Efficient Modelling and Performance Analysis of Wideband Communication Receivers

Andreas Eriksson
Abstract

Efficient Modelling and Performance Analysis of Wideband Communication Receivers

Andreas Eriksson

This thesis deals with Symbol Error Rate (SER)-simulation of wireless communications and its application into throughput analysis of UltraWideband (UWB) systems. The SERs will be simulated in C++ using the Monte Carlo method and when some are calculated, the rest will be estimated using a novel extrapolation method. These SER values will be very accurate and in this thesis go as low as 1.0e-14. Reaching that low values would otherwise be impossible using the traditional Monte Carlo method, because of very large computation time. However, the novel extrapolation method, can simulate a SER-curve in less than 30 seconds. It is assumed that the noise belongs to the Generalized Gaussian distribution family and among them noise from the Normal distribution (Gaussian noise) gives the best result. It is to be noted that Gaussian noise is the most commonly used in digital communication simulations. Although the program is used for throughput analysis of UWB, the program could easily be adapted to various signals. In this thesis, throughput analysis means a plot with symbol rate vs distance. From any given symbols, the user can, with a desired minimum SER, generate an extrapolated SER-curve and see what symbol rate can be achieved by the system, while obeying power constraints of signals imposed by international laws. The developed program is, by comparing with published theoretical results, tested for QAM and PSK cases, but can easily be extended to UWB systems.
Contents

1 Introduction iii

2 Theoretical Background 1
  2.1 Symbol error rate 3
  2.2 Signal modulations 4
  2.3 Generalized Gaussian distribution family 6
     AWGN 7
     Generating random variables 8
     Generate Uniform Pseudo-random variables 8
     Generate Gaussian noise from the Uniform distribution 9
     Generate noise from any distribution in the exponential family 9
  2.4 Simulation of Symbol error rates 11
     The Monte Carlo method 12
     Sequential estimation 12
     Minimum simulation time 13
  2.5 Efficient Simulation 13

3 Novel Tail Extrapolation method 15

4 Ultra Wideband 21
  4.1 Throughput for ultra-wideband 21

5 Result 25
  5.1 Novel Tail Extrapolation method 25
     Results using theoretical values as reliable bins 25
     Results using known noise 27
     Results using unknown noise 28
     Results and comparisons using different amount of errors in sequential estimation 28
     Results using randomly reliable bins 30
     Time Efficiency using the novel extrapolation method 31
  5.2 Ultra Wideband simulation results 32

6 Conclusion 39
  The performance of the extrapolation method 39
  When to rely on the results 39
## CONTENTS

- Best procedure to get good extrapolation results 39
- The poor result with other noises than Gaussian (V=2) noise 39
- Using many symbols 40

7 Future Work 41

8 Manual for C++ classes 43
  8.1 Manual introduction 43
  8.2 How to work with the classes 43
  8.3 Tutorials 45
  8.4 The contents of the classes 48

Bibliography 53
The primary goal of this thesis is to speed up simulation time while generating Symbol Error Rates (SER) curves and to use this technique for Ultra Wideband (UWB) systems. However, because of lack of published results for UWB systems, presentation of results in form of graphs and tables will be based on transmissions with Quadrature Amplitude Modulated (QAM) signals and Phase Shift Keying (PSK) modulated signals. The reason for that is that there are SER formulas for those modulations so theoretical SER-values can be plotted next to simulated ones.

In digital communications it is common that the symbols are coded, so when the receiver misjudge a symbol for another one, the error may be corrected. Coded symbols either require more bits per symbol, or less information bits per symbol. Either way, coding will lower the bit rate. That is bad for the throughput, so in some cases it may not be so important to code the symbols, for example when streaming live videos etc. Corrupted signals will only make a pixel wrong here and there and often, when there are not too many incorrect pixels, the human eye will neglect it. Some transmitting purposes demands different requirements for the symbol error rate and that is why it is important to have a SER-curve so the user knows at what Signal-To-Noise-Ratio (SNR) to transmit. The SER-curve can be derived by the use of pure Monte Carlo simulations, but that is very time consuming. Another approach is to simulate just a couple of high SER points with the Monte Carlo method, which does not take that much time, and then estimate the rest of the curve based on these values. The most famous estimation method for SER is the Weinstein extrapolation method, which was published in 1971. That is a method that works good for moderate SERs, but cannot be trusted for really small SER. There is a novel extrapolation method [6], published in January 2009, that work much better then the Weinstein method. In this thesis, the method presented in [6], is used to develop a program in C++ that extrapolates these SER-curves and then computes the throughput vs distance. The throughput analysis is aimed for UWB, because it uses a parameter for maximum allowed transmitting power for unlicensed signals, which applies for UWB signals. UWB is an upcoming technology which uses high-speed transmission over short ranges. It is very useful in wireless applications, such as wireless audio and video transmissions because of the large bit rate. For more information about UWB, the reader is referred to the theory section.
2

THEORETICAL BACKGROUND

2.1 Symbol error rate

This thesis will only focus on SER, even though calculation of the Bit Error Rate (BER) stem from the same theory. The meaning of SER, is the amount of incorrectly received symbols relative to the total amount of transmitted symbols. The definition for an error is, if the receiver misjudge a received symbol to be another symbol, which could happen, e.g. if the channel is noisy or is subject to interference. The common way to illustrate a channels SER is with a plot with axis SER vs SNR. The SNR is defined as the power of a transmitted signal divided with the power of the noise,

$$SNR = \frac{P_{signal}}{P_{noise}}.$$  (2.1)

The higher the SNR, the less the signal will be affected by noise. In computer simulations the power is calculated from a sampled signal. The signal is quantized into bins\(^1\) and the power of the signal is the sum of all the bins squared and divided with the amount of bins,

$$P_{signal} = \frac{1}{N} \sum_{i=1}^{N} s^2(i).$$  (2.2)

A typical sampled signal looks like the one in Fig. 2.1. When working with QAM-modulation, see Section 2.2, the amplitudes and phases between the symbols will differ. Different amplitudes gives different power, so in the SER plot it is needed to use average power, which is calculated by adding all the powers of the symbols and divide it with the number of symbols, M,

$$P_{average} = \frac{1}{M} \sum_{i=1}^{M} P_i.$$  (2.3)

In this thesis we shall use symbol energy divided by the noise, i.e. \(\frac{E_s}{N_0}\) instead of SNR and write it as EsN0, because it is a more common measure in literature. The conversion from SNR to EsN0 is,

$$E_sN0 = SNR \cdot T_s \cdot B = \frac{SNR \cdot T_s \cdot F_s}{2}.$$  (2.4)

\(^1\)A bin may consist of one or several samples. Here we shall assume that a bin consists on one sample.
CHAPTER 2. THEORETICAL BACKGROUND

Figure 2.1: A sampled signal, quantized into N bins

where,

Ts = symbol time
B = bandwidth
Fs = sampling frequency.

When knowing the average power for the signals and the noise power it is now possible to plot SER vs SNR or EsN0. For 16-QAM, the plot will look like Fig. 2.2. The curve is generated with Monte Carlo simulations and due to large computational time, it is very difficult to have SER-values lower than 0.0005. By speeding up simulations it is easy to reach much lower values, which can be studied in Section 2.5.

2.2 Signal modulations

To transmit data over a channel it is necessary to use a carrier wave to carry the information. This carrier wave can have different appearances but is often a sinusoidal wave. The carrier wave is then modulated so the receiver can distinguish each symbol from another and the most common parameters to vary are the amplitudes, phases and frequencies. Quadrature amplitude modulation distinguishes the symbols from each other by varying
2.2. SIGNAL MODULATIONS

Figure 2.2: SER vs EsN0. The error rate goes down with increasing EsN0.

both amplitude and phase, while phase shift keying modulation only varies the phase. A normal naming for the modulations is by first mentioning the amount of symbols and then the modulation type, for example 16PSK, 64QAM etc. To illustrate these symbols it is common to show them in a signal constellation diagram, which is a plot in complex plane. The imaginary axis is called the Quadrature axis (Q-axis) and the real axis is called the Inphase axis (I-axis), see Fig. 2.3, left image. Every symbol can be written as a complex number, the imaginary part is carried by one wave and the real part by another.

$$A(t) \cdot \sin(2\pi f t + \phi(t)) = I(t) \cdot \sin(2\pi f t) + Q(t) \cdot \cos(2\pi f t)$$

where

$$I(t) = A(t) \cdot \cos(\phi(t)) \quad Q(t) = A(t) \cdot \sin(\phi(t)).$$

It is the in-phase value and the quadrature value for each symbol, that are plotted against each other in the signal constellation diagram. The points in the diagram is only at the correct places if the signals has not been disturbed by noise. If they are disturbed by noise, they can end up anywhere in the complex plane, but if the SNR is high they will end up close to the correct location. Either way, the signals will end up at any location within some specific area, and depending on which region, coloured sectors of the right hand diagram of Fig. 2.3, the receiver will estimate the signal as the corresponding symbol. The areas do not have to be illustrated, it is enough to see which distance is the smallest when calculating the distance from the received signal with all the expected symbol locations. The minimum distance is also called the Euclidean distance. Here it is possible
to see a connection between the SER and the signal constellation, because each time a corrupted symbol is estimated incorrectly, an error will occur. When dividing the number of errors with the number of transmitted symbols when transmitting a very large number of symbols, the theoretical SER for the used modulation will be approached. Often when simulating SERs, it is common to let Gaussian noise disturb the transmitted signals. Gaussian noise is part of the generalized Gaussian distribution family.

### 2.3 Generalized Gaussian distribution family

The generalized Gaussian distribution, represented by its probability density function (PDF), is given by

$$f(x) = \frac{V}{2\alpha \Gamma(\frac{1}{\alpha})} e^{-(|x-\mu|/\alpha)^V}, \quad (2.6)$$

with standard deviation

$$\sigma = \sqrt{\frac{\alpha^2 \Gamma\left(\frac{3}{V}\right)}{\Gamma\left(\frac{1}{V}\right)}}, \quad (2.7)$$

is a function of probability density versus the random variable. An integral of the PDF from one threshold to another represents the probability that a random variable will have its outcomes in that interval. The Cumulative Probability Function (CPF) is the integral of the PDF from $-\infty$ to $x$,

$$F(x) = \int_{-\infty}^{x} f(t)dt. \quad (2.8)$$
2.3. GENERALIZED GAUSSIAN DISTRIBUTION FAMILY

With the CPF the PDF gets a practical meaning. The PDF could vary in shape depending on the variable $V$, which is called the shape parameter. Different values for $V$ produce different distributions, but all distributions are said to be in the same exponential family, or more called, the Generalized Gaussian distribution family. Low values for $V$ makes the distribution thinner and more spiky, and high values for $V$ makes it more uniform. As the value grows, the distributions converge to the same shape.

