A TRIDENT SCHOLAR PROJECT REPORT

NO. 270

Development and Implementation of an Adaptive Error Correction Coding Scheme for a Full Duplex Communications Channel

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

This document has been approved for public release and sale; its distribution is unlimited.

20000424 161

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response including g the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	5 May 1999	

4. TITLE AND SUBTITLE
Development and Implementation of an Adaptive Error Correction Coding Scheme for a Full Duplex Communications Channel

5. FUNDING NUMBERS

6. AUTHOR(S)
Waterston, John W.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD

8. PERFORMING ORGANIZATION REPORT NUMBER
USNA Trident Scholar project report no. 270 (1999)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Accepted by the U.S. Trident Scholar Committee

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT: This research investigates, via simulation, the bit error probability (BEP) associated with a variable redundancy coding scheme operating in a wireless communications environment. Within a slowly varying (flat fading) Rayleigh channel, adaptive algorithms provide increased throughput over fixed coding implementations. From a family of BCH codes of the same block length (n=63), a code with appropriate redundancy is chosen depending on the receiver's estimation of the current conditions experienced in this channel. Two different decision techniques are compared. The first method statistically evaluates the receiver's input and calculates the signal to noise ratio (Eb/No), while the second method observes the number of corrected errors in recently decoded blocks. With this information, the adaptive system decides to modify the correction ability of the code, and then transmits this decision to the encoder over a low bandwidth feedback channel. The correction ability can be changed on a block by block basis. This algorithm is implemented in software and, therefore, can be optimized for many real world communications systems. The low cost of high speed microprocessors and DSPs allows for the development of a robust adaptive coding system in hardware. The results are compared against fixed coding implementations and show that the adaptive process maintains a better efficiency (η = k/n) of information rate while keeping the bit error probability near the level obtained by maximum encoding.

14. SUBJECT TERMS
Variable Redundancy, Rayleigh Fading Channel, BCH Codes, Forward Error Correction, Channel Estimation, Adaptive Coding.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

U.S.N.A. — Trident Scholar project report; no. 270 (1999)

Development and Implementation of an Adaptive Error Correction Coding Scheme for a Full Duplex Communications Channel

by

Midshipman John W. Waterston, Class of 1999
United States Naval Academy
Annapolis, Maryland

(signature)

Certification of Advisers Approval

Assistant Professor Ellen Curran K. Wooten
Department of Electrical Engineering

(signature)

05 May 1999
(date)

Associate Professor William E. Bennett
Department of Electrical Engineering

(signature)

05 May 1999
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Chair, Trident Scholar Committee

(signature)

5 May 1999
(date)

USNA-1531-2

Abstract

This research investigates, via simulation, the bit error probability (BEP) associated with a variable redundancy coding scheme operating in a wireless communications environment. Within a slowly varying (flat fading) Rayleigh channel, adaptive algorithms provide increased throughput over fixed coding implementations. From a family of BCH codes of the same block length (n=63), a code with appropriate redundancy is chosen depending on the receiver's estimation of the current conditions experienced in this channel. Two different decision techniques are compared. The first method statistically evaluates the receiver's input and calculates the signal to noise ratio (Eb/No), while the second method observes the number of corrected errors in recently decoded blocks. With this information, the adaptive system decides to modify the correction ability of the code, and then transmits this decision to the encoder over a low bandwidth feedback channel. The correction ability can be changed on a block by block basis. This algorithm is implemented in software and, therefore, can be optimized for many real world communications systems. The low cost of high speed microprocessors and DSPs allows for the development of a robust adaptive coding system in hardware. The results are compared against fixed coding implementations and show that the adaptive process maintains a better efficiency ($\eta = k/n$) of information rate while keeping the bit error probability near the level obtained by maximum encoding.

Keywords: Variable Redundancy, Rayleigh Fading Channel, BCH Codes, Forward Error Correction, Channel Estimation, Adaptive Coding.

Acknowledgments

Throughout my Trident project I have worked with many wonderful people whom I would like to thank. My advisors, Dr. William Bennett and Dr. Ellen Wooten, have shared my excited discoveries and also guided me through periods of frustration. Without the two of them, I would not have been able to move past the endless problems and begin structured investigation. CDR Welch, with his expertise in wireless communications, has focused my research towards a realistic system and a worthwhile goal. He has been a wealth of information with an office door that is always open. Dr. Antal Sarkady's lectures in communications have given me the inspiration to pursue this field for my time at the Academy and another two years at Stanford.

Most of all, my parents have always been a great source of support and motivation. Even when they do not understand the problem, they are always there for me. Mark and Casey have kept me sane, allowing me to take a break from my academic pursuits. Thanks again.

"If I have seen further, it is by standing on the shoulders of Giants"

-Sir Issac Newton

Table of Contents

Abstract 1

Acknowledgments 2

Figures 4

1. Introduction 5
    1.1 Background 5
    1.2 Project Description 6
    1.3 Contributions 8
    1.4 Overview 8

2. Project Design 10
    2.1 Communication System 10
        2.1.1 Data Input 11
        2.1.2 BCH Encoder 11
        2.1.3 BPSK Modulator 14
        2.1.5 Additive White Gaussian Noise 15
        2.1.6 Demodulator 15
        2.1.7 BCH Decoder 15
    2.2 Adaptive System 16
        2.2.1 Statistical Adaptive Decision Method 16
        2.2.2 Error Based Adaptive Decision Method 18

