Irfan Ali / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 1, January -February 2013, pp.706-711

# Bit-Error-Rate (BER) Simulation Using MATLAB

**Irfan Ali**
M.Tech. Scholar, Jagan Nath University, Jaipur (India)

## Abstract

**This paper introduce the Bit error rate, (BER) simulation using Mat lab. Bit error rate, (BER) is a key parameter that is used in assessing systems that transmit digital data from one location to another. Systems for which bit error rate, is applicable include radio data links as well as fiber optic data systems, Ethernet, or any system that transmits data over a network of some form where noise, interference, and phase jitter may cause quality degradation of the digital signal. Mat lab is an ideal tool for simulating digital communications systems, thanks to its easy scripting language and excellent data visualization capabilities. One of the most frequent simulation tasks in the field of digital communications is bit-error-rate testing of modems. The bit-error-rate performance of a receiver is a figure of merit that allows different designs to be compared in a fair manner. Performing bit-error-rate testing with Mat lab is very simple, but does require some prerequisite knowledge.**

**Keywords:** BER, **Mat lab, $E_b/N_o$,**

## I. INTRODUCTION

As the name implies, a bit error rate (BER) is defined as the rate at which errors occur in a transmission system. This can be directly translated into the number of errors that occur in a string of a stated number of bits. The definition of bit error rate can be translated into a simple formula:

BER = number of errors / total number of bits sent

If the medium between the transmitter and receiver is good and the signal to noise ratio is high, then the bit error rate will be very small possibly insignificant and having no noticeable effect on the overall system However if noise can be detected, then there is chance that the bit error rate will need to be considered. Although there are some differences in the way these systems work and the way in which bit error rate is affected, the basics of bit error rate itself are still the same. When data is transmitted over a data link, there is a possibility of errors being introduced into the system. If errors are introduced into the data, then the integrity of the system may be compromised. As a result, it is necessary to assess the performance of the system, and bit error rate, BER, provides an ideal way in which this can be achieved. Unlike many other forms of assessment, bit error rate, BER assesses the full end to end performance of a system including the transmitter, receiver and the medium between the two. In this way, bit error rate, BER enables the actual performance of a system in operation to be tested, rather than testing the component parts and hoping that they will operate satisfactorily when in place. The main reasons for the degradation of a data channel and the corresponding bit error rate, BER is noise and changes to the propagation path (where radio signal paths are used). Both effects have a random element to them, the noise following a Gaussian probability function while the propagation model follows a Rayleigh model. This means that analysis of the channel characteristics are normally undertaken using statistical analysis techniques. For fiber optic systems, bit errors mainly result from imperfections in the components used to make the link. These include the optical driver, receiver, connectors and the fiber itself. Bit errors may also be introduced as a result of optical dispersion and attenuation that may be present. Also noise may be introduced in the optical receiver itself. Typically these may be photodiodes and amplifiers which need to respond to very small changes and as a result there may be high noise levels present. Another contributory factor for bit errors is any phase jitter that may be present in the system as this can alter the sampling of the data. Signal to noise ratios and Eb/No figures are parameters that are more associated with radio links and radio communications systems. In terms of this, the bit error rate, BER, can also be defined in terms of the probability of error or POE. The determine this, three other variables are used. They are the error function, erf, the energy in one bit, Eb, and the noise power spectral density (which is the noise power in a 1 Hz bandwidth), No. It should be noted that each different type of modulation has its own value for the error function. This is because each type of modulation performs differently in the presence of noise. In particular, higher order modulation schemes (e.g. 64QAM, etc) that are able to carry higher data rates are not as robust in the presence of noise. Lower order modulation formats (e.g. BPSK, QPSK, etc.) offer lower data rates but are more robust. The energy per bit, Eb, can be determined by dividing the carrier power by the bit rate and is a measure of energy with the dimensions of Joules. No is a power per Hertz and therefore this has the dimensions of power (joules per second) divided by seconds). Looking at the dimensions of the ratio Eb/No all the dimensions cancel out to give a dimensionless ratio. It is important to note that

POE is proportional to Eb/No and is a form of signal to noise ratio.