For $V=1$, we obtain the Laplacian distribution, which is used in many technical areas, but also used by financial and economic experts. For $V=2$, the distribution is called the Normal distribution, or Gaussian distribution, and is often used in signal processing for AWGN (Additive white Gaussian noise) but also in many areas of business administration like price changes of stocks and also in statistics. It is also common to use non-integer values for $V$, such as the Gamma distribution where $V=0.5$, which is used, e.g. in speech modeling [3]. This thesis will focus on the Normal distribution, because AWGN is the most common noise in digital communication.

![Figure 2.4: Multivariate normal distribution, or probability density function with two variables.](image)

**AWGN**

The additive white Gaussian noise channel is often used in digital communication simulations. It is easy to work with because the white noise has constant magnitude in all frequencies, in other words constant power spectral density. Continuous white signals have infinite power, because the bandwidth is infinite. In practice though, it is impossible to have unlimited bandwidth so white noise there means that the power are constant between some pre-defined frequencies. Some noise sources in real life are often thought
CHAPTER 2. THEORETICAL BACKGROUND

of as wideband Gaussian noise, for example thermal noise, which occur in any electronic device because of movement by charged particles when the device gets hot, even when the conductor is in charged equilibrium. The use of AWGN channels is actually an approximation, which works well in some applications but is too coarse for others, e.g. terrestrial channels. It will however make simulations and calculations a lot easier. It only includes noises that comes from, for example traffic, mechanical noises, cosmic background radiations etc. What the AWGN model does not add is multi path, interference and different kinds of fading.

Generating random variables

The ability to generate generalized Gaussian noise for simulations is widely needed in digital communication. Every method for doing this relies of a good random number generator, which in general uniformly distributes outcomes. Quite often these generators turns out to be quite poor. After the uniform number is generated it will be transformed with other methods to Gaussian noise etc. There are several ways of generating uniform random variables. The ideal way is to use True Random Number Generators (TRNGs), but it is seldom used in simulations due to the long computation time. TRNGs are hardware generators, a piece of electronics, which is connected to the computer and distributes amplified thermal noise. In simulations it is more common to use Pseudo-Random Number Generators (PRNGs), which often work well.

Generate Uniform Pseudo-random variables

Some PRNGs will be presented here, but just with some general and easy to understand information. Common PRNGs are [5]:

**Linear Congruential Generator:**
LCG uses a transition function \( x_{n+1} = (ax_n + c) \mod m \). The maximum period is \( m \), which means that a 32-bit integer has a period of \( 2^{32} \). The period is too small for the Monte Carlo simulations that is used in this thesis and the generated numbers are not sufficiently uncorrelated to be useful.

**Multiple Recursive Generator:**
MRG is an expansion of the LCG, \( x_n = (a_1x_{n-1} + a_2x_{n-2} + ... + a_kx_{n-k}) \mod m \). It uses multiple LCG combined and the periods can be much larger than the single LCG.

**Lagged Fibonacci Generator:**
This generator works almost as the LCG, but with one more feedback. The transition function could either use addition between the feedbacks \( x_{n+1} = (x_n + x_{n-k}) \) or multiplication \( x_{n+1} = (x_n \times x_{n-k}) \). The constant lag \( k \) should be large, normally greater than 1000, so the algorithm is dependent on previous distant outcomes.

**Mersenne Twister:**
This is a very complex generator, but with good statistical quality. It has a period of \( 2^{19,937} \). The period is bigger than any simulation application may need and is the used uniform random generator in Maple, MATLAB, Python, Ruby etc.


2.3. GENERALIZED GAUSSIAN DISTRIBUTION FAMILY

Combined Tausworthe Generator:
The combined tausworthe generator works almost as Marsenne Twister but is even better. Marsenne Twister uses a binary matrix to transform a vector of bits into a new vector of bits. The matrix and the vectors can be extremely big and the combined taurusworthe generator can use smaller ones. Some correlations may occur which of course is a drawback.

Generate Gaussian noise from the Uniform distribution
The uniform random variables, obtained with any of the methods described above can be used to create generalized Gaussian distribution random variables. We describe two methods next.

The Ziggurat Method:
The Ziggurat Method is a very fast method and focuses on dividing the Gaussian probability density function into rectangles and the rectangles are put so the calculation cost for generating a random variable is minimized. A drawback with this method is that for 2% of the generated values the method has to take an alternative route which contains even more uniform generated numbers.

The Box-Muller Transform
This method is commonly used. It uses two of the earlier uniformly generated samples and creates two Gaussian generated samples with the following equations:

\[ g_0 = \cos(2\pi u_2) \sqrt{-2 \ln(u_1)}, \quad g_1 = \sin(2\pi u_2) \sqrt{-2 \ln(u_1)}. \]  

(2.9)

It is a good method but it is computational demanding because of the sine and cosine calculations. The method is also efficient when using parallel computing architectures due to the lack of loops and branches. If the equations are written in polar form, we obtain:

\[ g_0 = u \sqrt{-2 \ln(R^2) / R^2}, \quad g_1 = v \sqrt{-2 \ln(R^2) / R^2}. \]  

(2.10)

where

\[ u = R \cos(2\pi u_2), \quad v = R \sin(2\pi u_2), \quad R = u^2 + v^2. \]  

(2.11)

The mean and variance for the generated random variables can easily be modified with an added bias for the mean and a scaling factor for the variance.

Generate noise from any distribution in the exponential family
In a few areas in signal processing it may be useful to generate different kinds of noise depending of what field the simulations are made in. There is a novel method for doing this [4] and it is open for anyone to implement in their own program. It is used in this
thesis for generating noise with $V=3$. It is a good number generator but it is quite time
demanding because of the use of the Gamma function. The method is described briefly
below.

Start with the generalized Gaussian distribution with mean value equal to zero:

$$ f(x) = \frac{V}{2a\Gamma\left(\frac{V}{a}\right)} e^{-\left|\frac{x}{\alpha}\right|^V}. \quad (2.12) $$

The cumulative distribution function will be

$$ F(x) = \frac{\Gamma\left(\frac{1}{V},\left(-\frac{x}{\alpha}\right)^V\right)}{2\Gamma\left(\frac{1}{V}\right)} \quad \text{if } x \leq 0 $$

$$ F(x) = 1 - \frac{\Gamma\left(\frac{1}{V},\left(\frac{x}{\alpha}\right)^V\right)}{2\Gamma\left(\frac{1}{V}\right)} \quad \text{if } x \geq 0 \quad (2.13) $$

where $\Gamma(\cdot,\cdot)$ is the incomplete gamma function, i.e.,

$$ \Gamma(a,x) = \int_x^\infty t^{a-1} e^{-t} dt. \quad (2.14) $$

When generating the random variables the regularized incomplete gamma function will
be used. It looks like follows:

$$ \Delta(a,x) = \frac{\Gamma(a,x)}{\Gamma(a)} \quad (2.15) $$

The cumulative distribution can now be expressed as follows:

$$ F(x) = \frac{1}{2} \Delta\left(\frac{1}{V},\left(-\frac{x}{\alpha}\right)^V\right) \quad \text{if } x \leq 0 $$

$$ F(x) = 1 - \frac{1}{2} \Delta\left(\frac{1}{V},\left(\frac{x}{\alpha}\right)^V\right) \quad \text{if } x \geq 0 \quad (2.16) $$

By inverting the last cumulative distribution we finally obtain the generated random vari-
ables:

$$ F^{-1}(u) = -\alpha[\Delta^{-1}\left(\frac{1}{V},2u\right)]^{\frac{1}{V}} \quad \text{if } u \leq 1/2 $$

$$ F^{-1}(u) = \alpha[\Delta^{-1}\left(\frac{1}{V},2(1-u)\right)]^{\frac{1}{V}} \quad \text{if } u \geq 1/2 \quad (2.17) $$

where $\Delta^{-1}$ is the inverse regularized incomplete gamma function and $u$ is a uniformly
generated random variable. The simulations in this thesis is made in C++ so a library
called boost, is used for generating values for the different gamma functions. In Fig.
2.5 and Fig. 2.6 it is seen how accurate this novel method is when generating random
variables. The generated curves exactly follow the theoreticals curves. Fig. 2.5 shows the
case where $V=1$ and Fig. 2.6 where $V=2$. 
2.4. SIMULATION OF SYMBOL ERROR RATES

Figure 2.5: The new random variable generator for generating Laplace distribution together with the theoretical curve of the Laplace distribution.

2.4 Simulation of Symbol error rates

Even though the SER can easily be derived in theory, it is still difficult to obtain accurate results when performing simulations. In the simulations we will actually do the same thing as the theory, that is, transmit waves, add noise, and check whether the receiver estimates the transmitted symbol right or wrong. In the simulations everything will be performed discrete in time, which means that every wave will have a specific amount of samples. Noise from the channel will be included by adding to each sample a random number, which is generated from a noise generator:

\[ \text{Received Signal}[i] = \text{Transmitted Signal}[i] + \text{Noise Sample}[i]. \]

The received signal will go through Maximum-Likelihood (ML) filters and will then be estimated. The ML-filters compares the sent symbol with the symbols in the signal constellation, i.e. one ML-filter for each symbol in the constellation. If the receiver estimates the symbol incorrectly, an error will occur. It is common to simulate this with the Monte Carlo method.
CHAPTER 2. THEORETICAL BACKGROUND

The Monte Carlo method

The fundamentals of the Monte Carlo method in SER-simulations, are to transmit some amount of symbols, check how many was estimated incorrectly and then divide it with the amount of transmitted symbols. The Monte Carlo method is repeating experiments many times, and based on these results a mean value can be computed. In SER-simulations the experiments will give an average SER-value made by Monte Carlo simulations. The more experiments the more accurate the result will be.