3. Procedure 20
    3.1 Simulation Technique 20
    3.2 Output Validation 20
    3.3 Experimental Results 24

4. Conclusions 28
    4.1 Challenges 29
    4.2 Future Work 29
    4.3 Future Applications 30

References 31

Appendix 32
    BCH Code Parameters 32
    MATLAB Files 33

Figures

Figure 1. System Block Diagram 10

Figure 2. Graphic Code Description 11

Figure 3. Signal Transformation through a Communications System. 13

Figure 4. Statistical Adaptive Decision Method 17

Figure 5. Error-Based Adaptive Decision Method 18

Figure 6. BPSK Waveform and AWGN 21

Figure 7. Rayleigh Channel and AWGN 22

Figure 8. Family of BCH Codes: Length 63 23

Figure 9. Efficiency (η) vs. SNR 24

Figure 10. BEP vs. SNR 25

Figure 11. Metric vs. SNR 26

1. Introduction

As the amount of digital data has exponentially increased in recent years, it is more important to provide efficient data rates while keeping high data integrity. It is impractical to maintain a very low bit error probability (*BEP*) with fixed error correction systems. BEP is calculated by taking the total number of bit errors and dividing by the total number of bits of transmitted data. When redundant bits are combined with data, less data is transmitted per message, thereby increasing the necessary time for transmission by up to a factor of nine. In simulation this is the difference between transmissions with an efficiency of 100 percent and those with 11 percent efficiency due to maximum levels of error correction. An example of the tradeoffs made in a specific application is that of P-3 Orion aircraft crews operating in mission situations. Because of the increased transmission time, these crews often tend to turn off error-control devices when transmitting digital pictures. They are willing to send degraded images rather than wait for the extremely slow transmission which occurs when using the correction scheme [7]. This inverse relationship between data reliability and efficiency is a focus of modern digital communications. Reliability is determined by the error correction ability of the system, while efficiency is determined by the transmission rate of the original data. The goal of this research is to develop and implement an adaptive system that could efficiently and effectively balance these two parameters for a wireless communications system.

1.1 Background

Error correction schemes were developed half a century ago. In 1948, Shannon

stated that coded data can be transmitted at rates near the channel capacity with an arbitrarily small probability of error. This statement has since led to much research in finding the most efficient methods of transmitting data in a noisy environment. Coding is the addition of redundant bits to a message in order to allow for the receiver to correct corrupted data without having to retransmit the message. More redundancy allows for more errors to be corrected upon reception. Coding transforms a block of information with length k into a coded block of length n. Shannon's theorem states more specifically that for any code of block length (n) with a rate ($R=k/n$) less than the channel capacity, there exists a block code with the probability of error (P_e) given below. $E(R)$ is a positive function of the code rate specified by transition probabilities. The probability of errors can be reduced by keeping the rate below capacity and increasing block length [6].

$$P_e \leq e^{-nE(R)} \tag{1.1}$$

The extensive and powerful class of codes used in this project were discovered by Bose, Chaudhuri, and Hocquenghem (BCH) in 1960.

1.2 Project Description

This Trident research project involved designing a system using error-correcting codes that adapts to a continuously changing communications environment. This system design was developed using programs written in the MATLAB® programming environment. This method of simulation allows for rigorous investigation while having very flexible design parameters. For example, the data type, coding, modulation, and channel parameters

all can be changed. The simulations for this research were completed in a communications system that modeled the characteristics of a digital cellular phone with a frequency of 881 MHz, 9600 bit per second data rate, and a speed of 30 kilometers per hour (IS-95 standard).

Included as part of a larger communications system, the project's central component is the adaptive system. There are many possible methods of adaptation applicable to digital communications, including slowing data rates, increasing transmitter power, employing diversity, and variable error control with codes [1]. A variable redundancy scheme was decided upon because the method involved coding, yet was easy to implement. In this scheme, the block length remains constant and the information length changes. The family of BCH codes with length 63 was chosen because it has 11 possible (n, k) combinations. This was few enough to keep the design simple, but allowed for enough states to demonstrate the system adapting to the environment. Keeping the block length constant keeps the transmitter operating at a constant bandwidth and simplifies the receiver structure. Adaptation is controlled by a device that makes a decision about the current channel condition and then selects an appropriate level of coding. This decision is then communicated over a low bandwidth feedback channel to the encoder. This process of evaluation and decision occurs with every transmitted block. Two different evaluation and decision-making methods are compared during the research.

Benice in 1966 looked at variable redundancy coding and said that this "adaptive technique was of little value [1]." However, the addition of greater computational ability brings new insight to the modern evaluation of this method in wireless environments.

1.3 Contributions

The entire adaptive error-correction system developed in this project incorporates new design aspects for the simulation and implementation of variable redundancy coding. A list of these specific contributions include:

(i) Developing two methods to make adaptation decisions

Before the adaptive system can reliably modify the current level of redundancy, development of the methods to properly determine the location of these transition decisions had to occur. One method uses statistical techniques to make these decisions. The other uses error information from the decoder to signal when more redundancy is needed. Within each method, threshold values control the response, so appropriate values had to be decided upon and evaluated.

(ii) Constructing a working simulation of a real-world system

A communications system is comprised of many sub-systems. Each of the sub-systems needs to be implemented and validated through comparison to known equations. Important blocks include: the random data generator, BCH encoder, BPSK modulator, Rayleigh Channel, additive white Gaussian noise, integrating receiver, and BCH decoder. Only after all of these components are working and validated can research into an unknown area be investigated.

1.4 Overview

With the given necessity for reliable data and rapid transmission, this research

examines an adaptive error-correction coding scheme. This method of variable redundancy will not only increase information sent per block when conditions are favorable, but will also increase redundancy as conditions deteriorate. The next section introduces the project design and the essential sub-systems using technical detail. The description of the validation and experimentation procedure follows, and this section concludes by presenting and comparing the results of both adaptive methods. Finally, the discussion is completed while drawing conclusions and listing areas of future work and application.

2. Project Design

The project can be divided into two fundamental components: a communications system and an adaptive system. Each is composed of many individual sub-systems. An introduction to each of these systems and sub-systems, with supporting examples, is now provided.

2.1 Communication System

The communications system for this project can be visualized by a multi-stage process which is implemented in MATLAB® code as individual sub-systems (Figure 1).

