**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.

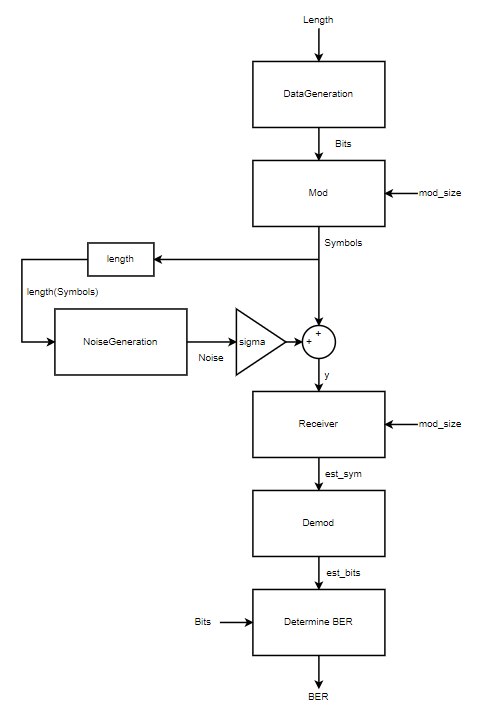


Figure 1 – Block Diagram of the Algorithm Used to Simulate a Baseband Digital Communications System.

* 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.

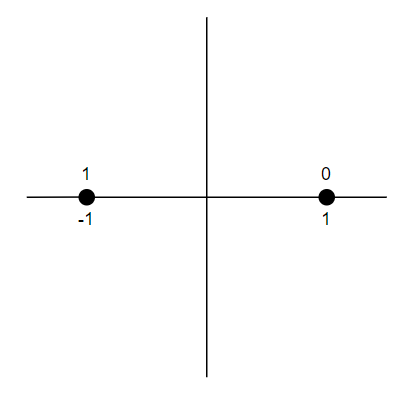


Figure 2 – BPSK Constellation Diagram.

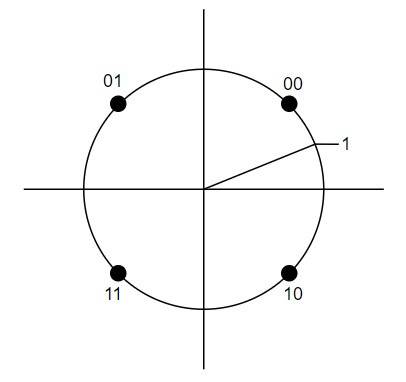


Figure 3 – 4QAM Constellation Diagram using Gray Mapping (i.e. the Nearest Symbol Differs by Only One Bit).

**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. It determines the received symbols in the signal used the decision boundaries shown in Figures

