On the effect of Gaussian and Laplacian Noise on a Binary Communication Channel through Monte Carlo Simulation

Alexandre William Bryan Leon[1]

[1]University of California, Los Angeles

Abstract

The paper studies the transmission of a signal in a binary data communication system, subject to Gaussian or Laplacian distributed noise. We run Monte Carlo simulations on MATLAB with the different noise classes. We focus our analysis on the influence of the Signal-to-Noise ratio of the channel, through examination of the Bit Error Curve (BER) for both distributions. We demonstrate how the two classes of noise compare in regards to their average errors, which can be translated to error-resolving strategies or signal amplitude design.

Introduction

The typical communication channel only sends two messages, in the form of a logic 0 and 1, encoded by symmetric signals with the same magnitude, but opposite amplitudes that we denote S. The symmetric signals are carried through channels that are not necessarily ideal: most times the signals will be contaminated by noise. Two types of noise that might occur in real applications are Gaussian and Laplacian noise. The non-ideal channel we will be modeling is an additive channel, which means the noise is added to the signal.

Problem Description

Our channel is described such as for the transmission of a '1'(denoted as hypothesis H_1) we send the amplitude S in volts, while for the transmission of a '0' (denoted as hypothesis H_0) we send the amplitude $-S$.

We receive the signal X with Noise N as follows

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

X is a random variable modeling the received message, and N is the noise random variable assumed to have a symmetric pdf noted

$$f_n(n), -\infty < n < \infty$$

We assume the probability of received each message is the same, therefore in deciding which message was received we can set a threshold $t_0 = 0$, around which we will decide which signal was sent. If above the threshold, we assume H_1 was sent, while if under the threshold we assume H_0 was sent.

Hence the probability of error when H_1 was sent is

$$\mathrm{P(e}\,|H_1) = \int_{-\infty}^{0-} f_1(x)dx = \int_{0}^{\infty} f_n(x - S)dx$$

And the probability of error when H_0 was sent

$$\mathrm{P(e}\,|H_0\,) = \int_{-\infty}^{0-} f_0(x)dx = \int_0^{\infty} f_n(\,x+S)dx$$

We note the average probability of error

$$P(e) = P(e\,|H_1) * P(H_1) + P(e \mid H_0) * P(\,H_0)$$

Which simplifies to

$$P(e) = P(e\,|H_1) * \left(\frac{1}{2}\right) + P(e \mid H_0) * (\frac{1}{2})$$

Given the pdf of the noise random variable is symmetric, we can write $P(e\,|H_1) = P(e \mid H_0)$ and therefore our average probability of error is

$$P(e) = P(e\,|H_1) = \;\; P(e \mid H_0)$$

To evaluate our average error we can concentrate on $P(e\,|H_1)$ since they are equal.

We define the Signal-to-Noise ratio (SNR) as $SNR(dB) = 20log_{10}(\frac{S}{\sigma})$

Where S is the amplitude of the signal sent, and σ is the standard deviation of the Noise random variable.

Hence

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

In our setting the standard deviation for the random variables will be fixed, therefore varying SNR has a direct influence on the amplitude of the Signal sent. The larger the signal compared to the expected noise, the lower the error should be.

Classes of Noise

For this simulation, we focus on two classes of noise, denoted by Case A and Case B, as we will run separate simulations for each.

Case A

We select our Noise random variable N to a be a zero-mean Gaussian random variable with standard deviation $\sigma = 1$.

The Gaussian random variable is very commonly used to model white noise, and is built in MATLAB.

Case B

We select our noise to follow a Laplacian pdf with standard deviation $\sigma = 1$.

Therefore the pdf of N will be

$$f_n(n) = \left(\frac{\alpha}{2}\right) e^{-\alpha|n|}\,, -\infty < n < \infty$$

Where $\sigma^2 = (2\,/\,\alpha^2)$.

Therefore our $\alpha = \sqrt{2}$

Experimental Procedure

We run simulations using the program MATLAB. We follow the same overall procedure for the two Noise variables, however some differences in procedure will arise at certain steps.

We analyze a stream of messages of length M, which is subject to the random variable N. Because we will study the noise over large ranges and obtain low average probabilities, we select a parameter M that can model a realistic application of a communication channel. We select M to be one million, which is relatively small and

would model a small time of a high frequency communication channel. We select this parameter to lower the computation time and still obtain a result on a logarithmic scale. We repeat the following procedure over different range for SNR.

We first look at the symmetric case of SNR from [-20:20], to see extreme cases. In realistic applications, the range of SNR will range from 5dB for a very weak signal to 40dB for a good signal[1] . We therefore observe SNR over a range of [-5:40]. Because MATLAB allows us to, we compute SNR with a step of 0.05.

We proceed with our Monte Carlo simulation as follows:

Step 1: Generating the Noise vector.

We first generate a Noise vector N of length M.

Case A:

We use the built-in function of MATLAB (randn) to generate a Gaussian noise vector N of length M.

Case B:

To obtain the noise vector N, we use the transformation method. We first generate a uniformly distributed vector over the range [0,1] with size M.

Then we use the inverse CDF of the Laplacian random variable to obtain a Laplacian noise. The formula for the Inverse Laplace CDF is derived in **Appendix 1.**

$$F^{-1}(Y) = \begin{cases} \frac{1}{\alpha}\ln(2Y) \ , 0 < Y < 0.5 \\ -\frac{1}{\alpha}\ln(2-2Y) \, , 0.5 \leq Y \leq 1 \end{cases}$$

We obtain a new vector NL (equivalent to N in the rest of the procedure) which represents a Laplacian noise.

Step 2: Generating the random binary data vector.