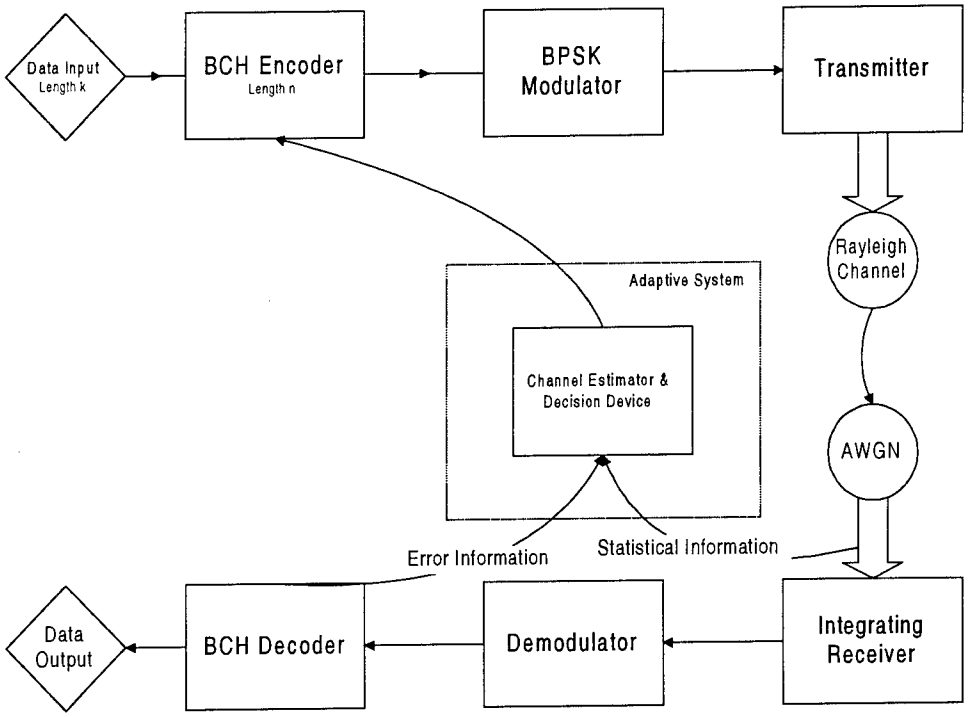

Figure 1. System Block Diagram

2.1.1 Data Input

In the first stage, the data is originally analog data (i.e., voice into a microphone) that is converted into binary symbols by an analog/digital converter, or the data is already in a digital format (i.e., satellite telemetry). The "data" chosen for simulation is characterized by a random set of 0's and 1's that are equiprobable. This data was created by modifying the output of a random number generator in MATLAB®. A benefit of this generator function was its ability to be reset, so that repeated simulations can be compared with exactly the same data set.

2.1.2 BCH Encoder

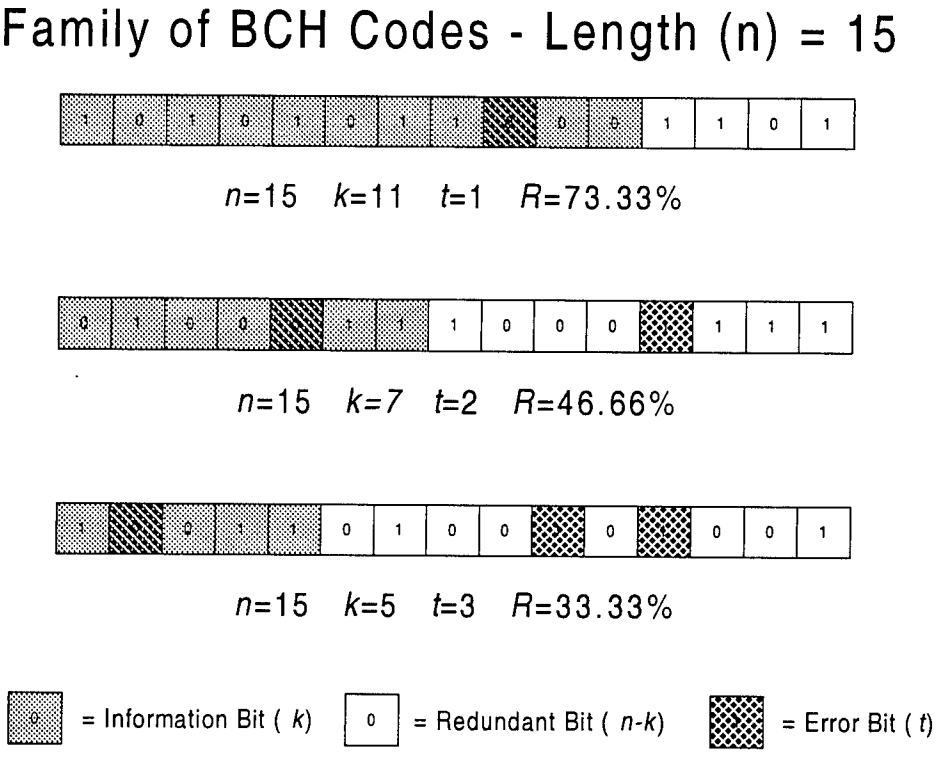

Figure 2. Graphic Code Description

This binary data stream of source bits is sent to the BCH encoder, where it is lengthened by coding bits and transformed into channel bits. Error-correcting codes rely on redundancy and data averaging. By the addition of redundant symbols, the uniqueness of each message is increased. In an optimal coding scheme, the data stream is divided into blocks of bits so that a percentage of the individual bits of the block can be in error without destroying all of the uniqueness contained in the block. To correct errors in a block, not all possible block sequences are used as legitimate messages. In fact, to correct t or fewer errors, each legitimate message must differ from each other by at least $2t+1$ positions. This minimum number of positions that a sequence must differ is called d or the *Hamming Distance*. The notations which are helpful when describing codes include n, the block code length, and k, the length of the information sequence. Therefore, the efficiency of a chosen code is the code rate, $R=k/n$, and the amount of redundancy is $n-k$. The efficiency is a good way to compare the performance of different codes. A visualization of these parameters is given in Figure 2.

