rand('seed',2003);
    %Generate uniform random vector with 100 elements
    S1 = rand(1,M);
    %Convert uniform vector to binary vector
    S2 = S1 >= 0.5;
    %Convert binary vector to signal vector
    SS = S*(2*S2-1);
    %Sum to simulate transmission
    X = SS + N;
    %Only record when H1 was sent
    XX = X .* S2;
    X1 = XX < 0;
    M1 = sum(S2);
    X11 = sum(X1);
    %Compute proportion of error
    Pe = X11/M1;
end
```

5.3 Case A Plotting

```
signalMagnitudes = 0:0.01:3;
BER = zeros(1,301);
%Run Case A for 301 different signal magnitudes
for n = 1:301
    BER(n) = Case_A(signalMagnitudes(n));
end
figure();
```

```
%Convert S to SNR(dB)
SNR = 20*log10(signalMagnitudes);
%Plot simulated BER vs. SNR(dB)curve
semilogy(SNR, BER);
hold
syms x
f(x) = (1/((2*pi)^0.5))*exp(-((x-10.^(SNR/20)).^2)/2);
%{
We will approximate the improper integral to a sufficiently large
negative number
%}
F = int(f, x, -100, 0);
%Plot analytical BER vs. SNR(dB) curve
semilogy(SNR, F);
legend('Simulated Error Rates','Analytical Error Rate');
xlabel('Signal to Noise Ratio (SNR)');
ylabel('Bit Error Rate (BER)');
title('Simulation and Analysis of BER for Gaussian Noise');
```

5.4 Case B Plotting

```
signalMagnitudes = 0:0.01:3;
BER = zeros(1,301);
%Run Case B for 301 different signal magnitudes
for n = 1:301
    BER(n) = Case_B(signalMagnitudes(n));
end
figure();
%Convert S to SNR(dB)
SNR = 20*log10(signalMagnitudes);
%Plot simulated BER vs. SNR(dB)curve
semilogy(SNR, BER);
hold
syms x
%Define Laplacian pdf
f(x) = ((2^0.5)/2)*exp((2^0.5)*(x-10.^(SNR/20)));
%{
We will approximate the improper integral to a sufficiently large
negative number
%}
F = int(f, x, -100, 0);
%Plot analytical BER vs. SNR(dB) curve
semilogy(SNR, F);
legend('Simulated Error Rates','Analytical Error Rate');
xlabel('Signal to Noise Ratio (SNR)');
ylabel('Bit Error Rate (BER)');
title('Simulation and Analysis of BER for Laplacian Noise');
```

Remark: Because I was unable to find a method in MATLAB to compute an improper integral in the calculation of the analytical curve, I approximated $-\infty$ to -100. The idea was that this would provide a very close approximation to the actual plot.