In both cases we first create a uniformly distributed random vector of length M. Because we determined each message had equal probability, we generate a new vector where we isolated the values higher than - $P(H_1) = (\frac{1}{2})$

We obtain a vector that only contains 0 and 1, which correspond to the information we want to send. We call this vector S2. This vector also tracks the indices at which we want to transmit signal "1", therefore we will re-use it later. We then encode the signal such that

$$1 = \ +S \, , 0 = \ -S$$

Using the formula

$$SS = S * (2 * S2 - 1)$$

The value of S is dependent on the value of SNR which will vary depending on the iteration. We now possess a new vector which models the signal we want to transmit, which we call SS.

Step 3: Generate the received vector

We obtain the received vector through the additive channel, which is nothing but

$$X = SS + N$$

Step 4: Collecting received data under hypothesis H_1.

Using our vector S2 which records the indices at which "1" was to be transmitted, we filter our received signal

$$XX = X.*S2$$

This vector XX records the data when +S was to be sent.

Step 5: Identify the errors

Using our threshold value of 0, we filter all the messages which were under 0 but meant to transmit +S. Those messages are errors.

$$X1 = XX < 0$$

X1 is a logical vector which tracks the indices at which there was an error.

Step 6: Average probability evaluation.

We obtain the sum of S2, which yields the number of +s message to be sent, and we sum X1, which yields the number of errors.

$$P(e) = \frac{sum(X1)}{sum(S2)}$$

We obtain our average probability of error by the ratio of the errors over the messages sent.

This procedure is iterated multiple times over the range

Experimental Data

Case A

We first want to verify our noise follows a Gaussian distribution. In Appendix 2, the histogram for N, our Noise can be found. The shape of the curve attests the noise to be Gaussian.

In **appendix 2** can be found the Histogram and BER graph of the simulation over the SNR range [-20:20].

In **appendix 3** can be found the Histogram and BER graph of the simulation over the SNR range [-5:40].

Those curves follow the expected "Waterfall" shape.

Case B

We first want to verify our noise follows a Laplacian distribution. In Appendix 4, the histogram for N, our Noise can be found. The shape of the curve attests the noise to be Laplacian.

In **appendix 4** can be found the Histogram and BER graph of the simulation over the SNR range [-20:20].

In **appendix 5** can be found the Histogram and BER graph of the simulation over the SNR range [-5:40].

Those curves follow the expected "Waterfall" shape, which showed a rapid decay.

Data Analysis and Inferences

Case A:

From the extreme case presented in **Appendix 2**, where we look at P(e) where SNR is -20, we see the value tends to 0.3. As SNR decreases, we see the probability of error goes higher, we infer the value will approach 0.5. This is because the noise amplitude is so high compared to the signal that the receiver will classify the noise rather than the signal.

The graph from **Appendix 3** shows the SNR ranging from [-5:40]. We see for both ranges that P(e) decreases exponentially with SNR. After an SNR of 5 (in which we're in the 10^{-2} error range, the error will drop very fast and reach a value close to $2*10^{-6}$ at around SNR = 13. This demonstrates that a channel subject to Gaussian noise will reach that threshold for an SNR higher than 13. From the source cited, this is within the range we expected.

Case B:

For the extreme case of **Appendix 4**, we look at P(e) when SNR = -20, and find an average error of 0.4. We infer that for a smaller SNR the value would tend to 0.5.

The graph from **Appendix 5** shows the P(e) on the range of [-5:40]. We notice that for the Laplacian noise, the value will reach the $2*10^{-6}$ threshold at an SNR of 17 and floor beyond that. This is due to a limitation of our simulation.

Comparatively:

We collected 1 million samples from the channel, so an error of $2*10^{-6}$ would correspond to approximately 2 errors in a million, which is the threshold we have investigated for both Noise classes. We demonstrated that this threshold is reached for different SNR values. For Gaussian it is 13, while for Laplacian it is 17. This means that at the same SNR level, a channel contaminated by Gaussian noise will have lower average error rate after a certain threshold. From this, we infer that if we know our channel will be subjected to Laplacian noise, we need to send a signal with higher amplitude (by a large factor considering the logarithmic nature of SNR) than if our channel was subject to Gaussian Noise. This trend is observed for high SNR, but is not necessarily applicable for all values, as Gaussian noise could be more prevalent than Laplacian noise at lower SNR values.

Further investigation with more samples (larger M) would allow us to investigate the threshold of 10^{-9} average error. We notice the scale will not indicate values beyond 20 SNR in the plot, which shows our simulation is limited by MATLAB and our choice of samples. For high SNR, the error will tend to zero.

Conclusion

The simulation allowed us to investigate two classes of noise at various SNR. We investigated the extreme cases of an unrealistic low SNR and found the probability of error tended to 0.5. Then we compared how the two average probabilities varied. The two had similar shape, which shows that the SNR will have the same influence regardless of the class of noise, and reinforce the real meaning of SNR which is how large the amplitude of the signal is compared to noise.

Finally we investigated how fast the two errors will reach the threshold of 2 errors in a million and we concluded the Gaussian noise would reach that threshold for a lower SNR, making the Laplacian noise harder to control in applications.

Works cited

1 - Geier,J , *How to: Define Minimum SNR Values for Signal Coverage,* www.wireless-nets.com

Pandya,A, *Week 6: Set #2.2 (Project)*

Appendix

Appendix 1 – Derivation of Inverse Laplace CDF

Appendix 2 – Experimental results for Gaussian, SNR in [-20:20]

Appendix 3 – Experimental results for Gaussian, SNR in [-5:40]

Appendix 4 – Experimental results for Laplacian, SNR in [-20:20]

Appendix 5 – Experimental results for Laplacian, SNR in [-20:20]

Appendix 6 – MatLab code for Gaussian simulation

Appendix 7 – MatLAb code for Laplacian simulation

Appendix I: Analysis of the Laplacian cdf.