The Bose-Chaudhuri-Hocquenghem (BCH) family of error-correcting codes is used to combat data errors in the system. Discovered in 1960, this class of cyclic codes defined from a generating polynomial $g(x)$ provides a large class of easily constructed codes with multiple block lengths (n) and code rates (R). For BCH codes, n always is equal to a power of two, minus one (e.g. 15;127;255). The longer the block length, the more possible levels of correction exist. A BCH code with length 63 has eleven levels, while the length 127 code has 17. The optimum code rates for BCH codes are between 1/3 and 3/4; at other rates the added redundancy does little to help performance and it can actually hinder performance.

The mathematics involved with code generation and decoding use linear algebra and Galois Theory, a form of modern algebra with finite fields [5,6]. The development of the techniques necessary to form the generator matrix of the block code is very complex and will not be covered in this report.

BCH codes can be implemented using shift registers. A shift register is a basic logic function available on most microprocessors and gate arrays which would allow hardware implementation with little difficulty.

In summary, the encoder adds redundancy following mathematical rules, increasing the block of data from length *k* to length *n*. An example of the signal transformation at the

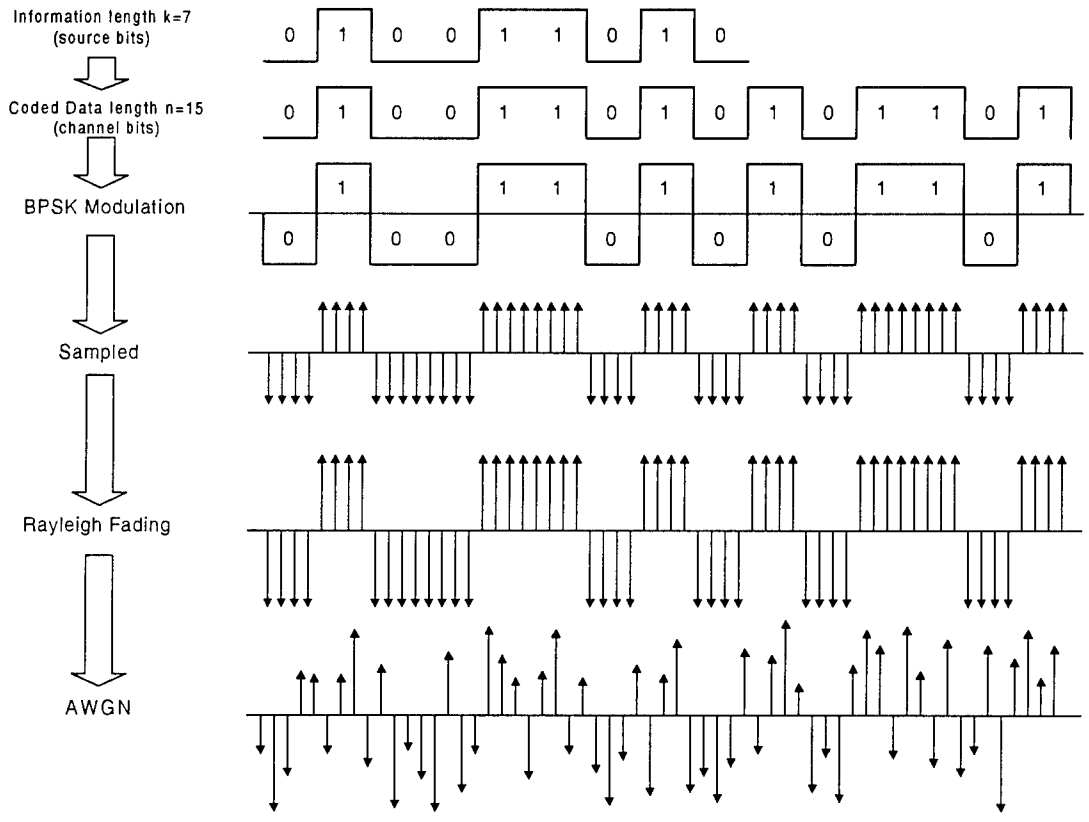

Figure 3. Signal Transformation through a Communications System.

encoder and subsequent portions of the communications system is shown in Figure 3. The constant block length (n=63) was used throughout the project.

2.1.3 BPSK Modulator

Within the current block of data (length 63 bits), each bit is sampled four times and modulated into a Binary Phase Shift Keyed (BPSK) signal, where -1 represents a binary zero and 1 represents a binary 1. The signal now has a mean of zero, thus removing the DC energy. Also the greater amplitude between binary levels decreases the signal's susceptibility to errors. The digital waveform output is sent to the transmitter.

2.1.4 Rayleigh Fading Channel

During transmission, the signal is modified by a channel modeled after the specific expected environment. The Rayleigh fading channel represents wireless propagation in a time-variant multipath scenario. This environment is applicable for cellular and HF applications. The slowly varying channel is described as flat fading since the fading does not change during the length of a block. Amplitude variations in the received signal, termed "signal fading," are caused by the constructive or destructive addition of incoming waveforms. The envelope at any instant is Rayleigh distributed and gives the channel its name. The digital waveform beginning with amplitudes of +/- 1 may now be represented by a smaller or larger value as shown in Figure 3. This fading is a random process, but it does have memory between adjacent states [8]. The channel used in simulation represents a moving digital cellular phone (IS-95) transmitting data at 9600 bps.

2.1.5 Additive White Gaussian Noise

Additive white Gaussian noise (AWGN) is added at the input of the receiver, where it represents the thermal noise at the receiver, with a zero mean and a Gaussian probability distribution. During simulation, each of the four sample points per bit has this noise added as shown in Figure 3. For different simulations, the level of the noise is modified, changing the signal to noise ratio (SNR).