Sequential estimation

The simulation length, which is depending on the number of transmitted symbols, is often fixed. The problem with this is that when simulating for high SERs many errors will be found, while the simulations at low SER will find few errors. At very low SERs it is possible that the simulations does not even find a single error during the simulation time. If we don’t know what SER value to expect at a specific SNR before simulation, it is not easy to choose a value for total amount of transmitted symbols. The solution for this is to use sequential simulations. Sequential simulations means that the current simulation will continue until a specific amount of errors have been detected, unaware of the SER it currently operates on. The system uses feedback to the transmitting source which tells it when it has reached the satisfied amount of errors and when it can stop transmitting.

Figure 2.6: The new random variable generator for generating Normal distribution together with the theoretical curve of the Normal distribution.
2.5. EFFICIENT SIMULATION

The drawback is that when operating at very low SER, the simulations may go on for an undesirable long time.

Minimum simulation time

When performing Monte Carlo simulations the user must decide for how long time the simulations should go on. What is accurate enough is not easy to decide, but a rule of thumb is to find more than 10 errors [2] at the current SNR. The user must choose what the purpose of the result is. If the purpose is to have an overview over the SER-curve, 10 errors will work fine, but if it is important to have accurate values, which is common for extrapolation of the curve at low SER, then the accuracy must be much higher. Almost every succeeded extrapolation procedure in the result section in this thesis uses at least 10,000 errors. However, with higher accuracy comes longer simulation times. There is a problem here that is worth mentioning. Each symbol in a signal constellation does not always have to have the same SER as the other symbols at a specific SNR. Claiming that finding 10 errors is enough is not exactly true. It is better to find at least 10 errors for each symbol, not a total of 10 errors. Often though, the SER is somewhat the same for all the symbols so it does not really matter. The user know what the signal constellation looks like and can therefore choose how to proceed in the matter.

2.5 Efficient Simulation

When simulating an SER-curve it is not possible to use the Monte Carlo method to compute very small SER, lets say at the level of 1.0e-14. The reason is that it will take too long time to obtain accurate results. To speed up the simulations there are two extrapolation methods that makes things easier for the user. The first one is the Weinstein method, a poor method, and the other one is a novel extrapolation method, which works very well. This thesis will only cover the novel method. This will be discussed in the next chapter.
The novel tail extrapolation method is founded by Nebojsa Stojanovic and this section is based on his article Tail Extrapolation in MLSE Receivers Using Nonparametric Channel Model Estimation [6]. The novel tail extrapolation method assumes that the distribution for the signals are part of the generalized Gaussian distribution family. The derivation of the method focus on extrapolation of unknown values in the PDF but it can be used for SER estimations as well. By taking the logarithm of the generalized Gaussian distribution, see equation (2.6), we obtain the following equation:

\[
    f_L(x) = \log f(x) = \log \frac{V}{2aX(\frac{1}{V})} - (|x - \mu|/a)^V \log e
\]

(3.1)

The equation is a polynomial of order \( V \) and we can use the Newton interpolation formula with equidistant values as arguments

\[
    f_L(x + (V + 1)\Delta x) = f_L(x) + \sum_{i=1}^{V} \binom{V + 1}{i} \theta_i.
\]

(3.2)

That means that for polating a specific value for \( f_L \), it uses \( f_L \)-values that are some predefined \( \Delta x \) units away. \( \Delta x \) is the distance between two \( f_L \)-values next to each other, \( V \) is the exponential constant (shape parameter) for the current distribution and

\[
    \theta_1^i = f_L(x + i\Delta x) - f_L(x + (i - 1)\Delta x), \quad i = 1, 2, 3, ..., V
\]

\[
    \theta_2^i = \theta_1^{i+1} - \theta_1^i, \quad i = 1, 2, 3, ..., V - 1
\]

\[
    \theta_V^i = \theta_1^{V+1} - \theta_1^{V}, \quad i = 1.
\]

(3.3)

The value of \( V \) must be an integer, because it is used in summations as an index. This algorithm will therefore be limited to those cases, even if there are many distributions which uses decimal values as shape parameter. Equations (3.2) and (3.3) gives:

\[
    \sum_{i=0}^{V+1} (-1)^{V+1-i} \binom{V+1}{i} f_L(x + i\Delta x) = 0.
\]

(3.4)

The formula only uses equal distances for \( x \), which make it possible to see the probability density function as a quantized function, or histogram, see Fig. 3.1. Each bin in the
Figure 3.1: Quantized probability density function

histogram has a center value $r_k$ and the area for each bin corresponds to the probability that a random variable between $r_k - \Delta x/2$ and $r_k + \Delta x/2$ will be generated. The probability of

$$r_k - \Delta x/2 \leq x \leq r_k + \Delta x/2$$

(3.5)

is thus given by

$$F(r_k) = \int_{r_k-\Delta x/2}^{r_k+\Delta x/2} f(x)dx$$

(3.6)

If $\Delta x$ is small enough we can approximate (3.6) with the area of an rectangle, i.e.,

$$f(r_k) \approx \frac{1}{\Delta} F(r_k).$$

(3.7)

Taking the logarithm of (3.7) gives:

$$f_L(r_k) \approx -\log \Delta + \log F(r_k) = -\log \Delta + F_L(r_k)$$

(3.8)

Using (3.8) in (3.4) gives:

$$\sum_{i=0}^{V+1} (-1)^{V+1-i} \binom{V+1}{i} \Phi(r_{k+i}) = 0.$$ 

(3.9)

$\Phi(r_k) = F_L(r_k)$ is the variable that will be extrapolated [6]. To use this equation for the extrapolation we replace ”$\approx$” with ”=” in (3.9). The goal with equation (3.9) is to have some simulated $\Phi$ and then estimate one unknown $\Phi$ with the help of the rest. The equation can be transformed into two equations depending on whether we want to extrapolate
the tail on the left side or the right side of the mean value of the pdf. If extrapolating on the left side we obtain:

\[ \Phi(r_k) = \sum_{i=1}^{V+1} (-1)^{1-i} \binom{V+1}{i} \Phi(r_{k+i}), \tag{3.10} \]

while extrapolating on the right side of the mean value it will look like

\[ \Phi(r_k) = \sum_{i=0}^{V} (-1)^{V-i} \binom{V+1}{i} \Phi(r_{k-V-1+i}). \tag{3.11} \]

So far the extrapolation has covered the PDF curve. What is more important in this thesis is to extrapolate BER or SER curves. Define \( F_C \):

\[ F_C(r_k) = 1 - \int_{-\infty}^{r_k-\Delta/2} f(x) dx = \int_{r_k-\Delta/2}^{\infty} f(x) dx = \sum_{i=k}^{\infty} F(r_i) \tag{3.12} \]

where \( \int_{-\infty}^{r_k-\Delta/2} f(x) dx \) is the cumulative distribution function [6]. The cumulative distribution function is closely related to the BER and can therefore be used in estimation of BER. Instead of \( F \) we have used \( F_C \) in the extrapolation formula.

In (3.11) it is seen that for calculating \( \Phi(r_k) \), old \( \Phi \) values which reaches from \( \Phi(r_{k-V-1}) \) to \( \Phi(r_{k-1}) \) is needed. That means that for a specific \( V \), \( V+1 \) bins have to be pre-simulated. These pre-simulated bins will be discussed much in the result section and they will from now on be called reliable bins. The extrapolation is very sensitive to these reliable bins and the closer they are to the theoretical values the better it is. In the result section it will be shown that if using theoretical values as reliable bins, this extrapolation algorithm is extremely accurate.

To use the algorithm the user must know what value for \( V \) to use. Often the noise is something that the user has included and in digital communication, AWGN, is the most common one used. But there are times, perhaps when measuring signals in a practical situation, when the noise, and therefore also \( V \), is unknown. Luckily this algorithm has a convergence pattern that makes the choice for \( V \) easy, at least when having high accuracy for the reliable bins. The algorithm will show the value of \( V \) (what distribution the noise follows) and also how well the extrapolation went. The user shall extrapolate a curve with a specific \( V \)-value and then also extrapolate a new curve with \( V+1 \). If the curve for \( V+1 \) almost exactly follows the curve for \( V \), then it means that \( V \) was the correct shape parameter. The closer the curves are to each other the more accurate the extrapolation will be. To get a measure for the success of the extrapolation we introduce a term called the differential. The differential is the difference between the logarithmic last bin value for \( V \) and the logarithmic last bin value for \( V+1 \).

\[ \text{differential} = |\Phi_V(\text{lastbin}) - \Phi_{V+1}(\text{lastbin})| \tag{3.13} \]

It does not have to be the last bins that are compared, they just need to have the same sample index. A tip to the user is to extrapolate curves for all \( V \) that is possible (it will depend on how many pre-simulated reliable bins there are), and just check the differentials for all of them. The first one that is low should be the right one.
The algorithm can be summarized in a flow chart, see Fig. 3.2, or in words as follows:

1. Decide what maximum value $V$ could have, $V_{\text{max}}$. It is better with a small value, because there have to be $V_{\text{max}}+1$ reliable bins for the algorithm to work. Also choose a value for the error threshold, which means when to be satisfied with the differential. Then start simulating $V_{\text{max}}+1$ amount of reliable bins with the Monte Carlo method.

2. Extrapolate with equation (3.11) for all possible values of $V$ and put each result in separate vectors, i.e. one vector for $V=1$, one for vector for $V=2$ etc. For each vector, check the differential, see equation (3.13), from it’s last bins value and the last bins value for the next vector. If the extrapolation has succeeded this differential should be almost zero and choose the lower $V$-value of the two as the correct $V$.