The pdf of Laplacian rv is given by

$$f(n) = \left(\frac{\alpha}{2}\right) e^{-\alpha |n|}, \quad -\infty < n < \infty$$

$$\sigma^2 = \frac{2}{\alpha^2} \quad \text{and} \quad \alpha = \sqrt{2} \text{ for } \sigma = 1$$

Cumulative density function

$$F(n) = \int_{-\infty}^{n} f(t)\,dt$$

$$= \int_{0}^{n} \frac{\alpha}{2} e^{-\alpha t}\,dt + \int_{-\infty}^{n} \frac{\alpha}{2} e^{\alpha t}\,dt$$

$$= \underbrace{\left[-\frac{1}{2} e^{-\alpha t}\right]_0^n}_{0 \le n < \infty} + \underbrace{\left[\frac{1}{2} e^{\alpha t}\right]_{-\infty}^{n}}_{-\infty < n < 0}$$

$$F(n) = \begin{cases} \frac{1}{2} e^{\alpha n} & n < 0 \\ 1 - \frac{1}{2} e^{-\alpha n} & 0 \le n \end{cases}$$

Inverse cumulative density function

Two cases

$0 < Y < 0.5$

$$Y = \frac{1}{2} e^{\alpha n}$$

$$\ln(2Y) = \alpha n$$

$$\Rightarrow n = \frac{1}{\alpha} \ln(2Y), \quad n < 0$$

$0.5 \leq Y < 1$

$$Y = 1 - \frac{1}{2} e^{-\alpha n}$$

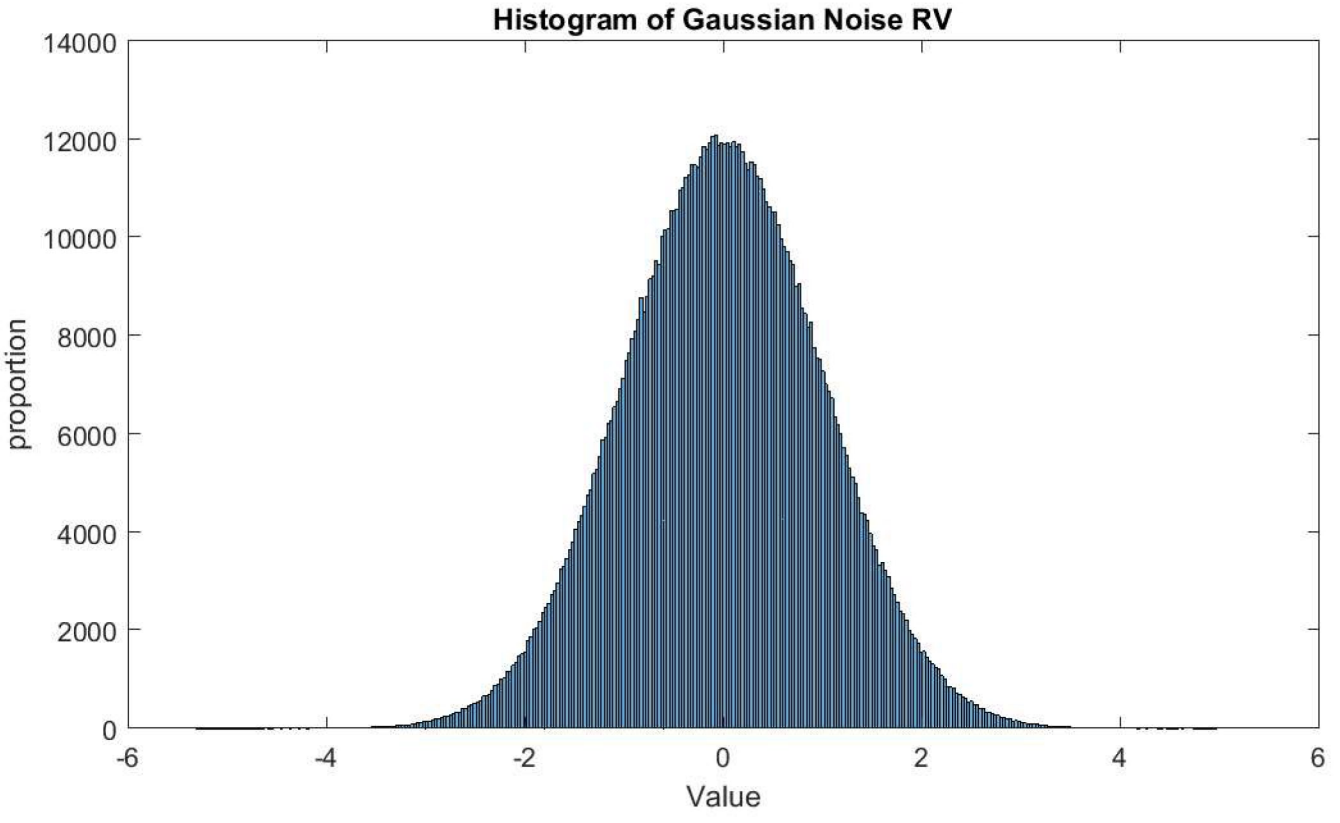
$$2 - 2Y = e^{-\alpha n}$$

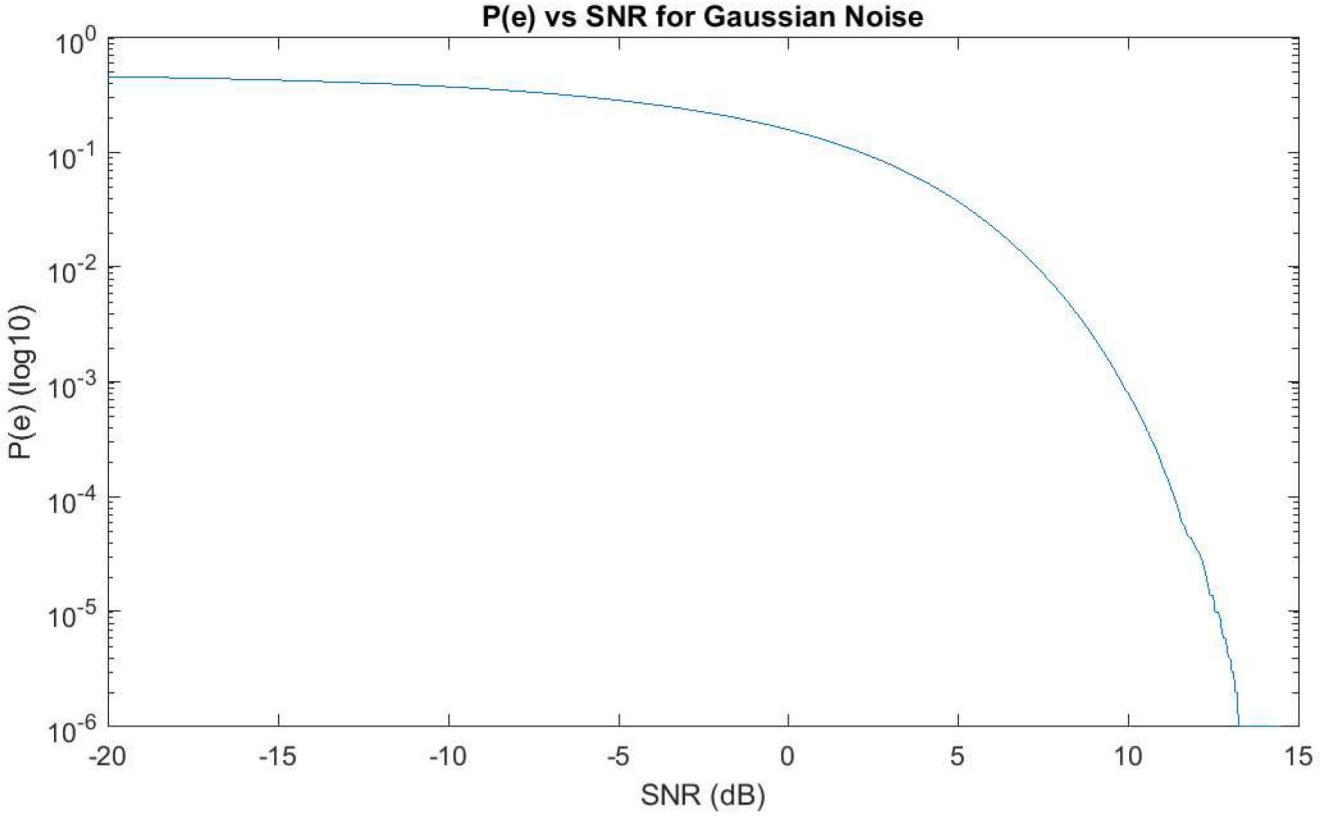
$$\ln(2 - 2Y) = -\alpha n$$

$$\Rightarrow n = -\frac{1}{\alpha} \ln(2 - 2Y)$$

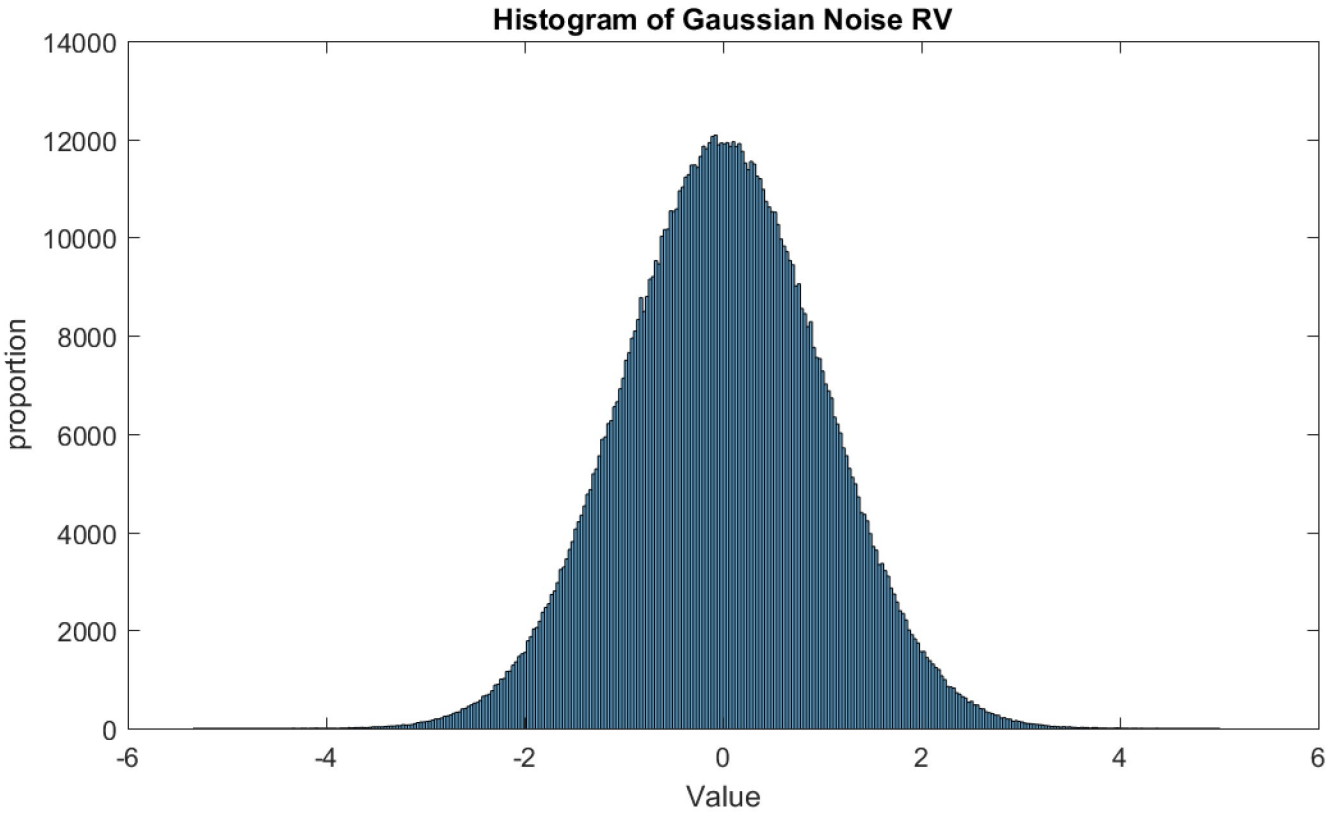
$$\therefore F^{-1}(Y) = \begin{cases} \frac{1}{\alpha} \ln(2Y) & 0 < Y < 0.5 \\ \frac{-1}{\alpha} \ln(2 - 2Y) & 0.5 \leq Y < 1 \end{cases}$$