2.1.6 Demodulator

The receiver integrates over every four samples (a bit) transforming a digital waveform back into a binary bit stream. Whether the result of this integration is greater or less than zero determines whether a 0 or a 1 was sent. Perfect framing and synchronization were assumed for the simulations. This binary stream forming a coded block is sent to the BCH decoder where the coding and errors induced by the channel are removed. The input to this system is also used as an input to the statistical decision process.

2.1.7 BCH Decoder

The decoder being used for these codes locates and correct errors in the information, while removing the redundant information. Errors can only be corrected up to the level specific to the minimum distance (d) of each code. If this threshold is exceeded, the output contains an unknown amount of errors. The decoding is accomplished by using complicated mathematical techniques that reverse the encoding process while tolerating induced errors [6]. The type of decoding process used is called hard decision because it takes an input with

a definite value for each incoming bit, either a 0 or 1. Soft decision decoders take a bit stream that adds an character representing an unknown bit when a bit cannot be determined a 0 or 1. These soft decision decoders have better performance, but a more complex algorithm. Besides the corrected data, another important output of the decoder is the number of errors that were successfully corrected. If the error correction ability has been exceeded, this value is replaced by an error flag.

The MATLAB® language comes with functions that perform both BCH encoding and decoding. Since the functions are designed for very generalized applications, they were changed to handle the requirements of this specific system's input and output. The new functions are faster and optimized for this application.

The information at the data output will be the same as the data which was presented at the source if the error correcting code was properly implemented. However, during periods of heavy attenuation and noise, errors are induced that cannot be corrected.

2.2 Adaptive System

The adaptive system includes the decision process which uses the information from the receiving side of the system. The output from this process determines whether to increase, decrease, or maintain the level of redundancy currently in use. Two different algorithms for making this decision are examined.

2.2.1 Statistical Adaptive Decision Method

The statistical method of estimating the channel examines the incoming signal and

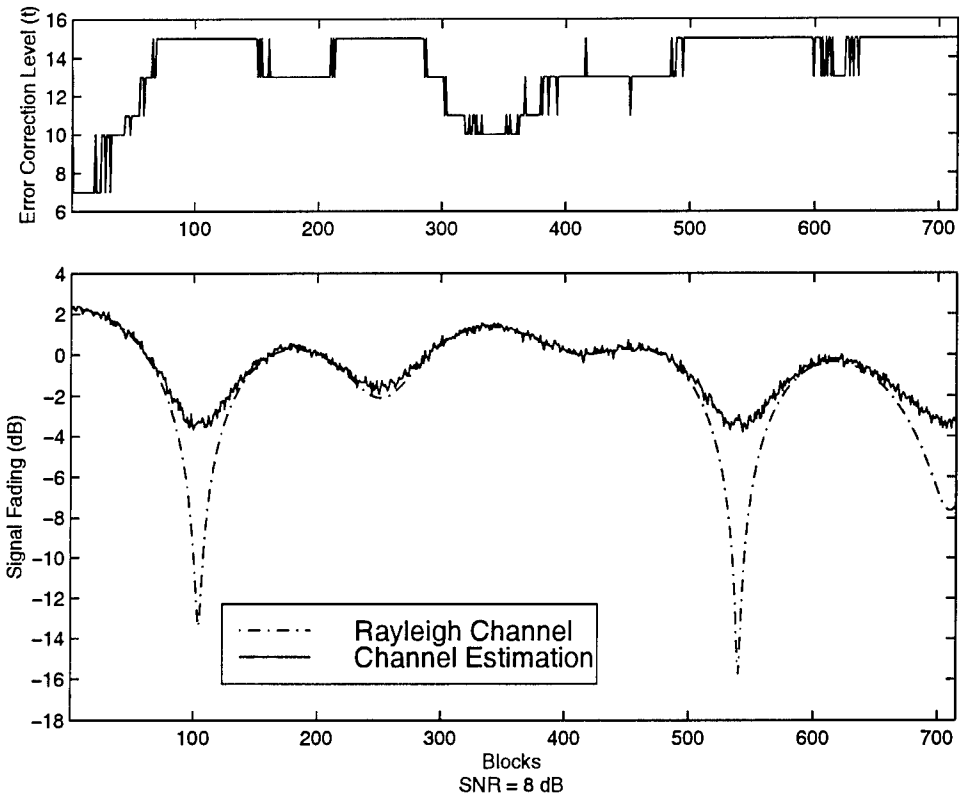

Figure 4. Statistical Adaptive Decision Method

performs statistical calculations using the signal mean (m_x), and the 2nd and 4th moments (E_2, E_4) to find the signal level ($\hat{m}$) after the channel fading. This computational method can be visualized as finding the new location of the 0 and 1 peaks in the probability distribution function (PDF) [2]. The equations below were implemented in MATLAB®.

$$E_{2,4} = \int_{-\infty}^{\infty} (x - m_x)^{2,4} p(x) dx \tag{2.1}$$

$$\hat{m} = \tfrac{3}{2}\sqrt[4]{E_2^2 - \tfrac{1}{2}E_4} \tag{2.2}$$

Figure 4 shows the Rayleigh channel and the estimation given by the statistical system. It is noted that the estimate deviates from the actual value during periods of high attenuation. This is due to the failure of the statistical estimator. Increasing amounts of uncertainty exist

when attenuation is high or when the SNR is low.

In order to adapt the coding in this first method, the system takes this estimated value of the channel conditions and enters a look-up table. This table lists what code is appropriate for a range of estimated conditions. This look-up table is generated by noting the dB level at which the BEP of each code crosses 10^{-4} in simulation. The BEP of 10^{-4} was arbitrarily chosen to represent a minimum acceptable error probability in voice transmissions.