## II. FACTORS AFFECTING BIT ERROR RATE

It can be seen from using Eb/No, that the bit error rate, BER can be affected by a number of factors. By manipulating the variables that can be controlled it is possible to optimize a system to provide the performance levels that are required. This is normally undertaken in the design stages of a data transmission system so that the performance parameters can be adjusted at the initial design concept stages. The interference levels present in a system are generally set by external factors and cannot be changed by the system design. However it is possible to set the bandwidth of the system. By reducing the bandwidth the level of interference can be reduced. However reducing the bandwidth limits the data throughput that can be achieved. It is also possible to increase the power level of the system so that the power per bit is increased. This has to be balanced against factors including the interference levels to other users and the impact of increasing the power output on the size of the power amplifier and overall power consumption and battery life, etc. Lower order modulation schemes can be used, but this is at the expense of data throughput. It is necessary to balance all the available factors to achieve a satisfactory bit error rate. Normally it is not possible to achieve all the requirements and some trade-offs are required. However, even with a bit error rate below what is ideally required, further trade-offs can be made in terms of the levels of error correction that are introduced into the data being transmitted. Although more redundant data has to be sent with higher levels of error correction, this can help mask the effects of any bit errors that occur, thereby improving the overall bit error rate.

## III. SIMULATION TOOL

MATLAB (matrix laboratory) is a calculating environment and fourth-generation programming language. Developed by Math Works, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Although Matlab is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic Computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems. In 2004, Matlab had around one million users across industry and academia. Matlab users come from various backgrounds of engineering, science, and economics. Matlab is widely used in academic and research institutions as well as industrial enterprises. Matlab is an ideal tool for simulating digital communications systems, thanks to its easy scripting language and excellent data visualization capabilities. One of the most frequent simulation tasks in the field of digital communications is bit-error- rate testing of modems. The bit-error-rate performance of a receiver is a figure of merit that allows different designs to be compared in a fair manner. Performing bit-error-rate testing with Mat lab is very simple, but does require some prerequisite knowledge. In Matlab, we represent continuous-time signals with a sequence of numbers, or samples, which are generally stored in a vector or an array. Before we can performance bit-error-rate test, we must precisely understand the meaning of these samples. We must know what aspect of the signal the value of these samples represents. We must also know the time interval between successive samples. For communications simulations, the numeric value of the sample represents the amplitude of the continuous-time signal at a specific instant in time. We assume this amplitude is a measurement of voltage, though it could just as easily be a measurement of current. The time between successive samples is, by definition, *Ts*. This tells us how often the continuous time signal was sampled. Instead of specifying *Ts* , we usually specify the sampling frequency, *fs* , which is the inverse of *Ts*. For convenience, we will always associate a sample value of 1.0 with a voltage of exactly one volt. Furthermore, we will always assume a resistance of exactly one ohm. This allows us to dispense with the notion of resistance altogether. For our simulations, we will represent a continuous time signal as an array of samples, the numeric value of which is in units of volts, referenced to a resistance of one ohm. Usually, the sampling frequency is 8 KHz, but other sampling frequencies are also in common use, so the sampling frequency should always be specified. Suppose we have a signal $x(n)$, where $n$ is an index of the sample number. We define the instantaneous power of the signal as:

$$P_{ins} \equiv x^2(n).$$

In other words, the instantaneous power of a sample is just the value of that sample squared. Since the units of the sample are volts, the units of the power are watts. A far more useful quantity is the average power, which is simply the average of the instantaneous power of every sample in the signal. For signal $x(n)$, of $N$ samples, we have:

$$Pave \equiv \frac{1}{N} \sum_{n=1}^{N} x^2(n). \qquad (1)$$

Note that this is simply the sum of the square of all samples, divided by the number of samples. One

way to compute the average power, 'pav', of signal 'x', using Matlab is:

pav= sum(x.^2)/length(x).

If our signal has a mean of zero, or in other words, no DC component, we can find the average power of the signal by taking its variance. This works because:

$\sigma(x) \equiv E[x^2]-(E[x])^2$,

which states: the variance of a signal is the mean of its square, minus the square of its mean. If the mean is zero, the variance is just the mean of the square, exactly the same as the average power. Therefore, if a signal has no DC value, we can compute its average power by finding its variance. We need to be careful using the variance to find the average power of a signal. This technique only works if the mean of the signal is zero. If the mean is not zero, we must use (1), which always works, regardless of whether the mean is zero or not. By definition, power is the time derivative of energy; or equivalently, energy is the time integral of power. For sampled signals, integration reduces to a summation. Since energy is the product of power and time, the total energy of a signal must be equal to its average power multiplied by its duration. Furthermore, the duration of a signal is its length in samples, divided by the sampling frequency, in samples per second. Therefore:

$$E_{tot} = Pave \cdot t$$
$$= \frac{1}{N} \sum_{n=1}^{N} x^2(n) \cdot \frac{N}{fs},$$
$$= \frac{1}{fs} \sum_{n=1}^{N} x^2(n). \qquad (2)$$