3. When all the differentials have been calculated and no one is small enough, then the reliable bins were poor. It could either be due to bad bin size $\Delta x$, bad location for the reliable bins or, the most common reason, inaccurate reliable bins. If it is the first reason, simply change the bin size and simulate new reliable bins. If it is the second reason, then simply simulate the reliable bins at other amplitudes. Is it the third reason, then just simulate the reliable bins for a longer time (transmit more symbols to obtain better mean values). It is often not easy to know which one of these that causes the problem, but the result section will cover these cases.
Figure 3.2: An advice how to use the algorithm in form of a chart
Ultra Wideband

Ultra-wideband (UWB) is a technology that focuses on transmitting symbols at a very large bandwidth, a bandwidth of more than 25 percent [1] of the center frequency. Because of the great bandwidth it is possible to transmit very large amounts of symbols in less time, making the throughput, or bit rate, very high. It is a technology that takes care of all the unused space in the frequency domain, which makes the possibilities enormous. As with all technologies there are pros and cons. The drawback with the technology is that the ultra-wideband transmitted signals will interfere with other narrowband signals which have license to transmit in some frequency bands. Companies and technologies which have spent lots of money will not be satisfied with more noise in their frequency band so the solution for the UWB technology is to transmit at a very low power level per hertz. The total power for the UWB signal in the whole band could be very large, but seen in a small frequency band it may be very low. It is the same with noise in a AWGN channel. The total power of the noise in a narrow frequency band is often small, but when transmitting over a large bandwidth it becomes significant. That is why noise in an AWGN channel is referred to as noise spectral density [W/Hz], which is flat throughout the whole bandwidth. Regulations, or rules, for transmitting signals in UWB, have been set up by the Federal Communication (FCC) organization, an US authority. The emitting radiators have been limited to transmit at a power spectral density of -41.3 dBm/MHz (dBm is a unit for mW in Decibels) in the frequency spectrum, which is between 3.1 GHz and 10.6 GHz. In more crucial frequency regions the limit can be much lower, for example where the Global Positioning System (GPS) operates, between 1.2 GHz and 1.5 GHz. Because of transmit power limitations, this technique is used particularly for short range communications.

It is possible to make simulations that calculates the throughput vs distance, which gives better understanding of how useful UWB is.

4.1 Throughput for ultra-wideband

Throughput is another word for symbol rate, the number of successful delivered symbols within some time limit.

\[
\text{throughput} = \frac{N}{t}
\]

(4.1)
where:

\( N \) = number of symbols sent
\( t \) = transmitting time for the symbols

The power restriction for the UWB technology makes the signals sensitive to noise. Often systems have demands that the SER at the receiver should be low, which in turn forces the transmitter to send at a specific SNR. Besides that, there are more issues like path loss and physical noises in the environment that makes the signal power decrease with increasing distance. Path loss makes the signal power decrease and may be due to diffraction, reflection or other physical reasons like terrain etc. Mathematically, path loss is used in Decibels and is therefore subtracted from the transmitted signal.

\[
L = 10 \cdot n \cdot \log_{10}(d) + C \tag{4.2}
\]

where:

\( L \) = path loss
\( n \) = environment constant (2=free space, 4-6=rough indoor environment)
\( d \) = distance
\( C \) = system losses constant (which depends on the frequency used)

One more factor that also decreases the transmitted signal power is the link margin. When allowed to transmit at whatever power is needed, the link margin if often seen as a safety factor for the transmitted signal power. No system is ideal and that is why it is necessary to transmit at higher power. In the case of UWB, signals are always transmitted at the maximum allowed power, that is, according to the regulations, -41.3dBm/MHz. In this cases the link margin is seen as a value that is subtracted from the transmitted signal power due to power loss. Both path loss and link margin is often given in dB and both affects the signals, thus they can be used in the same equation. There may also be an antenna gain in both the transmitting and the receiving antennas and these will amplify the signals (and the noise, but noise is not used so far in the formula). The antenna gain is also a scaling factor and will be an additative value when working with Decibels. The received power can then be obtained as

\[
P_r = P_t + G_t + G_r - L - LM \tag{4.3}
\]

where:

\( P_r \) = received signal power
\( P_t \) = transmitted signal power
\( G_t \) = gain from transmitting antenna
\( G_r \) = gain from receiving antenna
\( L \) = path loss
\( LM \) = link margin

Now when the power loss equation is derived, the throughput, see equation (4.1), can be written as follows:

\[
B_p = \frac{P_{dd} \cdot B_s}{N_0 \cdot E_s N_0} \tag{4.4}
\]
where:

\[ B_p = \text{throughput (symbol rate)} \]
\[ P_{sd} = \text{average power spectral density} \]
\[ B_s = \text{bandwidth of the transmitted pulse} \]
\[ N_0 = \text{noise spectral density}. \]

Equation (4.4) uses \( P_{sd} \), which is the maximum allowed power per hertz, so it actually calculates the pulse rate if no path loss or link margin would be used. Path loss and link margin should be used though, so it is necessary to transform the power spectral density into just power. That is possible with the formula

\[ \text{Power} = \text{Power Spectral density} \cdot \text{Bandwidth}. \]

Using this relation into equation (4.4) gives the following equation

\[ B_p = \frac{P_r}{N_0 \cdot E_s N_0} \] (4.5)

By using a larger bandwidth the total power will increase, which in turn makes the symbol rate increase. This formula assumes that there is no dead space in the time domain between each pulse, so directly after one pulse is transmitted the next will start. There is still one variable in equation (4.4), that has not been decided yet and that is the noise spectral density (noise floor), \( N_0 \), after the receiver. It is common to use a standard noise spectral density in the channel in digital communications and electronics, also called thermal- or Johnson-Nyquist noise. Thermal noise is white, flat noise through the whole frequency spectrum, and this thesis focus on AWGN channels. The noise spectral density for thermal noise is:

\[ N_0 = k_B \cdot T. \] (4.6)

where:

\[ k_B = 1.3806503 \cdot 10^{-23}[m^2 \cdot kg \cdot s^{-2} \cdot K^{-1}] \] Boltzmann’s constant
\[ T = \text{temperature in Kelvin (often 290 for room temperature)} \]

When using room temperature the noise spectral density has a very nice value of 4.0e-21 W/Hz.

The standard noise spectral density is a noise floor that exists in the channel, before the receiver, but inside the receiver it is common that the noise increase due to the electronics. The ratio for the SNR before the receiver and the SNR after the receiver is called noise figure and is often a documented parameter for different electronic equipment. Low noise figure means that the equipment is good and the ideal case is when the noise figure is 0dB. Noise factor is the same as noise figure but in inverse Decibel. The equation for noise figure is as follows:

\[ F = \frac{SNR_i}{SNR_o} = \frac{S_i/N_i}{G \cdot S_i/[G \cdot N_i + N_d]} = \frac{G \cdot N_i + N_d}{G \cdot N_i} \] (4.7)
CHAPTER 4. ULTRA WIDEBAND

where:

\begin{align*}
F &= \text{noise factor (inverse dB of noise figure, NF)} \\
SNR_i &= \text{signal to noise ratio at the input of the receiver} \\
SNR_o &= \text{signal to noise ratio at the output of the receiver} \\
S_i &= \text{signal power at the input} \\
N_i &= \text{noise power at the input} \\
N_d &= \text{internal noise power in the device} \\
G &= \text{gain inside the receiver}
\end{align*}

The \( N_0 \), that will be used in equation (4.4), is the input noise spectral density plus the internal noise spectral density in the device. Recall that noise spectral density is the same as noise power per hertz.

\[ N_0 = N_i + N_d. \]

In the result section there will be an example of throughput vs distance using SER that has been estimated using the novel extrapolation method.
5

RESULT

5.1 Novel Tail Extrapolation method

We assume that the noise statistics follow the AWGN model with shape parameter \( V=2 \) when comparing with theoretical values. In this section results of SER vs SNR will be presented by means of figures and tables.

Results using theoretical values as reliable bins

Using theoretical values in conjunction with popular modulation types enable us to evaluate the accuracy of the algorithm since popular modulation types have formulas for symbol error rates. As said previously when discussing the theory behind the novel extrapolation method, the goal is to get as small a difference as possible between the last bins for the correct \( V \) and \( V+1 \). The reason for that is because of convergence. The smaller the difference, the faster the convergence. In Fig. 5.1, we observe how good the algorithm works with theoretical values as reliable bins. The curves for \( V=2 \) and \( V=3 \) are really close to each other which makes the differential between their last bins small. Recall that the differential is defined as the difference between the logarithmic last bin value for \( V \) and logarithmic last bin value for \( V+1 \), see equation (3.13). In the simulation for generating the figure, extrapolation for \( V=4 \) was also made. That is not inserted in the figure but is used for calculating the differential between the last bin for \( V=3 \) and the last bin for \( V=4 \), to see if \( V=3 \) is the correct shape parameter. The simulation result for Fig. 5.1 is displayed in Table 5.1. The most important column is the differential column (that shows how trustworthy the result is). Note that the last bin for \( V=2 \) is very close to the corresponding theoretical value.

From the simulation with theoretical values as reliable bins it is noticeable that a differential with a value of 0.49 for \( V=2 \), see Table 6.1, gives accurate extrapolation result. Getting SER-values as small as this cannot be made by the traditional Monte Carlo method with reasonable amount of computations.

The differential for the case above was calculated like this:

\[
\text{differential} = |\log(7.40e-13) - \log(2.30e-12)| = 0.492951.
\]

If the differential would be less than 1.0 in the extrapolation algorithm it would defi-
CHAPTER 5. RESULT

Table 5.1: Differentials, used for deciding which Gaussian distribution that is the correct one. The noise used in the simulation is Gaussian noise with $V=2$.

<table>
<thead>
<tr>
<th>$V$</th>
<th>Differential</th>
<th>Extrapolated last bin</th>
<th>Target for last bin (theoretical value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.65061</td>
<td>3.31e-08</td>
<td>7.11e-13</td>
</tr>
<tr>
<td>2</td>
<td>0.492951</td>
<td>7.40e-13</td>
<td>7.11e-13</td>
</tr>
<tr>
<td>3</td>
<td>0.920707</td>
<td>2.30e-12</td>
<td>7.11e-13</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>2.76e-13</td>
<td>7.11e-13</td>
</tr>
</tbody>
</table>

Figure 5.1: Extrapolation with $V=1$, $V=2$, and $V=3$ using theoretical values as reliable bins. The reliable bin values are taken from the SER-formula for 16-QAM in an AWGN channel, which means that $V=2$ should be the correct shape parameter.