2.2.2 Error Based Adaptive Decision Method

The second method's results show in Figure 5 the system reacting to the number of corrected errors per block. The error count output from the decoder varies with the attenuation of the Rayleigh channel. Instead of calculating a statistical value describing the

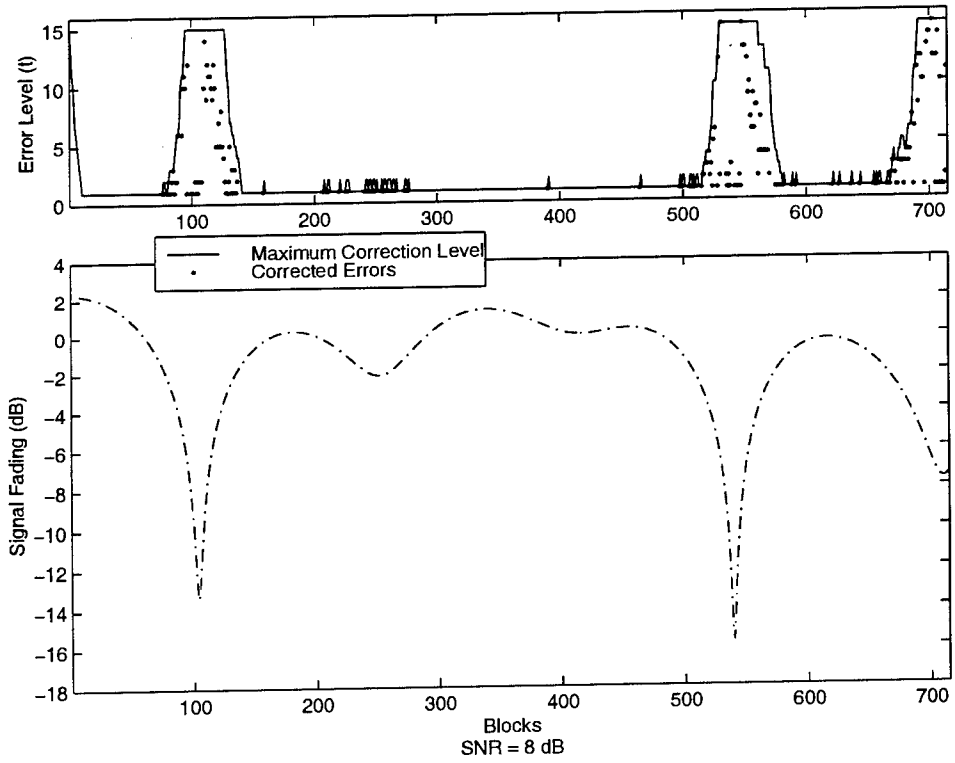

Figure 5. Error-Based Adaptive Decision Method

current channel, this decision algorithm looks at the errors corrected within the current block and compares this number to the current code's maximum correction capability. If the decoder corrected more than the upper threshold (75% of the maximum capability) for the current code (e.g., 5 errors when the code can only correct 6), the decision device will shift to a code with more redundancy. If the number of errors is below the lower threshold (25% of the maximum capability) the system will decrease the next code's redundancy.

In both Figures 4 and 5, the solid line in the upper block shows the maximum correction capability of the code used for each block. This changing level is an indicator of the working adaptive process. Higher levels of coding reduce the code's efficiency. It is noted that there is a large difference in coding efficiency between these two different methods. The error evaluation method changes due to the code's response to fading, and thus tightly fits the occurring errors. The statistical method tries to match an uncertain estimation of real world conditions to numbers found from a formula. The inherent differences between these values drive the system to less efficient behavior.

To communicate each adaptation decision, a message is sent via a low bandwidth feedback channel to the encoder. If small bursts of noise are creating excessive level changes, a smoothing factor can be added so that decisions are made from an average of multiple blocks. This reduction of changes reduces the load on the feedback channel, which may be limited in capacity by design constraints. During simulation, this feedback channel was assumed to be error-free and without delay. This allows necessary information about coding changes to be implemented before the next block is coded at the transmitter.

3. Procedure

In order to achieve the final working simulation of the entire adaptive system, individual sub-systems were developed and validated. Once each sub-system was checked, it was incorporated into the final system.

3.1 Simulation Technique

A Monte-Carlo approach was used for all simulations. This method works by sending data through the system until a statistically significant number of random events occur. The random events observed in communication simulations are transmission errors. As a greater amount of events occur, the probability of occurrence can be calculated with higher accuracy. The bit error probability (BEP) is calculated by dividing the total number of errors by the total number of bits that have be sent through the system. The problem with this method is that for very low error probabilities it takes a very long time to simulate. For example, six Pentium II - 350 MHz computers calculated error probabilities, continuously for twelve days and did not finish. These computers had to be stopped, and then restarted where they had left off. The excellent fit of data to expected values validates the correct output of the sub-systems. However, when data did not fit the expected results, various problems with the design were corrected.

3.2 Output Validation

To ensure that the simulation of the system was operating properly, the output of each

