**Digital Communication Transceiver Design**

Owen Sowatzke

Georgia Institute of Technology

School of Electrical and Computer Engineering

ECE 4271

Spring 2021

1. **Introduction**

The goal of this project was to simulate a baseband digital communication system using MATLAB. This document details the algorithm that I used and the theory behind it. It, then, describes the theoretical relationship between BER and SNR for BPSK and 4QAM. Finally, it compares the theoretical BPSK and 4QAM results to the results of the simulation.

1. **Algorithm**

The algorithm used for simulation takes three arguments: length of binary sequence, signal-to-noise ratio, and modulation size. It, then, returns the bit error rate of the received sequence. A block diagram of the algorithm is displayed in Figure 1.

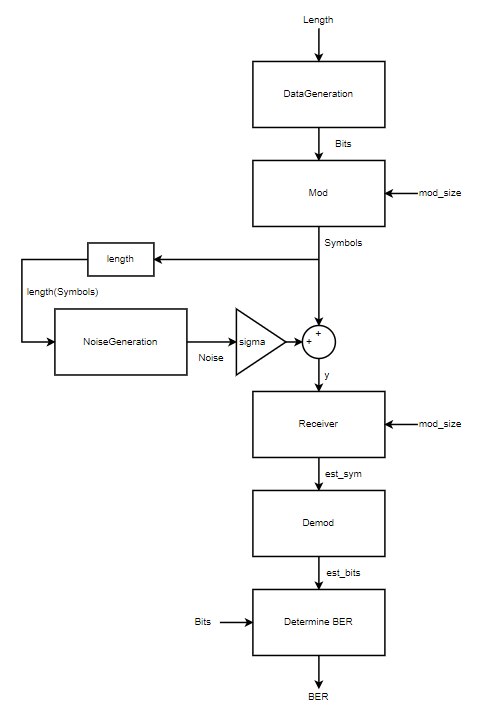


Figure 1 – Block Diagram of the Algorithm Used to Simulate a Baseband Digital Communications System.

* 1. **Data Generation**

The DataGeneration function takes length as an input argument and outputs a (1,length) vector entitled Bits. This vector is a random binary sequence generated by MATLAB’s randi function.

**2.2 Modulation**

The Mod function takes the vector Bits and the scalar input mod\_size as input arguments. It, then, modulates the input binary sequency according to the input modulation size. This function supports modulation sizes of 2 and 4, which correspond to either BPSK or 4QAM modulation. The constellation diagrams used for BPSK and 4QAM modulation are shown in Figures 2 and 3 respectively. After modulation is performed, the sequence of symbols, entitled Symbols, is output by the function Mod.

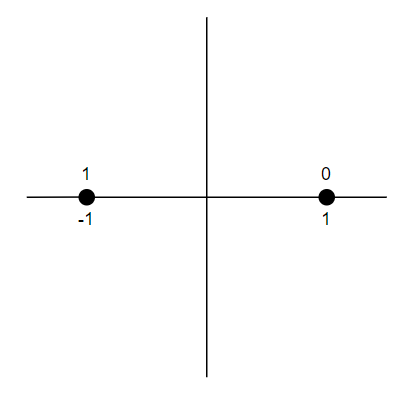


Figure 2 – BPSK Constellation Diagram.

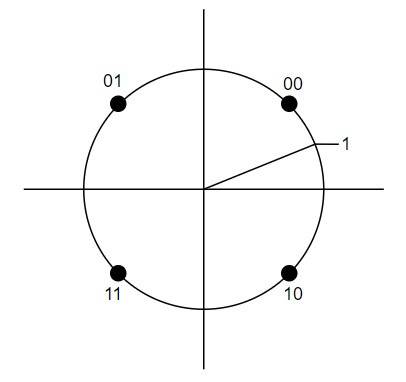


Figure 3 – 4QAM Constellation Diagram using Gray Mapping (i.e. the Nearest Symbol Differs by Only One Bit).

**2.3 Noise Generation**

The NoiseGeneration function takes the length of the Symbols as an input argument and generates complex noise of the same length with unit variance. The complex noise output by the function has the following form:

where and are normal random variables with unit variance.

To generate the received signal, y, the generated noise vector is scaled by sigma and added to the vector Symbols. Note that sigma is the standard deviation of the noise required to generate a sequence with a given SNR. The relationships between SNR and standard deviation are different for BPSK and 4QAM and are described in Formulas (2) and (3) respectively.