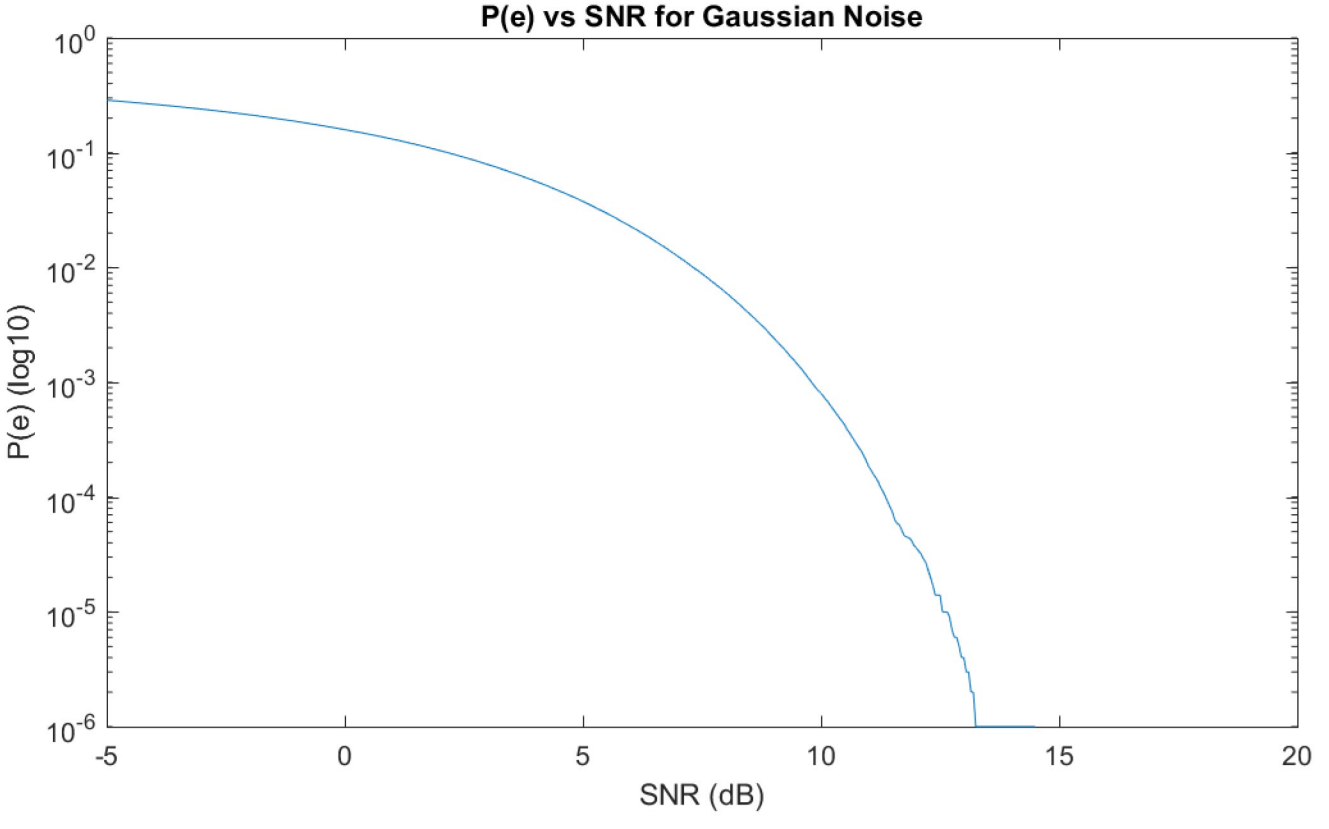
Appendix 2


Histogram of Gaussian Noise RV
proportion
Value
14000
12000
10000
8000
6000
4000
2000
0
-6
-4
-2
0
2
4
6