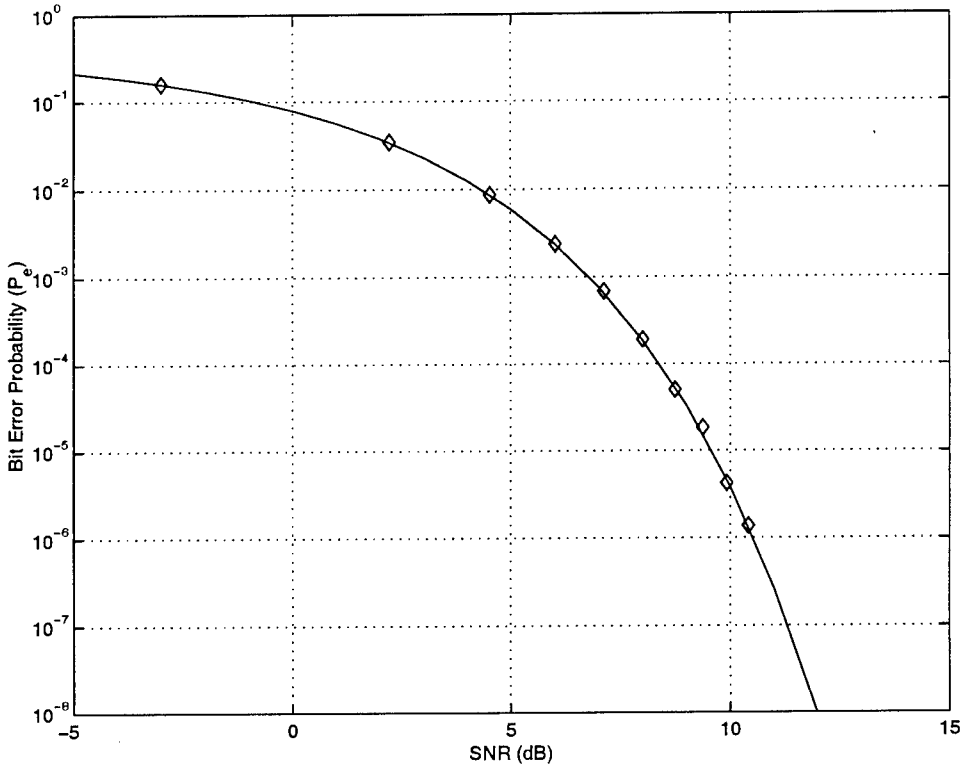

Figure 6. BPSK Waveform and AWGN

subsystem was checked against well-established equations. The data collected from Monte-Carlo simulations was plotted generating the "waterfall" curves of error probability (BEP) vs. signal to noise ratio (SNR) shown in Figures 6 through 8.

For a BPSK waveform in the presence of varying levels of AWGN, the expected probability of error (P_e) curve follows the function below and the graph comparing the experimental data to the closed form solution is shown in Figure 6 [7].

$$P_e = \text{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{3.1}$$

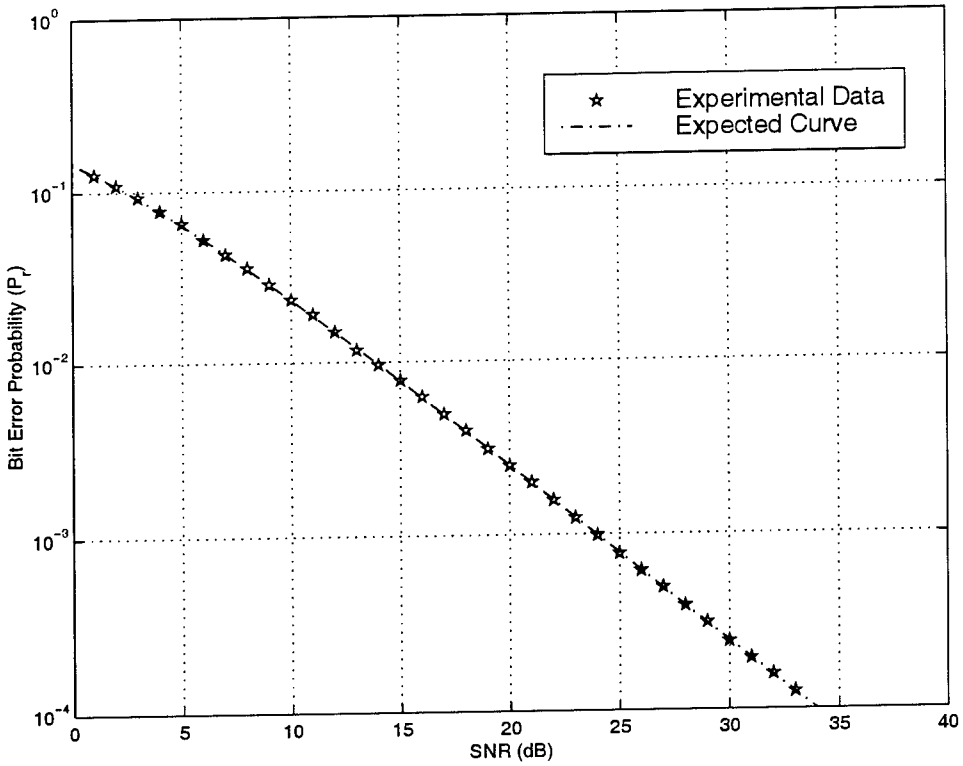

Figure 7. Rayleigh Channel and AWGN

The error probability of a BPSK signal in a Rayleigh fading channel (P_r) was tested in response to varying noise levels (SNR). The function plotted in Figure 7 has the form for a signal diversity of 1 (L=1), which simplifies greatly the generalized function below [5].

$$P_r = \left[\tfrac{1}{2}(1-\mu)\right]^L \sum_{k=0}^{L-1} \binom{L-1+k}{k} \left[\tfrac{1}{2}(1+\mu)\right]^k \tag{3.2}$$

$$\mu = \sqrt{\frac{\overline{\gamma}_c}{1+\overline{\gamma}_c}} \qquad \overline{\gamma}_c = \text{mean SNR} \tag{3.3}$$

Finally, the hard decoded BCH error curves (P_h) were plotted in Figure 8 against established error upper bound curves (P_{ub}) for comparison [6,8]. The equation used is stated below in Eq. 3.4. One curve is generated for each of the 11 valid (n,k) BCH code pairs of