Table 5.2 shows that after finding a small differential for a curve, there will be convergence for all next coming curves as well. In that table theoretical values have been used for reliable bins. A noise with $V=2$ is used and all the next coming curves has almost the same last bin value. Fig. 5.2 shows a simulation result with theoretical reliable bins, using 16-QAM. This plot shows that the extrapolation method works really well as long as the reliable bins are very accurate. The simulated curve is plotted side by side with its corresponding theoretical curve.
5.1. NOVEL TAIL EXTRAPOLATION METHOD

Table 5.2: Differentials and last bin values for each V-value, using theoretical values as reliable bins. The modulation used is 2-PSK, i.e. binary phase shift keying.

<table>
<thead>
<tr>
<th>V</th>
<th>Differential</th>
<th>Extrapolated last bin</th>
<th>Target for last bin (theoretical value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.05</td>
<td>7.79e-09</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>2</td>
<td>0.59</td>
<td>6.94e-14</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>3</td>
<td>0.64</td>
<td>1.80e-14</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>4</td>
<td>0.64</td>
<td>7.79e-14</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>5</td>
<td>0.44</td>
<td>1.79e-14</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>6</td>
<td>0.24</td>
<td>4.89e-14</td>
<td>3.63e-14</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>8.50e-14</td>
<td>3.63e-14</td>
</tr>
</tbody>
</table>

Figure 5.2: Simulated curve vs theoretical curve, using 16-QAM and noise V=2. The simulated curve has used theoretical reliable bins.

Results using known noise

After many simulations and trials with different noises, it is found that the algorithm works best if the shape parameter for the noise is given. Actually that is also often the case when simulating, the user chooses to add whatever noise is needed. When knowing that, it is only necessary to use the shape parameter plus 1 number of reliable bins, which will reduce the simulation time drastically. One other reason is that the algorithm does not have to check for the smallest differential and choose the shape parameter that way.
because it only needs to focus on one differential, the one for V=2.

**Results using unknown noise**

The novel extrapolation algorithm can be used with any kind of noise as longs as it belongs to the generalized Gaussian family. In Section 2.3, we described an algorithm how to generate noise with different noise parameters. By using that algorithm in a noise generator it is possible to validate the novel extrapolation method for V-values other than 2. The first challenge is to choose how many reliable bins to simulate. What is known is that \( V_{\text{max}} + 1 \) amount of reliable bins is needed, where \( V_{\text{max}} \) is the highest possible V-value to extrapolate with. A typical number may be \( V_{\text{max}} = 5 \) if nothing is known about the noise. When observing the SER curves for different shape parameters there are similarities for the curves at high SER values. In Fig. 5.3 it can be seen that for \( V > 1 \) the curves are close to each other for small SER when using a sequential estimation of 50 errors. Monte Carlo simulations was used, so only SER in areas around 1.0e-5 could be simulated. For \( V = 1 \), i.e. Laplacian noise, the SER-curve is significantly above the rest. When using noise with \( V > 1 \) the algorithm can mix up the extrapolation because of the similarities of the values for the reliable bins, so high accuracy here is important.

The algorithm does not work well when using other noise parameter values than \( V = 2 \). It is very sensitive and, when using \( V = 3 \), a sequential estimation of 1,000,000 errors must be used to get a reasonable accurate result. Either way, there are no theoretical curves to check the result against, as it does for SER in AWGN channels. The advice is thus to always use \( V = 2 \), because there will be trustful results and the AWGN channel is the most used channel anyway. Table 5.3, shows an extrapolation simulation using a noise with \( V = 3 \) and a sequential estimation using 100,000 errors. Theoretical values cannot be presented because there are no such values for \( V = 3 \) to be found. In Table 5.4 we display the results for \( V = 3 \) and 1,000,000 errors. The target for the last bin (theoretical value) cannot be presented because there are no such values for \( V = 3 \) to be found. The simulation time is 5h 56min 15sec and the reason that it took so much time to simulate in comparison with \( V = 2 \) is because this time the noise generator from Section 2.3 is used and that has a more complex algorithm than usual Gaussian noise generators.

### Table 5.3: Differentials and last bin values. BPSK modulation with a sequential estimation of 100,000 errors and shape parameter \( V = 3 \).

<table>
<thead>
<tr>
<th>( V )</th>
<th>Differential</th>
<th>Extrapolated last bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.38</td>
<td>7.69e-07</td>
</tr>
<tr>
<td>2</td>
<td>3.49</td>
<td>3.20e-13</td>
</tr>
<tr>
<td>3</td>
<td>40.5</td>
<td>1.04e-16</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>2.96e-57</td>
</tr>
</tbody>
</table>

**Results and comparisons using different amount of errors in sequential estimation**

The higher the differential, the less sure the user can be that the extrapolation is trustworthy. To be sure that the extrapolation is good, the user should re-simulate until the differential ends up at some acceptable value, typically around 1. Now it is time to test
5.1. NOVEL TAIL EXTRAPOLATION METHOD

Figure 5.3: SER for different values of $V$, using Monte Carlo simulations and a sequential estimation of 50 errors.

Table 5.4: Differentials and last bin values for a sequential estimation of 1,000,000 errors and shape parameter $V=3$. The modulation used is 16-QAM and simulation time is 5h 56min 15sec.

<table>
<thead>
<tr>
<th>$V$</th>
<th>Differential</th>
<th>Extrapolated last bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.50</td>
<td>2.50e-06</td>
</tr>
<tr>
<td>2</td>
<td>0.26</td>
<td>7.85e-12</td>
</tr>
<tr>
<td>3</td>
<td>0.54</td>
<td>4.30e-12</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>1.48e-11</td>
</tr>
</tbody>
</table>

As remarked in section 2.4, a rule of thumb for obtaining an acceptable SER value at a specific SNR is to detect at least 10 errors with Monte Carlo simulations. That would have worked great if the whole curve is simulated with MC, but when using the extrapolation algorithm, more accuracy is required because of its sensitivity. Table 5.5 shows the extrapolation results with different values for the variable $errorTot$. $errorTot$ means how many errors the Monte Carlo simulations must find for each SNR, see section 2.4, before stopping. The higher value $errorTot$ has, the more accurate the simulations will be. Three experiments have been performed in each case and it is shown that a low $errorTot$ value is unacceptable. For those poor cases the results will differ enormous from one time to another. The reason that Table 5.5 used three experiments for each level of sequential...
estimation, is just to show the user how one experiment output can differ from another when using low value for errorTot. The differentials gets lower and the extrapolated last

<table>
<thead>
<tr>
<th>errorTot</th>
<th>Differential</th>
<th>Extrapolated last bin</th>
<th>Target for last bin (th value)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10, try: 1</td>
<td>101.8</td>
<td>35.80</td>
<td>1.58e-13</td>
<td>0.035 sec</td>
</tr>
<tr>
<td>10, try: 2</td>
<td>782</td>
<td>1.36e-90</td>
<td>1.58e-13</td>
<td>0.15 sec</td>
</tr>
<tr>
<td>10, try: 3</td>
<td>360</td>
<td>7.15e-51</td>
<td>1.58e-13</td>
<td>0.13 sec</td>
</tr>
<tr>
<td>100, try: 1</td>
<td>116</td>
<td>1.90e-24</td>
<td>1.58e-13</td>
<td>0.19 sec</td>
</tr>
<tr>
<td>100, try: 2</td>
<td>21.8</td>
<td>3.29e-22</td>
<td>1.58e-13</td>
<td>0.27 sec</td>
</tr>
<tr>
<td>100, try: 3</td>
<td>42.1</td>
<td>7.61e-23</td>
<td>1.58e-13</td>
<td>0.19 sec</td>
</tr>
<tr>
<td>1000, try: 1</td>
<td>4.35</td>
<td>1.05e-13</td>
<td>1.58e-13</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>1000, try: 2</td>
<td>31.8</td>
<td>6.81e-11</td>
<td>1.58e-13</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>1000, try: 3</td>
<td>16.0</td>
<td>7.49e-11</td>
<td>1.58e-13</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>10,000, try: 1</td>
<td>1.59</td>
<td>2.26e-14</td>
<td>1.58e-13</td>
<td>13.3 sec</td>
</tr>
<tr>
<td>10,000, try: 2</td>
<td>0.32</td>
<td>9.32e-14</td>
<td>1.58e-13</td>
<td>13.1 sec</td>
</tr>
<tr>
<td>10,000, try: 3</td>
<td>5.68</td>
<td>2.00e-14</td>
<td>1.58e-13</td>
<td>13.1</td>
</tr>
<tr>
<td>100,000, try: 1</td>
<td>4.32</td>
<td>2.27e-14</td>
<td>1.58e-13</td>
<td>2min 11sec</td>
</tr>
<tr>
<td>100,000, try: 2</td>
<td>1.79</td>
<td>8.73e-14</td>
<td>1.58e-13</td>
<td>2min 12sec</td>
</tr>
<tr>
<td>100,000, try: 3</td>
<td>1.90</td>
<td>3.14e-14</td>
<td>1.58e-13</td>
<td>2min 12sec</td>
</tr>
<tr>
<td>500,000, try: 1</td>
<td>0.70</td>
<td>3.94e-14</td>
<td>1.58e-13</td>
<td>10min 57sec</td>
</tr>
<tr>
<td>500,000, try: 2</td>
<td>1.29</td>
<td>2.98e-14</td>
<td>1.58e-13</td>
<td>10min 59sec</td>
</tr>
<tr>
<td>500,000, try: 3</td>
<td>0.89</td>
<td>3.51e-14</td>
<td>1.58e-13</td>
<td>10min 59sec</td>
</tr>
</tbody>
</table>

The extrapolation almost always look good (at least when using the current SNR for the reliable bins), even though the differentials sometimes is quite big. However, it is necessary to re-simulate in those cases. The differential must be low to be sure of an accurate result. Table 5.5 also shows that for errorTot=500,000, the differential ended up less than 1.5 for each experiment. For these cases the output was definitely good.