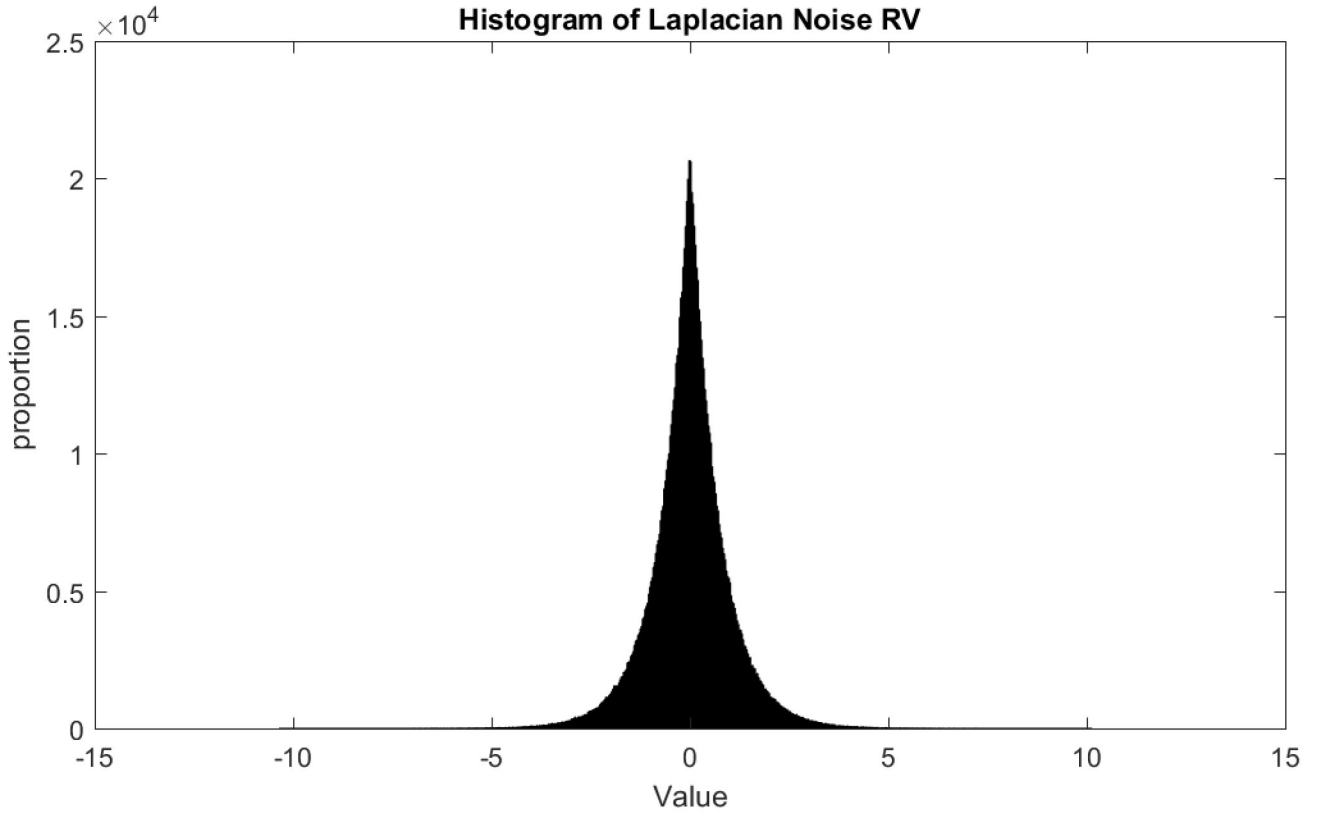

P(e) vs SNR for Gaussian Noise
P(e) (log10)
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
-20
-15
-10
-5
0
5
10
15