The standard deviation for both BPSK and 4QAM can be derived using the relationships in Formulas (2) and (3).

Note that SNR must be converted to linear units before being used in each of the expressions.

**2.4 Receiver**

The Receiver function takes the modulation size, mod\_size, and the received signal, y, as inputs. If the modulation size is 2, BPSK modulation was used, and the decision boundary is the imaginary axis. If the modulation size is 4, 4QAM modulation was used, and the decision boundaries are formed by the real and imaginary axes. Using the decision boundaries, estimated symbols can be determined. The Receiver function returns these estimated symbols in a vector est\_sym. The output vector is formatted according to the constellation diagram corresponding to its modulation size (See Figures 2 and 3).

**2.5 Demodulation**

The Demod function takes the vector of estimated symbols, est\_sym, as an input. Examining the imaginary part of the first symbol, the Demod function determines if BPSK or 4QAM modulation was used. If the imaginary part of the symbol is zero, BPSK modulation was used. Otherwise, 4QAM modulation was used. Using Figures 2 and 3, the estimated binary sequence can be determined. This binary sequence is formatted as a vector entitled est\_bits and is output by the Demod function.

**2.6 Determining the Bit Error Rate**

In the top-level MATLAB function, transceiver, the bit error rate is determined by comparing the transmitted bits, Bits, to the received sequence of bits, est\_bits. This comparison is performed bit-by-bit within a for loop and the number of bit errors is counted. Next, the BER is determined by dividing the number of bit errors by the number of transmitted bits.

**3. Underlying Theory**

The algorithm described above models how data is transmitted between the transmitter and receiver. This section elaborates on the theory behind the constellation diagram and calculates the symbol energy. It also discusses the mechanisms by which demodulation can be performed and elaborates on why gray mapping is used. Finally, it details the role sequence length plays in generating accurate BER results.

**3.1 Interpretation of the Constellation Diagram**

The constellation diagram describes the modulation of the signal at the transmitter prior to transmission. In order to represent different bits or sequences of bits, the amplitude and phase of the modulating sinusoid are varied. These variations are known as ASK (amplitude-shift keying) and PSK (phase-shift keying). The location of points in the constellation diagram captures this information. The amplitude is the distance from the point to the origin and the phase is the angle from the positive real axis to the constellation point.

**3.2 Symbol Energy**

Each constellation point has a unique energy and power associated with it. The power of a sinusoid is described by the following formula:

Energy is described by the integral of power of vs time. Assuming a constant power over the integration time, this relationship simplifies to power multiplied by the integration time. For both the BPSK and 4QAM constellations shown in Figure 1, the amplitude is a constant. Therefore, the energy of each symbol is equivalent and can be described by the following formula:

**3.3 Mechanisms for Demodulation**

Demodulation is performed at the receiver to determine the received symbol. There are two steps in the demodulation process. The first step generates the symbol plus additive noise. This step is performed using matched filters or correlators. Both methods are equivalent. Next, the result of the first step is passed through decision logic to determine the received symbol. The algorithm presented above only performs the second of these two steps. The output of the first step is generated by adding noise from the channel onto the transmitted symbols. Use of constellation diagrams helps to facilitate this process.

**3.4 Gray Mapping**

Gray mapping, shown in Figure 3, is used for 4QAM modulation. In Gray Mapping, the closest constellation point only differs by one bit. This helps to reduce the BER. Symbol errors are most likely to occur between the nearest constellation points. If the nearest constellation point differs by only one bit instead of two, the number of bit errors will only be one instead of two.

**3.5 Role of Sequence Length**

When generating bit error results, it is crucial to ensure that the sequence is adequately long. Consider for example if the bit error should be low (). If a sequence of 10 is chosen, and one of the rate bit error occurs, the calculated BER will be , which is far from the truth. The simulated bit error rate will always get closer to the theoretical bit error rate with an increase in sequence length. However, additional sequence length will slow down simulations. As such, sequence length proves to be a tradeoff.

The prompt recommends bits be used if the theoretical bit error rate is . The smallest expected theoretical bit error rate is below . Therefore, a longer window length than is needed. For the simulations that follow, a window length of bits is chosen.

**4. Theoretical Results**

To determine the accuracy of the algorithm results, the theoretical results are generated for comparison. In the parts that follows, theoretical results are determined for BPSK and for 4QAM.

**4.1 Theoretical BER for BPSK**

The received BPSK symbols vary about -1 and 1 due to the additive white gaussian noise. This result is shown in Figure 4.