The Matlab command for finding the total energy, 'et', of signal 'x', that has sampling rate '$f_s$', is:

et= sum(x.^2)/ $f_s$.

## IV. SIMULATION PROCEDURE

Bit-error-rate testing requires a transmitter, a receiver, and a channel. We begin by generating a long sequence of random bits, which we provide as input to the transmitter. The transmitter modulates these bits onto some form of digital signalling, which we will send though a simulated channel. Bit-error-rate performance is usually depicted on a two dimensional graph. The ordinate is the normalized signal-to-noise ratio (SNR) expressed as $Eb/N_0$: the energy-per-bit divided by the one-sided power spectral density of the noise, expressed in decibels (dB). The abscissa is the bit-error-rate, a dimensionless quantity, usually expressed in powers of ten. To create a graph of bit-error-rate versus SNR, we plot a series of points. Each of these points requires us to run a simulation at a specific value of SNR. To obtain the bit-error-rate at a specific SNR, we follow the procedure given below

### A. Run Transmitter

The first step in the simulation is to use the transmitter to create a digitally modulated signal from a sequence of pseudo-random bits. Once we have created this signal, $x(n)$, we need to make some measurements of it.

### B. Establish SNR

The signal-to-noise-ratio (SNR), $E_b/N_0$, is usually expressed in decibels, but we must convert decibels to an ordinary ratio before we can make further use of the SNR. If we set the SNR to $m$ dB, then $E_b/N_0 = 10m/10$. Using Matlab, we find the ratio, 'ebn0', from the SNR in decibels, 'snrdb', as:

ebn0= 10^(snrdb/10).

Note that $E_b/N_0$ is a dimensionless quantity.

### C. Determine $E_b$

Energy-per-bit is the total energy of the signal, divided by the number of bits contained in the signal. We can also express energy-per-bit as the average signal power multiplied by the duration of one bit. Either way, the expression for $E_b$ is:

$$E_b = \frac{1}{N.fbit} \sum_{n=1}^{N} x^2(n)$$

where $N$ is the total number of samples in the signal, and $f_{bit}$ is the bit rate in bits-per-second. Using Matlab, we find the energy-per-bit, 'eb', of our transmitted signal, 'x', that has a bit rate 'fb', as:

eb = sum(x.^2)/(length(x)*fb).

Since our signal, $x(n)$, is in units of volts, the units of $E_b$ are Joules.

### D. Calculate $N_0$

With the SNR and energy-per-bit now known, we are ready to calculate $N_0$, the one-sided power spectral density of the noise. All we have to do is divide $E_b$ by the SNR, providing we have converted the SNR from decibels to a ratio. Using Matlab, we find the power spectral density of the noise, 'n0', given energy- per-bit 'eb', and SNR 'ebn0', as:

n0 = eb/ebn0.

The power spectral density of the noise has units of Watts per Hertz.

### E. Calculate $\sigma_n$

The one-sided power spectral density of the noise, $N_0$, tells us how much noise power is present in a 1.0 Hz bandwidth of the signal. In order to find

the variance, or average power, of the noise, we must know the noise bandwidth. For a real signal, $x(n)$, sampled at $fs$ Hz, the noise bandwidth will be half the sampling rate. Therefore, we find the average power of the noise by multiplying the power spectral density of the noise by the noise bandwidth:

$$\sigma_n = \frac{N0.fs}{2} ,$$

where $\sigma_n$ is the noise variance in W, and $N_0$ is the one-sided power spectral density of the noise inW/Hz. Using Matlab, the average noise power, 'pn', of noise having power spectral density 'n0', and sampling frequency 'fs', is calculated as:

```
pn= n0*fs/2.
```

The average noise power is in units of Watts.