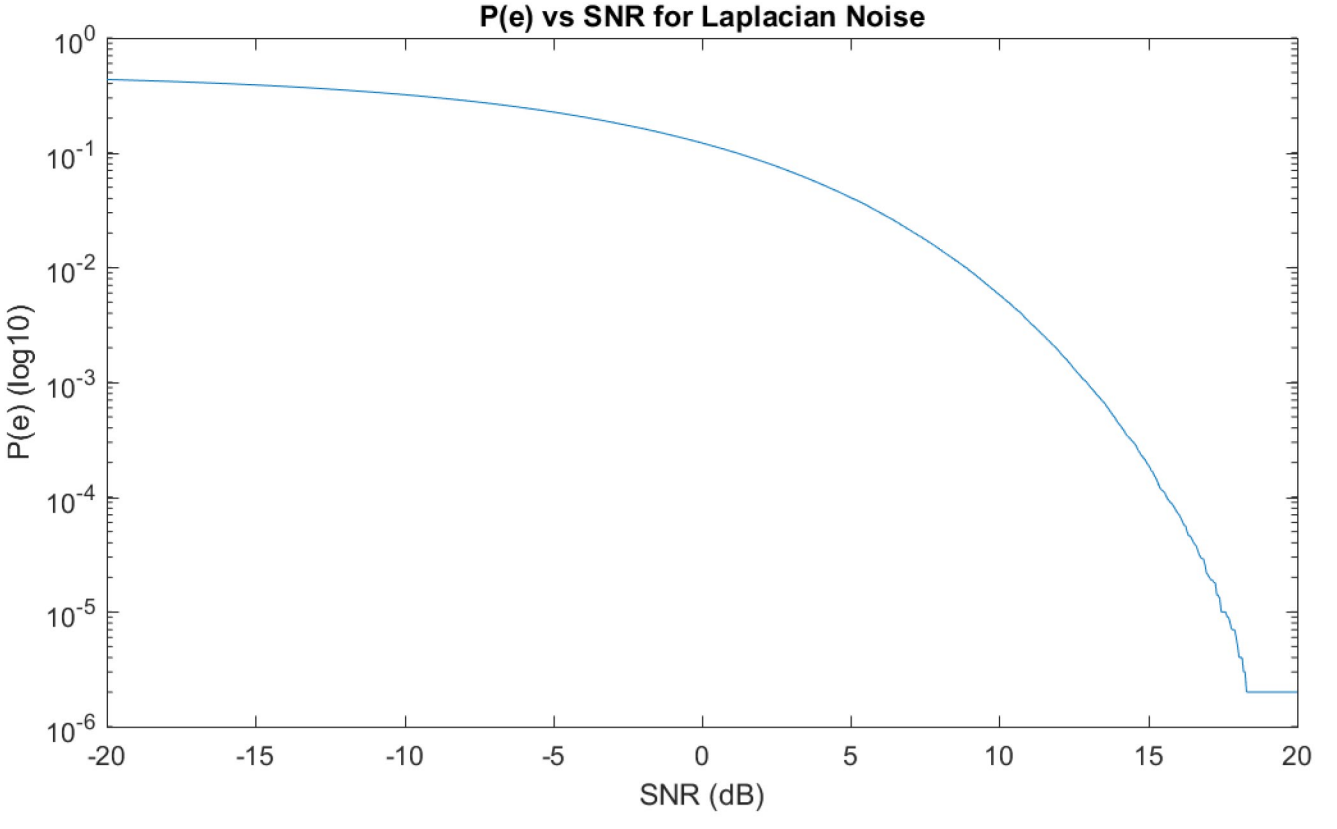
Appendix 3


Histogram of Gaussian Noise RV
proportion
Value
14000
12000
10000
8000
6000
4000
2000
0
-6
-4
-2
0
2
4
6

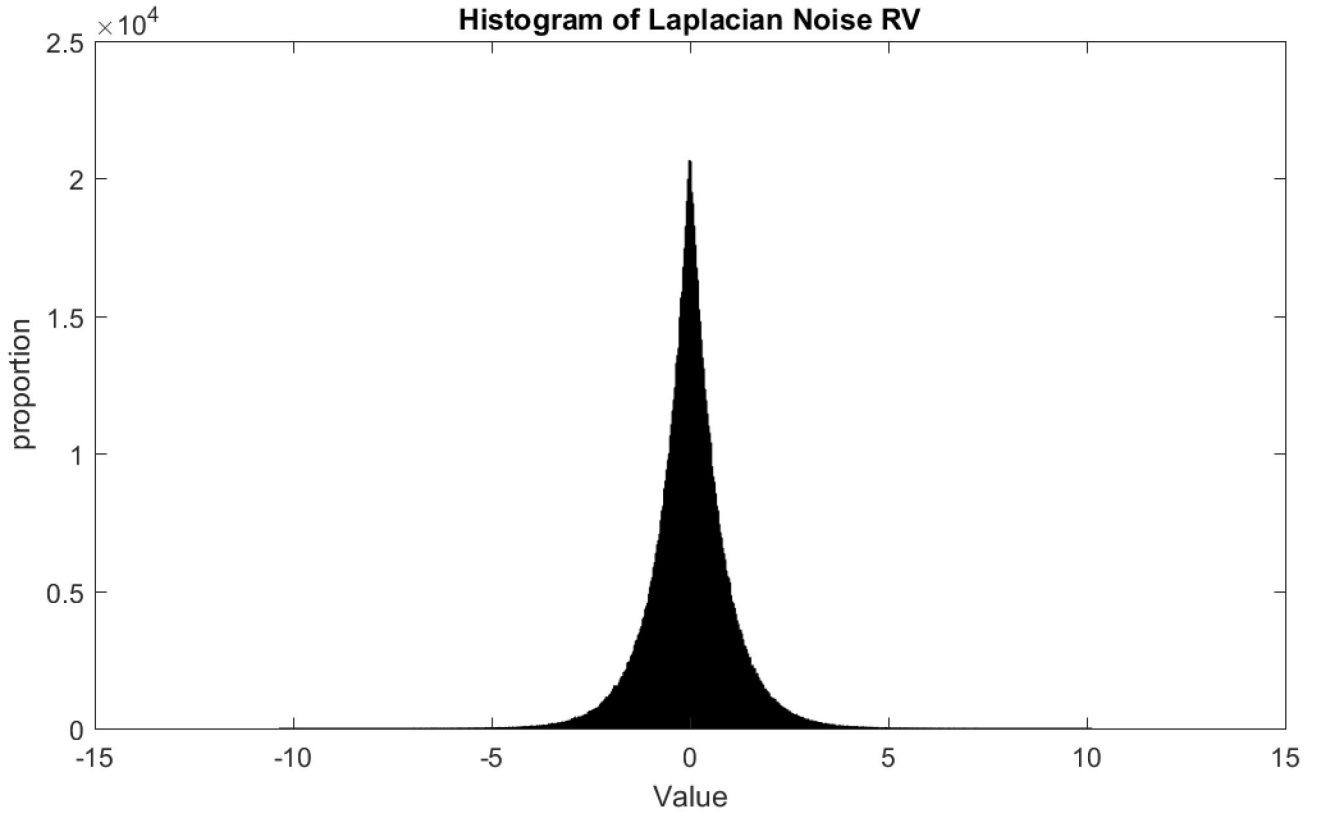

P(e) vs SNR for Gaussian Noise
P(e) (log10)
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
-5
0
5
10
15
20