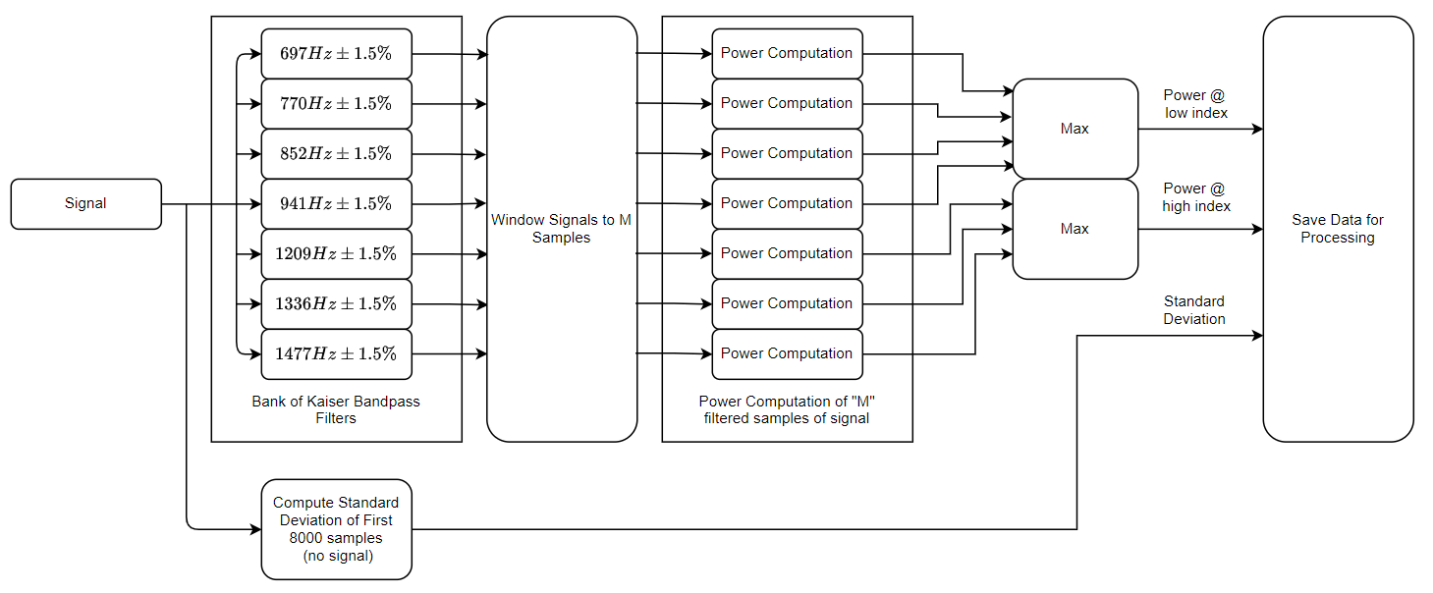


Figure 1 – Block diagram displaying the calculation and flow of data through the power calculation partition of the algorithm.

**2.1.1 Standard Deviation Calculation**

My algorithm estimates the standard deviation of the additive white noise using the final second of the signal. As detailed in tt\_create.m, the final second of signal samples are by definition only noise. Thus, we can efficiently use this portion of the second to calculate the standard deviation. Let be the number of samples in one second of the signal . In my algorithm, Using this parameter, the standard deviation can be approximated with the following formula.

**2.1.2 Power Calculations of the Windowed Signal**

Let the signal be windowed by a rectangle window of length . Each signal is, then, filtered with Kaiser bandpass filter. The passband and transition band of each filter is defined to meet the and tolerances of passband and stopband respectively. More details on the passband and transition bands is detailed in section 3.3’s tradeoff description.

In my algorithm, each filter was implemented in MATLAB using the kaiserord and fir1 commands. Using equation (1) and the window size, in place of , I calculated the power of each frequency during each window. This information was saved in an array with dimensions where describes the number of windows and 7 is the number of DTMF frequencies.

* 1. **Threshold Calculations**

There was only one threshold that I used in my algorithm. It is the power required for a signal to be declared valid. This threshold is related to the probability density function of the filtered noise. At high SNR’s, the larger signal amplitudes and steeper transitions can be blurred by the filter. As such the transition width should be chosen based on the amplitude and not on the probability density function. A simplified block diagram describing the threshold calculation is shown in Figure 2.

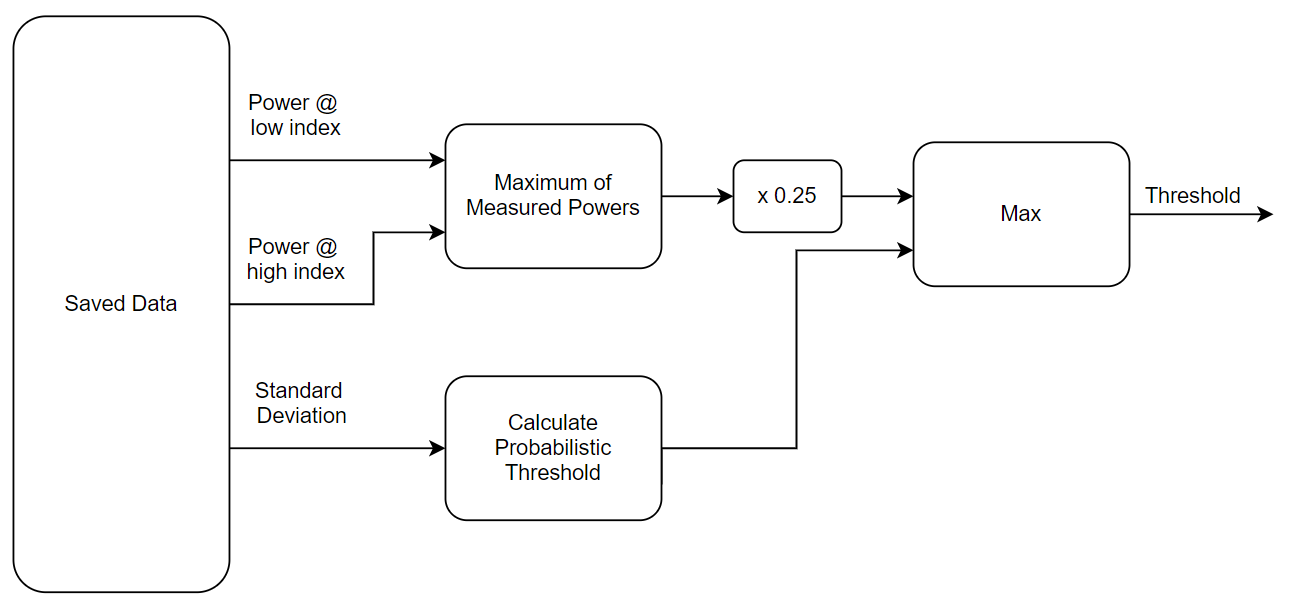


Figure 2 – Block diagram displaying the calculation of the threshold value.