F. **Generate Noise**

Although the communications toolbox of Matlab has functions to generate additive white Gaussian noise, we will use one of the standard built-in functions to generate AWGN. Since the noise has a zero mean, its power and its variance are identical. We need to generate a noise vector that is the same length as our signal vector $x(n)$, and this noise vector must have variance $\sigma_n$ W. The Matlab function 'randn' generates normally distributed random numbers with a mean of zero and a variance of one. We must scale the output so the result has the desired variance, $\sigma_n$. To do this, we simply multiply the output of the 'randn' function by p¾n. We can generate the noise vector 'n', as:

```
n = sqrt(pn)*randn(1,length(x));.
```

Like the signal vector, the samples of the noise vector have units of volts.

G. **Add Noise**

We create a noisy signal by adding the noise vector to the signal vector. If we are running a fixed-point simulation, we will need to scale the resulting sum by the reciprocal of the maximum absolute value, so the sum stays within amplitude limits of ±1.0. Otherwise, we can simply add the signal vector 'x' to the noise vector 'n' to obtain the noisy signal vector 'y' as:

```
y = x+n;.
```

H. **Run Receiver**

Once we have created a noisy signal vector, we use the receiver to demodulate this signal. The receiver will produce a sequence of demodulated bits, which we must compare to the transmitted bits, in order to determine how many demodulated bits are in error.

I. **Determine Offset**

Due to filtering and other delay-inducing operations typical of most receivers, there will be an offset between the received bits and the transmitted bits. Before we can compare the two bit sequences to check for errors, we must first determine this offset. One way to do this is by correlating the two sequences, then searching for the correlation peak. Suppose our transmitted bits are stored in vector 'tx', and our received bits are stored in vector 'rx'. The received vector should contain more bits than the transmitted vector, since the receiver will produce (meaningless) outputs while the filters are filling and flushing. If the length of the transmitted bit vector is *lt x* , and the length of the received vector is *lrx* , the range of possible offsets is between zero and *lrx* −*lt x* −1. We can find the offset by performing a partial cross-correlation between the two vectors. Using Matlab, we can create a partial cross-correlation, 'cor', from bit vectors 'tx' and 'rx', with the following loop:

```
for lag= 1 : length(rx)−length(tx)−1,
cor(lag)= tx*rx(lag : length(tx)−1+lag)′;
end.
```

The resulting vector, 'cor', is a partial cross-correlation of the transmitted and received bits, over the possible range of lags: 0 : *lrx* −*lt x* −1. We need to find the location of the maximum value of 'cor', since this will tell us the offset between the bit vectors. Since Matlab numbers array elements as 1 : *N* instead of as 0 :*N*−1, we need to subtract one from the index of the correlation peak. Using Matlab, we find the correct bit offset, 'off', as:

```
off= find(cor== max(cor))−1.
```

J. **Create Error Vector**

Once we know the offset between the transmitted and received bit vectors, we are ready to calculate the bit errors. For bit values of zero and one, a simple difference will reveal bit errors. Wherever there is a bit error, the difference between the bits will be ±1, and wherever there is not a bit error, the difference will be zero. Using Matlab, we calculate the error vector, 'err', from the transmitted bit vector, 'tx', and the received bit vector, 'rx', having an offset of 'off', as:

```
err = tx−rx(off+1 : length(tx)+off);.
```

K. **Count Bit Errors**

The error vector, 'err' contains non-zero elements in the locations where there were bit errors. We need to tally the number of non-zero elements, since this is the total number of bit errors in this simulation. Using Matlab, we calculate the

total number of bit errors, 'te', from the error vector 'err' as:

te= sum(abs(err)).

### L. Calculate Bit-Error-Rate

Each time we run a bit-error-rate simulation, we transmit and receive a fixed number of bits. We determine how many of the received bits are in error, then compute the bit-error-rate as the number of bit errors divided by the total number of bits in the transmitted signal. Using Matlab, we compute the bit error-rate, 'ber', as:

ber= te/length(tx),

where 'te' is the total number of bit errors, and 'tx' is the transmitted bit vector.

## V. Simulation Results

Performing a bit-error-rate simulation can be a lengthy process. We need to run individual simulations at each SNR of interest. We also need to make sure our results are statistically significant.