Results using randomly reliable bins

When the extrapolation method does not yield good results, the user have to re-simulate again. One way to do it is to use the same signal to noise ratio for the reliable bins and hope that the accuracy will be better the next time. Another approach is to change the signal to noise ratio, and the bin size, for the reliable bins each time the extrapolation has failed and hope for a different output. In the computer program written for this thesis there is a member function, StartMultiSim_Using_Randomness(int maxIter, double threshold), in the SER class, that allows for randomized SNRs in some predefined range, a change in bin size, different amount of bins and where the bins are positioned. The reliable bins will then be simulated at these new randomized positions, i.e. operating at random SNR-levels. With this approach the extrapolation should succeed because of randomness. This
may shorten the calculation and simulation time and it is also not necessary to know that much about the system (know at what SNR to simulate the reliable bins). A preview is shown in Table 5.6. Here, three simulations are made (user choice that can be set in the program) with sequential simulation of 10,000 errors and the output will be the best one out of these three, see Fig. 5.4. The column Reliable bins in Table 5.6 shows what SER-values that will be used as reliable bins in the extrapolation method. In Table 5.5, it can be seen that the result for 10,000 errors is often good, but not for all simulations, so we give it three tries at different SNR operating levels. The goal is to reach to a differential under 1 or at least somewhere in that area and the third try went well, giving a differential of 0.53. That is a result to be happy with and to trust and the simulation only took 65 seconds. When using a sequential estimation of a large value, say 10,000, then it is not necessary to do that many re-simulations, iterations. We will show the same strategy with operating at random SER-values for the reliable bins but with much less accuracy, a sequential estimation of 1,000 errors. Here it is necessary to iterate more times, because the accuracy will be poorer. Fifteen iterations will be used and the result can be found in Table 5.7. The second experiment of the fifteen went really good, giving a differential of 0.14. The eleventh try, which gave 1.63 as differential, may also be seen as a good result. The output plot for the second try can be seen in Fig. 5.5. The conclusion here is that even an approach that should give poor accuracy may give good result by the use of many experiment. Also, the simulation time here only took 40 seconds.

**Table 5.6:** Random generated SER operating levels in three iterations using a sequential estimation of 10,000 errors. The bin size, the start SNR and the end SNR for the reliable bins are random for the different cases.

<table>
<thead>
<tr>
<th>Try</th>
<th>Differential</th>
<th>Reliable bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.58</td>
<td>0.87 0.66 0.43 0.23</td>
</tr>
<tr>
<td>2</td>
<td>10.02</td>
<td>0.64 0.43 0.26 0.13</td>
</tr>
<tr>
<td>3</td>
<td>0.53</td>
<td>0.63 0.39 0.20 0.082</td>
</tr>
</tbody>
</table>

**Time Efficiency using the novel extrapolation method**

In this section, an example for the time efficiency for the novel extrapolation method will be presented versus the old fashioned MC-method. MC has been widely used and has been the only fully trusted way for generating symbol error rates. When using MC there does not have to be much knowledge about the system, which makes it the ultimate method. The user could just start the MC simulation and let it do the dirty work. The drawback is, of course, the computation time when reaching low SER. For example, if the current theoretical SER is at 1.0e-9.0, then the MC method must transmit one billion symbols just to find one error and to get a good mean value it must at least find ten errors. Ten billion symbols must be sent and even then the accuracy is quite poor. The program that is written for this thesis uses MC for high SER and extrapolates the low SER. The drawback for the novel extrapolation method is the demand for accuracy, see Section 5.1, where it may have to find 10,000 errors, while the traditional MC only needs to find about 10 for each SER. The traditional MC method wins if only SER above 1.0e-04 is required, but below that the novel extrapolation is better. Even the best computers today will never be able to produce SER values down to 1.0e-15 using MC. Table 5.8 shows the result
CHAPTER 5. RESULT

Figure 5.4: Output when using random generated reliable bins. Three iterations was made with a sequential estimation of 10,000 errors. The last try gave the best result with a differential of 0.53. It took 65 sec to iterate three times and get this result.

of pure MC simulations vs the novel extrapolation method using modulation 16QAM. The MC simulations uses a sequential estimation of 10 errors, while the novel method uses a sequential estimation of 10,000 errors for its reliable bins. The most interesting columns for comparing the efficiency are the total time and transmitted symbols column. It is seen that the extrapolation method reaches a perfect extrapolation down to 1.0e-13 in 15.6 seconds, while the traditional MC reaches about 1.0e-04 in the same time. Also, the MC simulation was stopped at a SER of 3.22e-08 because it would have taken too much time to simulate one more bin. The next bin would have taken about 130 hours to simulate, because the SER is proportional to simulation time (transmitted symbols).

5.2 Ultra Wideband simulation results

The main variable for the throughput analysis (symbol rate), is the signal to noise ratio, EsN0. The rest of the variables are only constants for the current system. Noise figure and link margin are constants for the current environment, gain is constants for the antenna etc. EsN0 though, is something that can vary depending on what signal power that is needed. That signal power can be decided by choosing an acceptable limit for symbol error rate. Perhaps it does not matter if a specific receiver misjudge one signal out of one
5.2. ULTRA WIDEBAND SIMULATION RESULTS

Table 5.7: Random generated operating areas in fifteen iterations using a sequential estimation of 1,000 errors. The second try gave the best result.

<table>
<thead>
<tr>
<th>Try</th>
<th>Differential</th>
<th>Reliable bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.39</td>
<td>0.86 0.67 0.43 0.23</td>
</tr>
<tr>
<td>2</td>
<td>0.14</td>
<td>0.67 0.40 0.19 0.070</td>
</tr>
<tr>
<td>3</td>
<td>26.4</td>
<td>0.47 0.32 0.18 0.093</td>
</tr>
<tr>
<td>4</td>
<td>5.79</td>
<td>0.81 0.55 0.31 0.14</td>
</tr>
<tr>
<td>5</td>
<td>43.6</td>
<td>0.81 0.62 0.38 0.23</td>
</tr>
<tr>
<td>6</td>
<td>29.2</td>
<td>0.54 0.38 0.23 0.13</td>
</tr>
<tr>
<td>7</td>
<td>130.3</td>
<td>0.32 0.22 0.13 0.084</td>
</tr>
<tr>
<td>8</td>
<td>6.3</td>
<td>0.78 0.58 0.36 0.19</td>
</tr>
<tr>
<td>9</td>
<td>2.66</td>
<td>0.64 0.38 0.18 0.068</td>
</tr>
<tr>
<td>10</td>
<td>12.1</td>
<td>0.42 0.27 0.15 0.072</td>
</tr>
<tr>
<td>11</td>
<td>1.63</td>
<td>0.38 0.25 0.16 0.093</td>
</tr>
<tr>
<td>12</td>
<td>3.70</td>
<td>0.77 0.61 0.45 0.30</td>
</tr>
<tr>
<td>13</td>
<td>48.1</td>
<td>0.31 0.19 0.12 0.070</td>
</tr>
<tr>
<td>14</td>
<td>21.4</td>
<td>0.78 0.58 0.37 0.21</td>
</tr>
<tr>
<td>15</td>
<td>6.48</td>
<td>0.82 0.51 0.25 0.092</td>
</tr>
</tbody>
</table>

Million signals. The great thing here is that this thesis covers extrapolation of SER curves and the low extrapolated SER can now be used in the throughput analysis for ultra wideband. The extrapolation method estimates the whole SER curve and then lets the UWB throughput analysis calculate what symbol rates that can be used for the different low SER. The output will be a symbol rate vs distance curve, which is presented in Example 1.

**Example 1**

The following information is used:

The Symbol error rate part:
- 16-QAM signals
- Sequential estimation of 10,000 errors
- Using V=2 noise (Gaussian)

The UWB part:
- Link margin = 4dB
- Noise Figure = 7dB
- Bandwidth = 2.5GHz
- Carrier Frequency = 3.75GHz
- Antennas gain = 0dB
- Maximum allowed transmitting power spectral density = -41.3 [W/Hz]

To be displayed:
- The SER-curve in one figure
- Four throughput curves in one figure with SER < 1.0e-06
CHAPTER 5. RESULT

Figure 5.5: Output when using randomly generated reliable bins. Fifteen iterations was made with a sequential estimation of 1,000 errors and simulation time took 40s.

Example 1 result:

Four reliable bins were simulated and used in the extrapolation. These reliable bins are 0.81, 0.51, 0.24 and 0.094. The extrapolation gave a differential of 0.45, the elapsed time was 27 seconds and the plot for the SER curve can be seen in Fig. 5.6. Four of the next coming SER points after 1.0e-06 are used for throughput analysis and can be seen in Fig. 5.7. The figure shows that when operating at 8MSymbol/s you can almost double the transmitting distance if using the SER for Curve 1 compared with Curve 4. You may also double the symbol rate at 15 meter if using SER for Curve 1 compared with Curve 3. The curves in the figure follow the following SER:

Curve 1 -> SER = 7.23e-08
Curve 2 -> SER = 2.17e-09
Curve 3 -> SER = 4.62e-11
Curve 4 -> SER = 7.01e-13
Table 5.8: Comparison between pure MC simulations and the novel extrapolation method. For each of the two cases the simulated SER, the total time since simulation start and the amount of transmitted symbols at the current level are presented. Pure MC simulations uses a sequential estimation of 10 errors and the extrapolating method uses a sequential estimation of 10,000 for the four reliable bins.