Chart, line chart

Description automatically generated

Figure 4 – Probability Distribution Function of Received Symbols. The Shaded Areas Represent Error Probabilities.

As shown in Figure 4, the received symbol is normally distributed about 1 or -1. Using the imaginary axis as the threshold, we find the following:

In other words, the area of each shaded region is equivalent to the probability of bit error. If we integrate the shaded area, we get the following value:

is the standard deviation of the real part of the complex noise . They can be related as follows:

Using the results of Equations (4), (7), (9), and (10) we can relate to .

**4.2 Theoretical BER for 4QAM**

The real and complex parts of the noise are independently distributed random variables. Because the real and imaginary noise are independent, the problem simplifies into two separate problems. Referring to Figure 3, we find that the real part of the noise has the potential to lead to a bit error in the first bit, while the imaginary part of the noise has the potential to lead to a bit error in the second bit. Since the real and imaginary parts of the noise have the same standard deviation, the probability of bit error is equivalent to an error along the real or imaginary axis. This result is summarized in Equation (12).

is the standard deviation of the real part or imaginary part of the complex noise . They can be related as follows:

Using the results of Equations (5), (7), (12), and (13) we can relate to .

**5. Comparison of Algorithm Results to Theoretical Results**

This section uses a binary sequence length of to derive BER results for both BPSK and 4QAM. These results are then compared with the theoretical results.

**5.1 BPSK BER Results**

Figure 5 displays the theoretical and actual BER results for BPSK modulation. Note that both the theoretical and actual results are approximately equivalent.

Chart

Description automatically generated

Figure 5 – Plot of Theoretical and Actual BER Results when BPSK Modulation is Used.

**5.2 4QAM BER Results**

Figure 6 displays the theoretical and actual BER results for 4QAM modulation. Note that both the theoretical and actual results are approximately equivalent.

Chart, line chart

Description automatically generated

Figure 6 – Plot of Theoretical and Actual BER Results when 4QAM Modulation is Used.

**5.3 Comparison of BPSK and 4QAM Modulation**

Examining both plots, we find that the BER is approximately equivalent for BPSK and 4QAM modulation. This result can be confirmed both through reason and mathematically. We would expect BPSK to have a lower BER for a given SNR because the distance to the decision boundary is greater. However, we also know that the noise variance is greater for BPSK. This increase in noise variance offsets the increased distance to the decision boundary. As a result of this offset, BPSK and 4QAM have the same BER for a given SNR. This result has been proven mathematically as well. Referring to equations (11) and (14), we can draw the same conclusion.

**6. Conclusion**

This project successfully simulated a baseband digital communication system using MATLAB. The document describes the algorithm and blocks composing it. These blocks included: data generation, modulation, noise generation, receiver, demodulation, and determination of bit error rate. The theory behind the algorithm was also discussed. This discussion included topics such as the constellation diagram, symbol energy, demodulation techniques, gray mapping, and bit sequence length. After this discussion of algorithm and theory, the theoretical BPSK and 4QAM bit error rates were derived. Finally, the algorithm results were compared to the theoretical results, and were found to be approximately equivalent. Not only this, but the bit error rate for BPSK and 4QAM was also found to be equivalent.

**Appendix A: MATLAB Code**

% Top level MATLAB function

function BER = transceiver(Length,SNR,mod\_size)

% perform error checking of user input

% length argument must be equal to or greater than 1

if Length < 1

error("Function Requires that length >= 1");

% mod\_sizes of 2 and 4 are the only ones supported

elseif mod\_size ~= 2 && mod\_size ~= 4

error("Function Requires that mod\_size be 2 or 4");

% mod\_size of 4 requires an even number of bits

elseif mod\_size == 4 && mod(Length,2) ~= 0

error("Function Requires an Even Length Argument when mod\_size = 4");

end

% generate random bit sequence

Bits = DataGeneration(Length);

% generate symbols for bit sequence

Symbols = Mod(Bits, mod\_size);

% generate additive complex noise with unit variance

Noise = NoiseGeneration(length(Symbols));

% different definitions of standard deviation for BPSK and 4QAM

if mod\_size == 2

sigma = sqrt(1/10^(SNR/10));

else

sigma = sqrt(1/(2\*10^(SNR/10)));

end

% received signal contains both Symbols and Noise

y = Symbols + sigma.\*Noise;

% determine esitmated symbols

est\_sym = receiver(y,mod\_size);

% determine estimated bit sequence

est\_bits = Demod(est\_sym);