### A. Statistical Validity

When the bit-error-rate is high, many bits will be in error. The worst-case bit-error-rate is 50 percent, at which point, the modem is essentially useless. Most communications systems require bit-error-rates several orders of magnitude lower than this. Even a bit-error-rate of one percent is considered quite high. We usually want to plot a curve of the bit-error-rate as a function of the SNR, and include enough points to cover a wide range of bit-error-rates. At high SNRs, this can become difficult, since the bit-error-rate becomes very low. For example, a bit-error-rate of $10^{-6}$ means only one bit out of every million bits will be in error. If our test signal only contains 1000 bits, we will most likely not see an error at this Bit-error-rate. In order to be statistically significant, each simulation we run must generate some number of errors. If a simulation generates no errors, it does not mean the bit-error-rate is zero; it only means we did not have enough bits in our transmitted signal. As a rule of thumb, we need about 100 (or more) errors in each simulation, in order to have confidence that our bit-error-rate is statistically valid. At high SNRs, this can require a test signal containing millions, or even billions of bits.

### B. Plotting

Once we perform enough simulations to obtain valid results at all SNRs of interest, we will plot the results. We begin by creating vectors for both axes. The X-axis vector will contain SNR values, while the Y-axis vector will contain bit-error- rates. The Y-axis should be plotted on a logarithmic scale, whereas the X-axis should be plotted on a linear scale. Supposing our SNR values are in vector 'xx', and our corresponding bit-error-rate values are in vector 'yy', we use Matlab to plot:

semilogy (xx,yy,′o′).

Fig. 1, 2 and 3 shows example of plot of the results of a bit-error-rate simulation at different $E_b/N_0$.

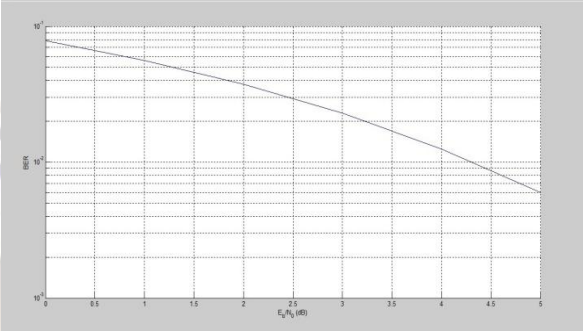

Figure 1

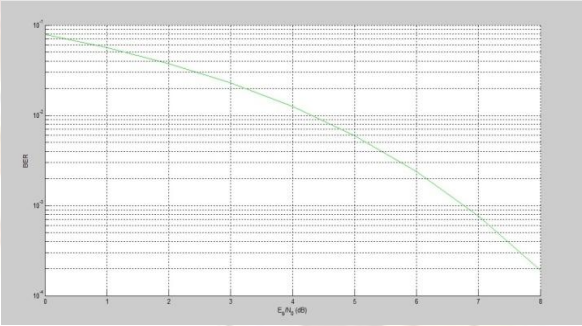

Figure 2

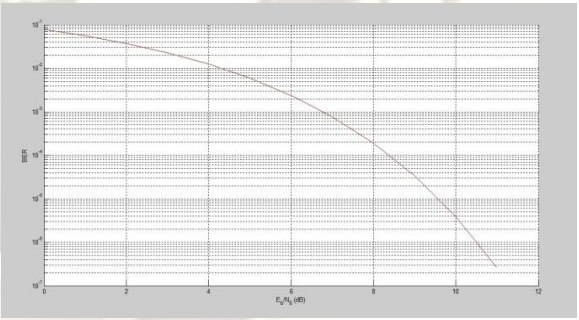

Figure 3

**Matlab Code For BER Simulation**

```
N = 10^6 % number of bits or symbols
rand('state',100); % initializing the rand() function
randn('state',200); % initializing the randn() function

% Transmitter
ip = rand(1,N)>0.5; % generating 0,1 with equal probability
s = 2*ip-1; % BPSK modulation 0 -> -1; 1 -> 1
n = 1/sqrt(2)*[randn(1,N) + j*randn(1,N)]; % white gaussian noise, 0dB variance
```

```
Eb_N0_dB = [-3:10]; % multiple Eb/N0 values

for ii = 1:length(Eb_N0_dB)
  % Noise addition
  y = s + 10^(-Eb_N0_dB(ii)/20)*n; % additive white gaussian noise

  % receiver - hard decision decoding
  ipHat = real(y)>0;

  % counting the errors
  nErr(ii) = size(find([ip- ipHat]),2);

end

simBer = nErr/N; % simulated ber
theoryBer = 0.5*erfc(sqrt(10.^(Eb_N0_dB/10))); % theoretical ber

% plot
close all
figure
semilogy(Eb_N0_dB,theoryBer,'b.-');
hold on
semilogy(Eb_N0_dB,simBer,'mx-');
axis([-3 10 10^-5 0.5])
grid on
legend('theory', 'simulation');
xlabel('Eb/No, dB');
ylabel('Bit Error Rate');
title('Bit error probability curve for BPSK modulation');
```