<table>
<thead>
<tr>
<th>MC-SER</th>
<th>MC-timeTOT</th>
<th>MC-symb</th>
<th>Extrap-SER</th>
<th>Extrap-timeTOT</th>
<th>Extrap-symb</th>
<th>th. SER</th>
<th>EsN0[dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.833</td>
<td>0.011s</td>
<td>12</td>
<td>0.873</td>
<td>1.6s</td>
<td>11,500</td>
<td>0.873</td>
<td>-8</td>
</tr>
<tr>
<td>0.667</td>
<td>0.027s</td>
<td>15</td>
<td>0.632</td>
<td>3.8s</td>
<td>15,800</td>
<td>0.634</td>
<td>3.0</td>
</tr>
<tr>
<td>0.667</td>
<td>0.046s</td>
<td>15</td>
<td>0.367</td>
<td>7.6s</td>
<td>27,300</td>
<td>0.372</td>
<td>7.7</td>
</tr>
<tr>
<td>0.182</td>
<td>0.083s</td>
<td>55</td>
<td>0.174</td>
<td>15.6s</td>
<td>57,400</td>
<td>0.176</td>
<td>10.8</td>
</tr>
<tr>
<td>0.0427</td>
<td>0.13s</td>
<td>234</td>
<td>0.0677</td>
<td>15.6s</td>
<td>-</td>
<td>0.0675</td>
<td>13.0</td>
</tr>
<tr>
<td>0.0272</td>
<td>0.183s</td>
<td>367</td>
<td>0.0215</td>
<td>15.6s</td>
<td>-</td>
<td>0.0212</td>
<td>14.8</td>
</tr>
<tr>
<td>0.00412</td>
<td>0.51s</td>
<td>2,425</td>
<td>0.00558</td>
<td>15.6s</td>
<td>-</td>
<td>0.00546</td>
<td>16.3</td>
</tr>
<tr>
<td>0.00126</td>
<td>1.57</td>
<td>7,924</td>
<td>0.00118</td>
<td>15.6s</td>
<td>-</td>
<td>0.00116</td>
<td>17.5</td>
</tr>
<tr>
<td>9.75e-05</td>
<td>15.2s</td>
<td>1,03e05</td>
<td>0.000206</td>
<td>15.6s</td>
<td>-</td>
<td>0.000202</td>
<td>18.6</td>
</tr>
<tr>
<td>4.81e-05</td>
<td>42.7s</td>
<td>2,08e05</td>
<td>2.92e-05</td>
<td>15.6s</td>
<td>-</td>
<td>2.90e-05</td>
<td>19.6</td>
</tr>
<tr>
<td>2.88e-06</td>
<td>8min 21s</td>
<td>3.47e06</td>
<td>3.38e-06</td>
<td>15.6s</td>
<td>-</td>
<td>3.42e-06</td>
<td>20.5</td>
</tr>
<tr>
<td>2.76e-07</td>
<td>1h 30min</td>
<td>3.63e07</td>
<td>3.21e-07</td>
<td>15.6s</td>
<td>-</td>
<td>3.29e-07</td>
<td>21.3</td>
</tr>
<tr>
<td>3.22e-08</td>
<td>13h 11min 9s</td>
<td>3.10e08</td>
<td>2.49e-08</td>
<td>15.6s</td>
<td>-</td>
<td>2.60e-08</td>
<td>22.0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.58e-09</td>
<td>15.6s</td>
<td>-</td>
<td>1.68e-09</td>
<td>22.7</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.18e-11</td>
<td>15.6s</td>
<td>-</td>
<td>8.83e-11</td>
<td>23.3</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3.47e-12</td>
<td>15.6s</td>
<td>-</td>
<td>3.80e-12</td>
<td>23.9</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.20e-13</td>
<td>15.6s</td>
<td>-</td>
<td>1.33e-13</td>
<td>24.4</td>
</tr>
</tbody>
</table>
CHAPTER 5. RESULT

Figure 5.6: The SER curve for the example. 15 points are included and the assignment chose the four next coming points after 1.0e-06 to be used in a throughput analysis. Those SER are: 7.23e-08 2.17e-09 4.62e-11 7.01e-13.
5.2. ULTRA WIDEBAND SIMULATION RESULTS

Figure 5.7: Throughput in symbols/s as a function of distance for four different SER-values.
CONCLUSION

The performance of the extrapolation method

The formulas for SER has two unknown variables, SER and EsN0 (signal to noise ratio) and EsN0 has always been known in these simulations. The performance of the extrapolation method can easily be checked by plotting the theoretical value side by side with the simulated values and the closer they are to each other the better it is. To see if the performance of the extrapolation method is good, reliable bins with theoretical values can be used and Fig. 5.2 shows that the extrapolation method is extremely accurate. In real life the reliable bins will not be theoretical but now it is at least known that if using very accurate reliable bins, close to theoretical ones, the result should be satisfying.

When to rely on the results

The conclusion is to rely on the result when the differential reaches below 1.0 and an sequential estimation of at least 1000 errors is used. To be sure that the result is reliable and trustworthy, the same differential target should be used, but with a sequential estimation of 10,000 errors.

Best procedure to get good extrapolation results

The best way to get good extrapolation results is to use the algorithm over and over again (let it iterate) and be satisfied with the result when the differential reaches under 1.0. After each failed iteration the signal to noise ratio for the reliable bins should be randomized and new reliable bins should be simulated. Also, the SNR interval, where the random SNR are taken from, can have a very low start value, lets say at a SER-value of 0.8. The reason for that is because the calculation time for SER at that level is short. Try to have reliable bins in the area 1.0 > SER > 0.1, due to less computation time. Do not use lower SER-values.

The poor result with other noises than Gaussian (V=2) noise

It is unfortunate that the extrapolation algorithm does not work effectively for noises other than V=2. It is not worthwhile simulating with a sequential estimation of 10,000,000.
errors and so on, just to get a somewhat accurate curve. The thoughts about why accuracy is more important with increasing V-value is because of the sums in the extrapolation equation. The higher the value of V is, the more old simulated SER values is used to calculate the nextcoming SER. The need for more information for high V makes the simulation more sensitive. When using V=2 it uses three pre-simulated values for the calculation of the nextcoming SER. When using V=5 it needs six pre-simulated values and so fort.

**Using many symbols**

Using many different symbols will make the computation heavier, because of the need for more Maximum-Likelihood (ML) filters. If we are using 500 symbols, there will be a usage of 500 ML-filters, see Section 2.4, which means that each transmitted symbol must go through each of them so the receiver can choose which symbol that was sent. In these cases it is necessary to simulate with a sequential estimation of not too many errors.
When working on a thesis assignment like this, you notice that there is never enough time to do as much as you want to do. When working with a project, the project itself can always be made bigger and be updated now and then. I only had the time to test the program with famous modulation types like QAM and PSK, but it would have been great to test it with non-modulated symbols and with no theoretical curves to compare it with, using Gaussian noise.

More features may be implemented in the program that can handle more cases than the ones I cover, like generating UWB signals using special filters in the transmitter etc. I have written functions that can read symbols from files though, which in the long run will be the most common usage of the program. It would have been easier to fill the program with more important functions and classes if I would have more knowledge about the market and what is really needed, but more work experience will cover that. That is why I hope my source code is easy to study so it gets easy to implement more features in it for someone who wants to continue with this work.
8

MANUAL FOR C++ CLASSES

8.1 Manual introduction

The library is for making symbol error rates simulations.

There are four classes:
1. SER
2. Signal
3. ExtraPol
4. UWB

The classes work together and the following can be said about it:
SER declares some reference signals of the class Signal. A symbol error rate curve will
be generated with the Monte Carlo method. When the symbol error rate curve reaches
low values it will be time efficient to extrapolate the rest of the curve. That is done by the
class ExtraPol. When the symbol error rate curve is completed, a throughput analysis in
ultra wideband can be obtained for some of the generated symbol error rate values. That
is done by the class UWB.

8.2 How to work with the classes

The SER class has member variables of the classes ExtraPol and Signal. That means that
it is enough to make an instance of the class SER, because the rest will come along. If
reading signals from files though, it is easier to declare the signals yourself and then call
for the function SER::SetRefSignals. But when that is done you should only play around
with the SER class and its member functions to make symbol error rates simulations, even
when using extrapolation. There are some parameters that must be set for a simulation to
be done. All those parameters have appropriate default values (except reference signals),
so the simulations will always work. For simulations in digital communications there are
some parameters that is very crucial to be correctly set.

These crucial parameters in SER are:
• Reference signals  default = ...
• Bin size  default = 3.0
• Start EsN0  default = -8dB

If the bin size is too big there may be too few samples in the symbol error rate curve, making the resolution poor. There are help functions for producing these reliable bins.

Here are the other member variables that SER uses:

• noiseParam  default = 2 (Gaussian noise)
• numBins  default = 100
• minErrors  default = 10,000 (sequential estimation)
• numRelBins  default = 4
• printouts  default = true (printouts during simulations)
• useExtraPol  default = true (automatic extrapolation)
• minSER  default = 1.0e-10 (automatic stop sim at minSER)

There are more member variables, but these are the only ones that are important for the user. numBins has a high value, but it does not matter because it is normal to stop simulations when the symbol error rate has reached minSER, rather than when simulated some numBins amount of values. Often the simulations stop after 10 to 15 bins, even when numBins has the value 100. What is recommended for the user is to try to get good values for starting EsN0 and bin size. When not knowing anything about the system, there is a help function that calculates what starting EsN0 and bin size to use. The member function \texttt{SER::Search\_Good\_Bins\_For\_Extrapolation(double PsHigh, double PsLow)} calculates a good starting EsN0 and bin size so the reliable bins have values between PsHigh and PsLow. When not using any arguments, PsHigh and PsLow will automatically get good default values that are specific for the amount of symbols that are used:

\[
PsHigh = (M-1)/M - 0.15 \\
PsLow = PsHigh/10 \\
M = \text{amount of symbols.}
\]

It is recommended to \textbf{not} use any arguments so default arguments are used! Now the crucial members have appropriate values and the simulation may start. This is either done with the function \texttt{SER::StartSim()} or the function \texttt{SER::StartMultiSim(int maxIter, double threshold)}. The first function just simulates one time and when using extrapolation, hopes that the extrapolation succeeds. The second function simulates maxIter times and chooses the best result as the correct one. Threshold is a parameter that breaks the simulation when a good result is obtained. A good value for threshold should be 1.0. The user must not use any arguments at all and then the default values for maxIter and threshold are used. The default value for maxIter is 3 and the default value for threshold is 0.0. There will be some tutorials here, which capture all the situations the user may face.
8.3 Tutorials

TUT 1.
- Use 5 symbols from files.
- Use Gaussian noise (V=2)
- Just simulate one time
- Plot the simulation result

```cpp
int main()
{
    int M = 5;
    SER ser;
    Signal *reference = new Signal[M];
    reference[0] = Signal("signal1.dat", "time.dat");
    reference[1] = Signal("signal2.dat", "time.dat");
    ser.SetRefSignals(reference, M);
    ser.Search_Good_Bins_For_Extrapolation();
    ser.StartSim();
    ser.PlotSim();
    ser.PrintResult();
}
```

TUT 2.
- Use 16-PSK modulation
- Use Gaussian noise (V=2)
- Simulate 10 times and choose the best result.
- While simulating 10 times it may be good to not have so much accuracy. Else it would take a long time simulating. Use minError = 1000 instead of the default value of 10,000.
- Plot simulations result vs. theory result.

```cpp
int main()
{
    int M = 16;
    SER ser;
    double power = 1.0; //signal power
    double ts = 0.0000001; //symbol time
    double fc = 3.0*(1/ts); //carrier freq
    double fs = 20.0*fc; //sample freq
    ser.MakePSK(power, ts, fc, fs, M); //make 16-PSK
    ser.Set_MinErrors(1000);
    ser.StartMultiSim(10, 0.0);
    ser.PlotSim_And_Theory(2, M); // arg1: 1=QAM 2=PSK
    ser.PrintResult();
}
```
SER::StartMultiSim(10, 0.0) can be changed to SER::StartMultiSim_Using_Randomness(10, 0.0), if a small change in location for the reliable bins should be made in each simulation.