The additive white noise is described by Gaussian random variables with unit mean. The noise output by the bandpass filter retains the same distribution, with a scaled variance. This relationship is described in equation (2), where is the variance of the unfiltered noise and is the variance of the filtered noise. For stricter tolerance, is described to contain the widest filter’s passband and transition bands.

The density function of the measured power of filtered white-noise samples is described according to a chi-squared distribution of orders of freedom. Note that the chi-squared distribution is based on samples of unit variance. Therefore, the pdf I derived has the following form:

The probability of false alarm is described by the cdf function of this distribution. Using the following line of MATLAB code, we can calculate a probability-based threshold.

power\_thres = var/M\*chi2inv(1-Pfa,M);

In this line of code, the variance is described by the variable var, and can be related to the variance found in part 1 using equation (4). The other parameters of interest, M and Pfa, are the window size and the probability of false alarm respectively.

* 1. **Decision Logic**

My algorithm decision logic operates on the highest received power values to determine if they describe a valid signal. The flow of this logic is shown in figure 3.

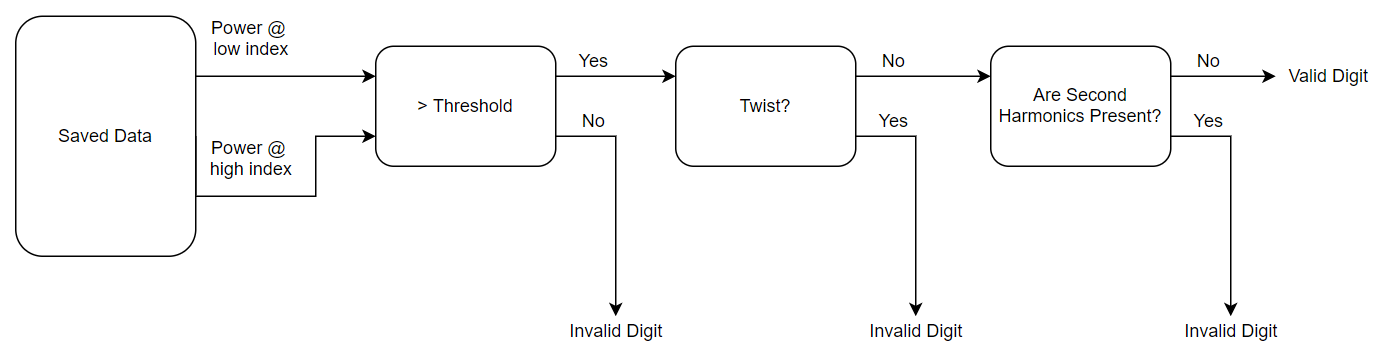


Figure 3 – Block Diagram describing the decision logic that determines if received power values describe a valid digit.

First, the maximum power values in each band are compared with the threshold value. If they exceed the threshold value, a twist test is performed. In order for the received power values to pass this test, the low frequency power must be no more than above the high frequency power. Additionally, the high frequency power must be no more than above the low frequency power.

After this test is performed, a second harmonics check is performed. The frequency tolerances are less important for this application. As such, I choose the goertzel algorithm to calculate the magnitude of these values. Now instead of having to pre-filter the signal with 7 more bandpass filters, I can find the frequency component of the second harmonics only when I need them. In other words, this minimized computational delays.

To maximize the accuracy of DFT samples, zero-padding is used to extend the signal to a length . Then, the Goertzel algorithm is applied on the desired indices. Finally, using the measured value of at the desired frequency index, we can approximate the power of the second harmonics using the following formula.

is approximately the power of a cosine. The measured amplitude is scaled by the maximum value in the frequency domain. This value is equivalent to the window size . Additionally, note that . As a result, the power of the second harmonics can be approximated by equation (4). This approximation and comparison with the power are displayed in Figure 4. Note that scaling the signal power by a constant factor does reduce the probability of false second harmonics being detected. At low SNR’s (below 0) I scaled the signal power by 4 for better results.