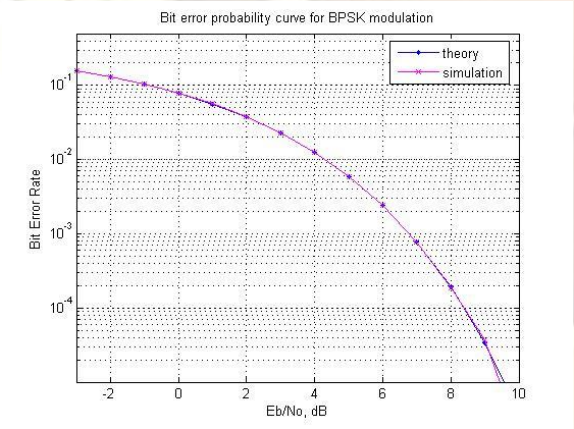


Figure 4

## VI. CONCLUSION

Bit error rate BER is a parameter which gives an excellent indication of the performance of a data link such as radio or fibre optic system. As one of the main parameters of interest in any data link is the number of errors that occur, the bit error rate is a key parameter. A knowledge of the BER also enables other features of the link such as the power and bandwidth, etc to be tailored to enable the required performance to be obtained. Bit error rate (BER) testing, is a powerful methodology for end to end testing of digital transmission systems. A BER test provides a measurable and useful indication of the performance of the performance of the system that can be directly related to its operational performance. If the BER rises too high then the system performance will noticeably degrade. If it is within limits then the system will operate satisfactorily. We simulate the Bit-error-rate performance of digital communication system by adding a controlled amount of noise to the transmitted signal. This noisy signal then becomes the input to the receiver. The receiver demodulates the signal, producing a sequence of recovered bits. Finally, we compare the received bits to the transmitted bits, and tally up the errors through BER versus $E_b/N_0$ plot.

## REFERENCES

[1] JAMES E. GILLEY: "BIT-ERROR-RATE SIMULATION USING MAT LAB", TRANSCRYPT INTERNATIONAL, INC., AUGUST 19, 2003.

[2] WIKIPEDIA, FREE ENCYCLOPAEDIA, ARTICLE ON BIT ERROR RATE HTTP://EN.WIKIPEDIA.ORG/WIKI/BIT_ERROR_RATE.

[3] WIKIPEDIA, FREE ENCYCLOPAEDIA, ARTICLE ON SIGNAL TO NOISE RATIO HTTP://EN.WIKIPEDIA.ORG/WIKI/S/N_RATIO

[4] JOHN. G. PROAKIS, "DIGITAL COMMUNICATIONS", MCGRAW-HILL SERIES IN ELECTRICAL AND COMPUTER ENGINEERING, THIRD ED.

[5] THE MATH WORKS, INC., THE STUDENT EDITION OF MATLAB VERSION 7 USER'S GUIDE, PRENTICE HALL, ISBN 0-13-184979-4, 1995.

[6] D. HANSEL MAN AND B. LITTLEFIELD, MASTERING MATLAB 7. A COMPREHENSIVE TUTORIAL AND REFERENCE, PRENTICE HALL, UPPER SADDLE RIVER, NJ, 1998

[7] B. SKLAR, DIGITAL COMMUNICATIONS: FUNDAMENTALS AND APPLICATIONS, CH. 4, ENGLEWOOD CLIFFS, NJ: PRENTICE HALL, 1988.

[8] M. JERUCHIM, "TECHNIQUES FOR ESTIMATING THE BIT ERROR RATE IN THE SIMULATION OF DIGITAL COMMUNICATION SYSTEMS," IEEE J. SELECT. AREAS COMMUNICATION., VOL. SAC-2, PP. 153-170, JAN.1994.