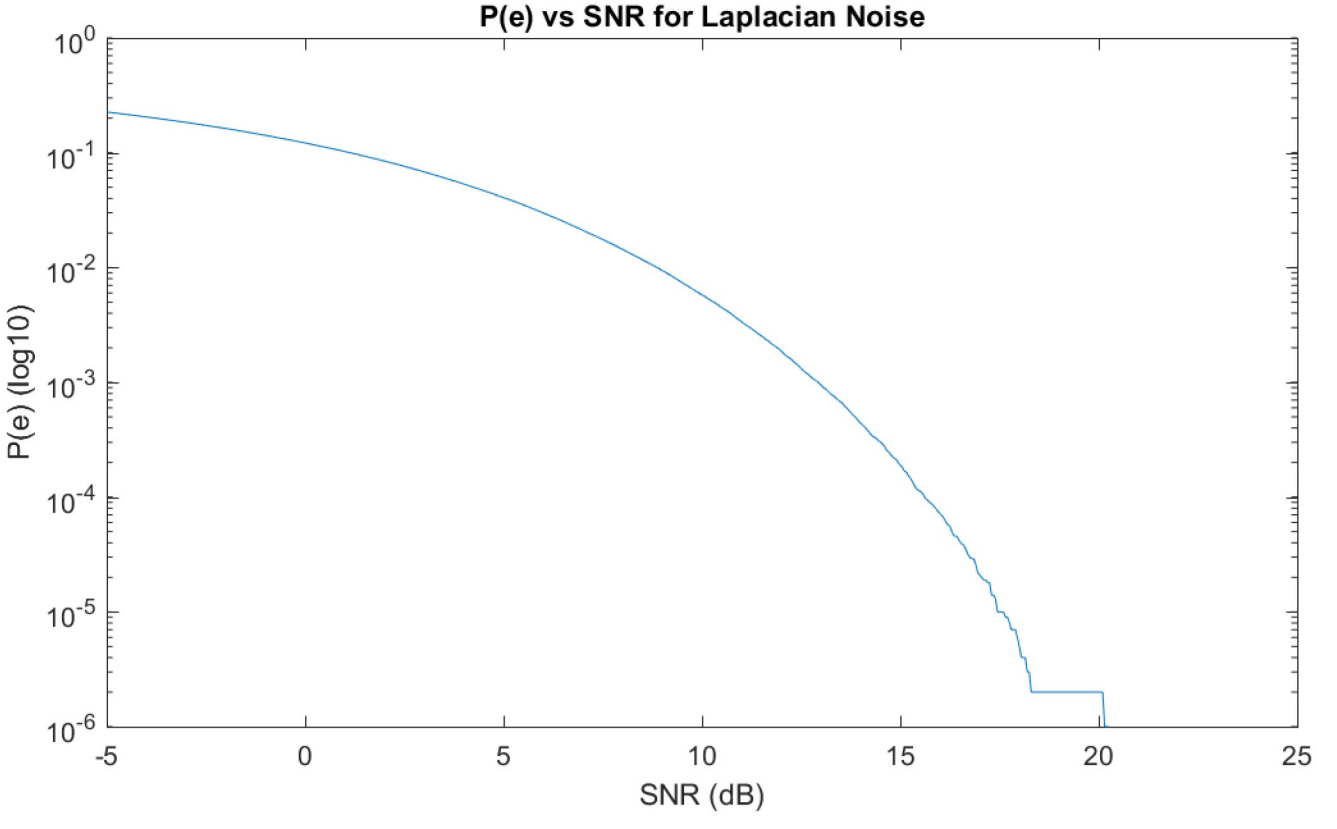
Appendix 4


Histogram of Laplacian Noise RV
×10^4
2.5
2
1.5
1
0.5
0
proportion
-15
-10
-5
0
5
10
15
Value


P(e) vs SNR for Laplacian Noise
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
P(e) (log10)
-20
-15
-10
-5
0
5
10
15
20
SNR (dB)

Appendix 5


Histogram of Laplacian Noise RV
×10^4
2.5
2
1.5
1
0.5
0
proportion
-15
-10
-5
0
5
10
15
Value


P(e) vs SNR for Laplacian Noise
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
P(e) (log10)
-5
0
5
10
15
20
25
SNR (dB)

Appendix 6 – Code for Gaussian simulation

```
% Alexandre Leon 504-941-684
% EE131A project case A
hold off
hold on
subplot(2,1,1);
M = 1000000;
sigma1 = 1;
t = [-20:0.05:20];
P = zeros(1,size(t,2));
%Step 1
randn('seed',2000);
N = randn(1,M);
histogram(N);
title('Histogram of Gaussian Noise RV');
xlabel('Value');
ylabel('proportion');

for u = 1:801
SNR = -20 +(u-1)*0.05;
S = sigma1* 10^(SNR/20);
%Step 2
rand('seed',2001);
S1 = rand(1,M);
S2 = S1 >= 0;
SS = S.*(2*S2 - 1);
%Step 3
X = SS + N;
%Step4
XX = X .*S2;
%Step 5
X1 =( XX <= 0);
%Step 6
M1 = sum(S2);
X11 = sum(X1);
P(1,u) = X11/M1;
end

subplot(2,1,2);

semilogy(t,P)
title('P(e) vs SNR for Gaussian Noise');
xlabel('SNR (dB)');
ylabel('P(e) (log10)');
done = 1
```

Appendix 7– Code for Laplacian simulation

```
% Alexandre Leon 504-941-684
% EE131A project case B
hold off
hold on
subplot(2,1,1);
M = 1000000;
sigma1 = 1;
t = [-20:0.05:20];
P = zeros(1,size(t,2));
%Setp 1 INVERSE METHOD
rand('seed',2002);
N = rand(1,M);
NL = zeros(1,M);
for i = 1:M
NL(1,i) = Inverse_Lap(N(1,i));
end
histogram(NL);
title('Histogram of Laplacian Noise RV');
xlabel('Value');
ylabel('proportion');
for u = 1:801
SNR = -20 +(u-1)*0.05;
S = sigma1* 10^(SNR/20);
%Setp 2
rand('seed',2001);
S1 = rand(1,M);
S2 = S1 >= 0;
SS = S.*(2*S2 - 1);
%Setp 3
X = SS + NL;
%Setp 4
XX = X .*S2;
%Setp 5
X1 =( XX <= 0);
%Setp 6
M1 = sum(S2);
X11 = sum(X1);
P(1,u) = X11/M1;
end

subplot(2,1,2);
semilogy(t,P)
title('P(e) vs SNR for Laplacian Noise');
xlabel('SNR (dB)');
ylabel('P(e) (log10)');

done = 1

function m = Inverse_Lap(x)
m = ( x>0 & x< 0.5)*((1/sqrt(2))*log(2*x))+(x
>=0.5 & x <= 1)*(-(1/sqrt(2))*log(2-2*x));
end
```