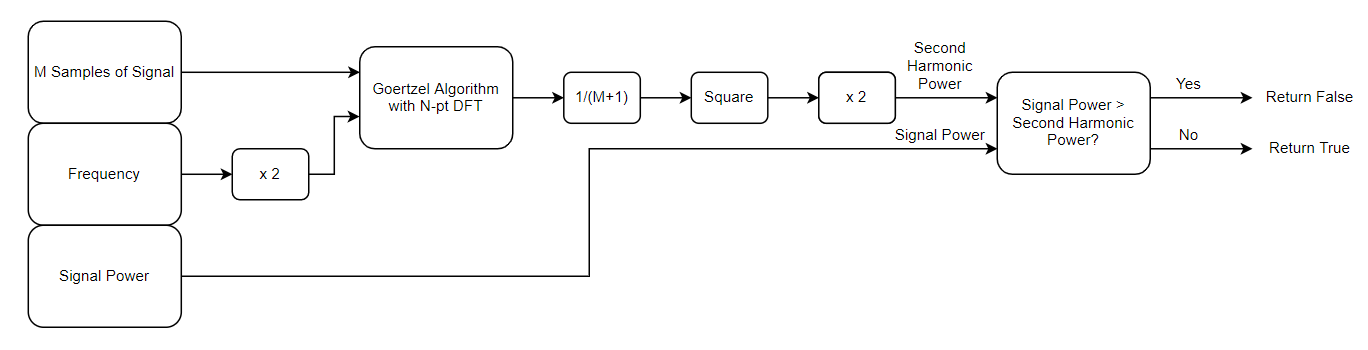


Figure 4 – Power approximation and comparison to desired value. This block diagram returns if there are second harmonics and if there is none.

Note that the process in Figure 4 is executed for low frequency and high frequency power maximums.

Using both Figure 3 and Figure 4, one can trace how a valid digit is determined. At the most abstract level, the decision of which digits to output is related to the window size. For example, if the window size is small, we may need multiple detections before a signal is declared valid or a pause is declared valid. In my algorithm, I choose as described in section 3.2. For this value of , we know that there must be at least one invalid digit returned for each pause and at least two valid and identical digits for each valid digit.

To ensure that valid digits were not printed multiple times, I kept track of when a string of two valid digits was found. This value was then only saved once this two-digit string of digits ceased. Using this two-digit validation allowed me to further reduce the probability of false alarm.

1. **Algorithm and Parameter Tradeoffs**

There are a variety of tradeoffs regarding both algorithm and parameters. The first of these tradeoffs regard choosing a bandpass filter implementation instead of a goertzel filter implementation. The remaining tradeoffs involve the window size, bandpass filter specifications, and the DFT size when calculating second harmonics.

* 1. **Bandpass vs Goertzel Filter Digit Detection**

A bandpass filter implementation requires additional computations as opposed to performing the DFT. In MATLAB, bandpass filtering requires the FFT of the filter and signal and IFFT of the filtered result. However, when using a bandpass filter, we can very precisely set frequency specifications of passed signals. Conversely, when using the FFT on a window of size M, frequency components that should not be recognized are convolved with a function. The main lobe width of this function is described by the following formula.

At 3.5% frequency error, we should not detect a signal. For this to occur, let us assume that main lobe of the error signal should not overlap the center frequency. Then, the following relationship describes the range of .

Note this calculation does not account for any error in the sampled frequency or noise. Both of these can further increase required tolerances for M.

Additionally, the DFT does not account for frequencies within of the desired frequency. Choosing a small , makes these values to be visible. However, a large makes these values harder to detect. They can be detected with additional DFT samples and logic. However, we can achieve the same effect using a bandpass filter without having to worry about our window size and adding additional logic. For this reason, I chose to use a bandpass filter for signal detection.

* 1. **Window Size**

As established in section 3.1, bandpass filters are not as susceptible to our choices in window size. As such the main constraint on our signals is the length of digits and pauses. According to tt\_create.m, each valid digit is at least long, and each pause is at least long. I chose the window size, , such that there was at least one full window during each pause. To accomplish this, I let . Using this value for , there must be at least 2 valid digit detections for each valid digit.

A large value of is desirable such that the pauses less than ms have a minimal effect on the measured power. For this reason, we choose , instead of using multiple small windows of length . Note it may be possible to use a smaller or larger with smaller step sizes. However, this requires additional logic, and I have received desired results without these additional complications.

* 1. **Bandpass Filter Specifications**

There are two primary bandpass filter specifications that have a large effect on our system. These include the bandpass filter length and the size of our passband.