% determine number of errors in estimated bit sequence

% initialize variable to store the number of errors

num\_errors = 0;

% loop through bit sequence and count the number of errors

for n = 1:length(Bits)

if est\_bits(n) ~= Bits(n)

num\_errors = num\_errors + 1;

end

end

% determine BER

BER = num\_errors/length(Bits);

end

% function generates a random data sequence of zeros and ones

% with Length bits

function Bits=DataGeneration(Length)

Bits = randi([0,1],1,Length);

end

% function generates symbol sequence for bit sequence given a modulation size

function Symbols = Mod(Bits, mod\_size)

% binary case

if mod\_size == 2

% assume all bits are ones

Symbols = ones(1,length(Bits));

% loop through all bits

for n = 1:length(Bits)

% replace ones with minus one when bit is 0

if Bits(n) == 1

Symbols(n) = -1;

end

end

% 4QAM case

elseif mod\_size == 4

% create empty sequence of symbols

Symbols = zeros(1,length(Bits)/2);

% loop through bits two at a time

for n = 1:length(Bits)/2

% pick a symbol for every 2 bits

if Bits(2\*n-1) == 0 && Bits(2\*n) == 0

Symbols(n) = (1 + 1i)/sqrt(2);

elseif Bits(2\*n-1) == 0 && Bits(2\*n) == 1

Symbols(n) = (-1 + 1i)/sqrt(2);

elseif Bits(2\*n-1) == 1 && Bits(2\*n) == 0

Symbols(n) = (1 - 1i)/sqrt(2);

elseif Bits(2\*n-1) == 1 && Bits(2\*n) == 1

Symbols(n) = (-1 - 1i)/sqrt(2);

end

end

end

end

% function generates complex white Gaussion noise with zero mean

% and unit variance

function Noise = NoiseGeneration(Length)

% generate real noise of with variance 1/sqrt(2)

real\_noise = 1/sqrt(2)\*randn(1,Length);

% generate complex noise with variance 1/sqrt(2)

imag\_noise = 1i/sqrt(2)\*randn(1,Length);

% sum both real and imaginary noise to get complex noise

Noise = real\_noise + imag\_noise;

end

% function returns estimated symbols from noisy received signal

function est\_sym = receiver(y, mod\_size)

% generate empty sequence of estimated symbols

est\_sym = zeros(1,length(y));

% binary case

if mod\_size == 2

% loop through each of the received symbols

for n = 1:length(y)

% determine if the received symbol should be a -1 or a 1

if real(y(n)) < 0

est\_sym(n) = -1;

else

est\_sym(n) = 1;

end

end

% 4 QAM case

elseif mod\_size == 4

% loop through each of the received symbols

for n = 1:length(y)

% make a decision about the real part of the symbol

if real(y(n)) < 0

est\_sym(n) = -1/sqrt(2);

else

est\_sym(n) = 1/sqrt(2);

end

% make a decision about the imaginary part of the symbol

if imag(y(n)) < 0

est\_sym(n) = est\_sym(n) - 1i/sqrt(2);

else

est\_sym(n) = est\_sym(n) + 1i/sqrt(2);

end

end

end

end

% function determines bits from estimated symbols

function est\_bits = Demod(est\_sym)

% all 4QAM symbols have an imaginary part

% determine the modulation size base on the imaginary part of the first symbol

if imag(est\_sym(1)) ~= 0

mod\_size = 4;

else

mod\_size = 2;

end

% binary case

if mod\_size == 2

% assume all bits are ones

est\_bits = zeros(1,length(est\_sym));

% loop through all symbols

for n = 1:length(est\_sym)

% assumption is wrong if symbol is -1

if est\_sym(n) == -1

est\_bits(n) = 1;

end

end

% 4QAM case

elseif mod\_size == 4

% there are 2x as many bits as there are symbols in 4QAM

est\_bits = zeros(1,2\*length(est\_sym));

% loop through all symbols to determine bits

for n = 1:length(est\_sym)

% consider all four cases

if real(est\_sym(n)) > 0 && imag(est\_sym(n)) > 0

est\_bits(2\*n-1:2\*n) = [0 0];

elseif real(est\_sym(n)) < 0 && imag(est\_sym(n)) > 0

est\_bits(2\*n-1:2\*n) = [0 1];

elseif real(est\_sym(n)) < 0 && imag(est\_sym(n)) < 0

est\_bits(2\*n-1:2\*n) = [1 1];

else

est\_bits(2\*n-1:2\*n) = [1 0];

end

end

end

end