TUT 3.
- Use 8-PSK modulation
- Use Gaussian noise (V=2)
- Simulate one time using only MC-simulations. No extrapolation.
- Only use minError = 10, because accuracy is not that important when not extrapolating.
- Set minSER = 1.0e-05. Too low value will make the simulation time go to infinity.
- Plot simulations result vs. theory result.

```c++
int main(){
    int M = 8;
    SER ser;
    double power = 2.0; //signal power
    double ts = 0.000001; //symbol time
    double fc = 4.0*pi/(ts); //carrier freq
    double fs = 15.0*fc; //sample freq
    ser.makePSK(power, ts, fc, fs, M); //make 8-PSK
    ser.Set_MinErrors(10);
    ser.Set_MinSER(0.00001);
    ser.Search_Good_Bins(0.6, 0.1, 4);
    ser.Set_UseExtrapol(false);
    ser.StartSim();
    ser.PlotSim_And_Theory(2, M); // arg1: 1=QAM 2=PSK;
    ser.PrintResult();
}
```

The function Search_Good_Bins, is only for calculating bin size and start EsN0. What it does is that it tries to find 4 equidistant simulated SER between 0.6 > SER > 0.1. The user could use ser.Search_Good_Bins(0.6, 0.1, 10), but the only difference is that the simulation will use a smaller bin size.

TUT 4.
- Use BPSK modulation
- Use noise with shape parameter V = 3.
- Let the extrapolation not know what shape parameter that is used. This is actually bad, but this tutorial is just to show that the extrapolation algorithm has a strength of finding what noise that is used.
- Use minError = 10,000.
- Plot simulations result vs. theory result.

```c++
int main(){
    int M = 2;
    SER ser;
    double power = 1.0; //signal power
```
double ts = 0.00001; //symbol time
double fc = 2.0*(1.0/ts); //carrier freq
double fs = 15.0*fc; //sample freq
ser.MakePSK(power, ts, fc, fs, M); //make BPSK
ser.Set_NoiseParam(3); //a slower noise generator is used.
ser.Make_EP_Not_Know_What_Noise();
ser.Set_NumBins(15);
ser.Search_Good_Bins_For_Extrapolation();
ser.StartSim();
ser.PlotSim();
ser.PrintResult();

minErrors does not work when using the function Make_EP_Not_Know_What_Noise. The reason for that is that any \( V \) can be the correct shape parameter. minError will not generate the same number of bins for all the \( V \) and that is why the function Set_NumBins should be set (if not the default value of 100 will be used). It is ALWAYS better to make the EP know what noise that is used. It is also better to use Gaussian noise \( V=2 \), because that is the most common noise in digital communication and it has a faster noise generator. It is also easier to get a good extrapolation result using Gaussian noise.

**TUT 5.**

Tutorial 5 is about making a throughput analysis using the UWB class.

- Do a normal SER simulation.
- Use the four last EsN0 values in the SER simulation and make a throughput analysis on them.

```c
int main(){
    double LM = 4.0; //Link Margin [dB]
    double NF = 7.0; //Noise Figure [dB]
    double Bs = 2.5*pow(10.0, 9.0); // Bandwidth [Hz]
    double Gain_dB_r = 0; //Receiver Gain
    double dist_min = 12.5; //Minimum distance for analysis
    double dist_max = 102.5; //Maximum distance for analysis
    int dist_numBins = 19; //Plot resolution
    int Nplot = 4; //Number of curves in the same plot
    double Fcarrier = 3.75*pow(10.0, 9.0); //Carrier frequency

    SER ser;
    UWB uwb;
    ser.Make16QAM(1.0, (1.0/Fcarrier)*3.0 , Fcarrier, Fcarrier*20.0);
    ser.Search_Good_Bins_For_Extrapolation();
    ser.StartSim();

    int offset = ser.Get_NumBins()-Nplot; // The four last EsN0 values is used for analysis.
    uwb.SetInfo(LM, NF, Bs, Fcarrier, Gain_dB_r, ser.Get_EsN0dB_pointer_using_offset(offset), Nplot);
    uwb.Calc_Distances(dist_min, dist_max, dist_numBins);
    uwb.Throughput_analysis();
    uwb.Plot_Throughput(Nplot);
}```
8.4 The contents of the classes

All member variables are private and all the member functions are public.

signal.h

Member Functions:
Signal();
Signal(char *signal_File, char *time_File);
void NextData(double Sval, double Tval);
void NextData(double Sval);
void EmptySignal();
void Set_T(vec vector);
void Set_S(vec vector);
vec Get_T();
vec Get_S();

Member Variables:
vec T, S;
int length;

SER.h

Member Functions:
SER();
int errors_Mfilter(int symbol_index, double scale);
int errors_Mfilter(int symbol_index);
void MonteCarlo(double &SER, int min_errors, double scale);
void MonteCarlo(double &SER, int min_errors);
void MonteCarlo(double &SER);
void Make16QAM(double Signal_power, double ts, double fc, double fs);
void MakeBPSK(double Signal_power, double ts, double fc, double fs);
void MakePSK(double Signal_power, double ts, double fc, double fs, int M);
void SetRefSignals(Signal Ref[], int M);
void ScaleRefSignals(double scale);
void CalcN0(double EsN0_dB);
void Calc_Ps_theory(int mod, int M);
void SetInfo(double EsN0start_dB, int numBins, double binSize, int minErrors, int np);
void SetInfo(int numBins, int np);
void SetInfo(int numBins);
void Search_Good_Bins_For_Extrapolation(double PsHigh, double PsLow);
void Search_Good_Bins_for_Extrapolation();
void Search_Good_Bins(double PsHigh, double PsLow, int numBins);
double Power_To_EsN0_dB(double power);
double QAM_theory(double EsN0_dB, int M);
double PSK_theory(double EsN0_dB, int M);
void Set_N0(double N0);
void Set_NumBins(int N);
void Set_BinSize(double size);
void Set_Ep(ExtraPol ep[]);
void Set_MinErrors(int minErrors);
void Set_NoiseParam(int p);
void Set_MinSER(double minSER);
void Set_Printout(bool printout);
void Set_UseExtrapol(bool useExtrapol);
void Set_StartEsN0dB(double EsN0dB);
int Get_NoiseParam();
double Get_MeanAmplitude();
double Get_MeanAmplitude(double scale);
double Get_Energy();
double Get_Energy(double scale);
double Get_Power();
double Get_Power(double scale);
double Get_BinSize();
double Get_N0();
double Get_MinSER();
int Get_MinErrors();
bool Get_Printout();
bool Get_UseExtrapol();
ExtraPol * Get_EpPointer();
int Get_NumBins();
double * Get_EsN0dB_pointer();
double * Get_EsN0dB_pointer_using_offset(int offset);
double * Get_SER_pointer();
void Calculate_All_EsN0dB();
void Calculate_All_EsN0dB(double offset);
void Make_EP_Not_Know_What_Noise();
void PlotSim_And_Theory(int mod, int M);
void PlotSim_And_Theory();
void PlotSim();
void SetExtrapInfo(int VE_max);
void SetExtrapInfo(int VE_max, int ExtrapolIndex);
void StartSim();
void StartSim(double offset, int extrapolIndex);
void StartMultiSim(int maxIter, double threshold);
void StartMultiSim_Using_Randomness(int maxIter, double threshold);
void StartMultiSim_Using_Randomness();
void PrintResult();

Member Variables:
Signal r_signal, *reference;
double N0, binSize, *EsN0dB, *Ps, *Ps_theory, Fs, Ts, minSER;
int M, numBins, numRelBins, noiseParam, minErrors, bestExtrapIndex;
bool useExtrapol, printout, extrapol_Know_What_Noise;
ExtraPol *ep;
Real_Timer tt;

ExtraPol.h

Member Functions:
ExtraPol();
void Extrapolate(double Ps_rel[], int NumRelBins, bool printout);
void Extrapolate(double Ps_rel[], int NumRelBins);
double Get_Difference();
double *Get_PsPointer();
int Get_NumBins();
int Get_VEmax();
int Get_ChosenV();
void Set_VEmax(int V);
void Set_ChosenV(int V);
void Set_NumBins(int numBins);
void Set_MinSER(double minSER);
void Set_KnownNoise(int n);
int fact(int number);
int combin(int n, int k);

Member Variables:
int VE_max, numBins, ChosenV;
static int knownNoise;
double difference, *Ps, minSER;

UWB.h

Member Functions:
UWB();
void SetInfo(double LinkM, double NoiseF, double B, double Fc, double Gr, double EsN0_dB[], int N);
void Calc_Distances(double dmin, double dmax, int dbins);
void Set_LM(double LM);
void Set_NF(double NF);
void Set Bs(double Bs);
void Set_Fcarrier(double Fcarrier);
void Set_GaindB(double GaindB);
void Set_EsN0dB(double EsN0dB[]);
void Set_Distance(double distance[]);
void Set_NumData(int numData);
void Set_NumCurves(int numCurves);
double Get_LM();
double Get_NF();
double Get_Bs();
double Get_Fcarrier();
double Get_GaindBr();
double * Get_EsN0dB_Pointer();
double * Get_Distance_Pointer();
int Get_NumData();
int Get_NumCurves();
void Throughput_analysis();
void Plot_Throughput(int N);
void PrintResult();

**Member Variables:**
double LM, NF, Bs, Fcarrier, Gain_dB_r, *EsN0dB, *distance;
int numData, numCurves;
mat SymbolRate;
BIBLIOGRAPHY