* + 1. **Bandpass Filter Length**

Increasing the bandpass filter length influences the computational delays. However, a larger filter can provide a steeper transition from passband to stopband and minimize ripple. For this reason, we desire to minimize the filter length while meeting transition and ripple specifications. To accomplish this, I assigned each filtered signal a constant transition band width that is defined on either side of the passband defined by the following expression:

where is the smallest frequency we wish to detect. This transition band width ensures that frequencies from the desired frequency are not detected. A constant value for the transition band width ensures that all band pass filters are of the same order and contribute the same known group delay.

* + 1. **Passband Size**

The passband size is another design tradeoff. Choosing a transition width that is too small passes less white noise, however it also attenuates transitionary portions of the signal. Specifically, a single turning on or off is described by a function. In the frequency domain, this is a function. A function requires all frequencies. Passing too few of these frequencies, can affect our ability to see signal transitions. Refer to Figure 5 for context.

Chart, histogram

Description automatically generated

Figure 5 – Outputs of Bandpass Filter, using desired specs (top) and using a narrower passband (bottom). Note it can be hard to see when the signal transitions if the passband is too small.

In my algorithm, I defined the passband of each filter according to the following equation.

In this formula, is the frequency we wish to pass and is the sampling rate. The constants and are chosen such that the frequency is passed to a tolerance of .

* 1. **DFT Size for Second Harmonic Computations**

The DFT used to compute second harmonics is zero-padded to some length , to get the most accurate measurement of the second harmonics. This parameter must be greater than , the window size, and should place the sampled frequency indices as close as possible to the second harmonic locations. I created a MATLAB script to optimize my choice of . To speedup processing I chose a low value for , that still minimized error. My choice of is displayed in Figure 6.

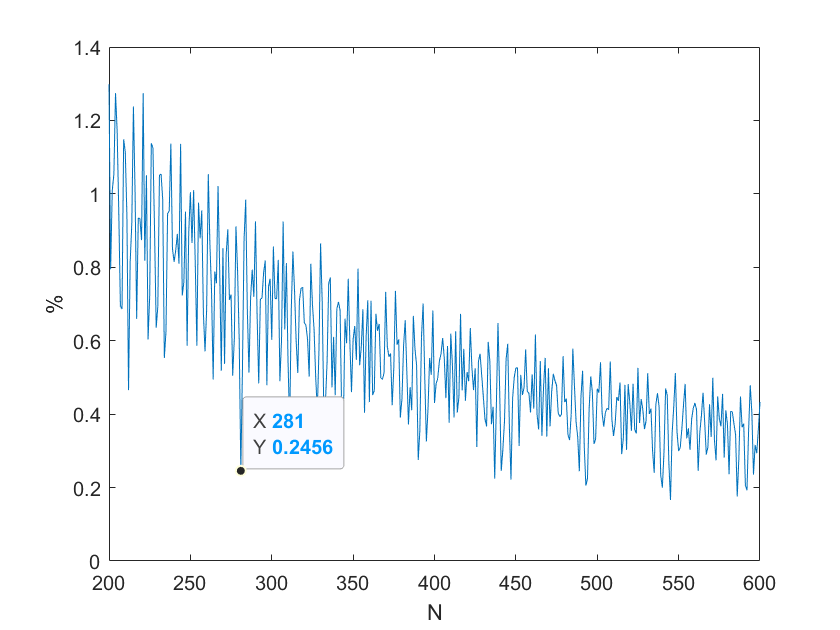


Figure 6 – Plot of maximum error between sampled frequency and desired frequency. I chose , a local minimum, to minimize both and error.

1. **Results**

Over the SNR range of interest (0 dB to 50 dB), the algorithm output 100 random phone numbers without error at each 5 dB increment without error. This is displayed in Figure 7.

Chart, histogram

Description automatically generated

Figure 7 Percentage of correct phone numbers when given 100 random phone number to decode. Note the percentage of error is 0% over the entire range.

Examining the results at lower SNR’s, we find that we can accurately decode -5dB nearly 100% of the time. However, as the SNR falls, the percentage of correct phone detections decays rapidly. This result is displayed in Figure 8.

**Chart

Description automatically generated**

Figure 8 – Percentage of correct phone numbers when given 100 random phone number to decode. Note the percentage of error is until about -5 dB, after which it rapidly falls.

As demonstrated by both Figure 7 and 8, the DTMF decode is effective until the SNR falls below -5dB. This level of performance has been achieved using bandpass filters and a variety of checks which include thresholds, twist, and second harmonics. Note that because of bandpass filtering, no frequency tolerance checks are required. Parameter optimizations, increased DFT usage, and/or the amount of window sliding can further be used to improve results. Future work will investigate the effects of each.