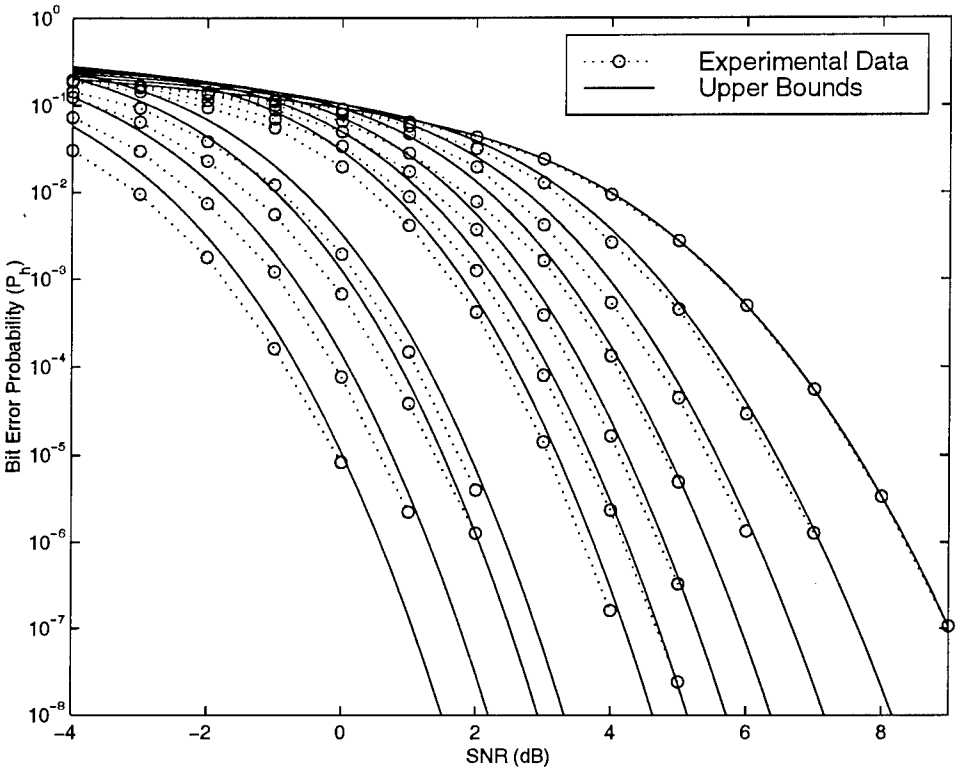

Figure 8. Family of BCH Codes: Length 63

length n=63. It is noted that the experimental data approaches the bounds at lower error rates.

$$P_b < P_{ub} = \frac{1}{n}\sum_{i=e+1}^{n}\binom{n}{i}(i+e)p^i(1-p)^{n-1} \qquad (3.4)$$

$$e = \frac{(d-1)}{2} \qquad p = \text{erfc}\left(\sqrt{\frac{2k}{n}\frac{E_b}{N_o}}\right)$$

n = block length

k = information length

d = minimum distance (Hamming)

These three figures (Figures 6-8) show that each system is properly implemented and outputting valid simulation results.

3.3 Experimental Results

Initially, the first simulations of the system used a non-realistic channel operating at a SNR 5 decibels higher than during final simulations. For this type of channel, the first decision method (statistical estimation) performed better. But once a realistic Rayleigh channel was implemented, the estimator's loss in performance forced the use of a second method. These two methods are contrasted through simulation.

The criteria used to compare the coding methods include efficiency, BEP, and a performance metric. Efficiency (η) is the total amount of information bits transmitted divided by the total number of bits in blocks (total number of blocks times 63). The bit error

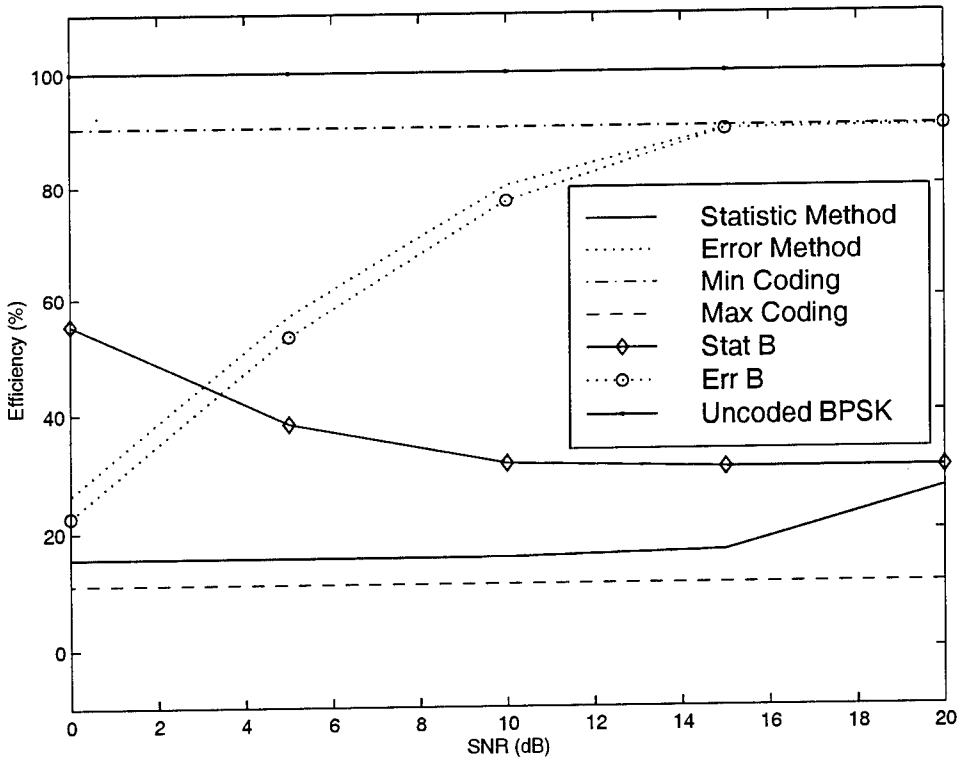

Figure 9. Efficiency (η) vs. SNR

probability (BEP) is the number of information bits received in error after decoding and correction, divided by the total number of bits of transmitted information. The performance metric(Ψ) allows both parameters to be combined and analyzed on the same graph.

$$\Psi = -\log_{10}(\text{BEP}) \cdot \eta\% \tag{3.5}$$

Figures 9 through 11 show the performance of both types of adaptive systems while changing the level of the AWGN encountered at the receiver, thus varying the SNR. Examples of the minimum and maximum levels of fixed coding are shown for reference. The BCH codes used for this simulation have a range of Eb/No in which they operate most efficiently. For the length 63 family of BCH codes, this range extends upward from -0.7 dB

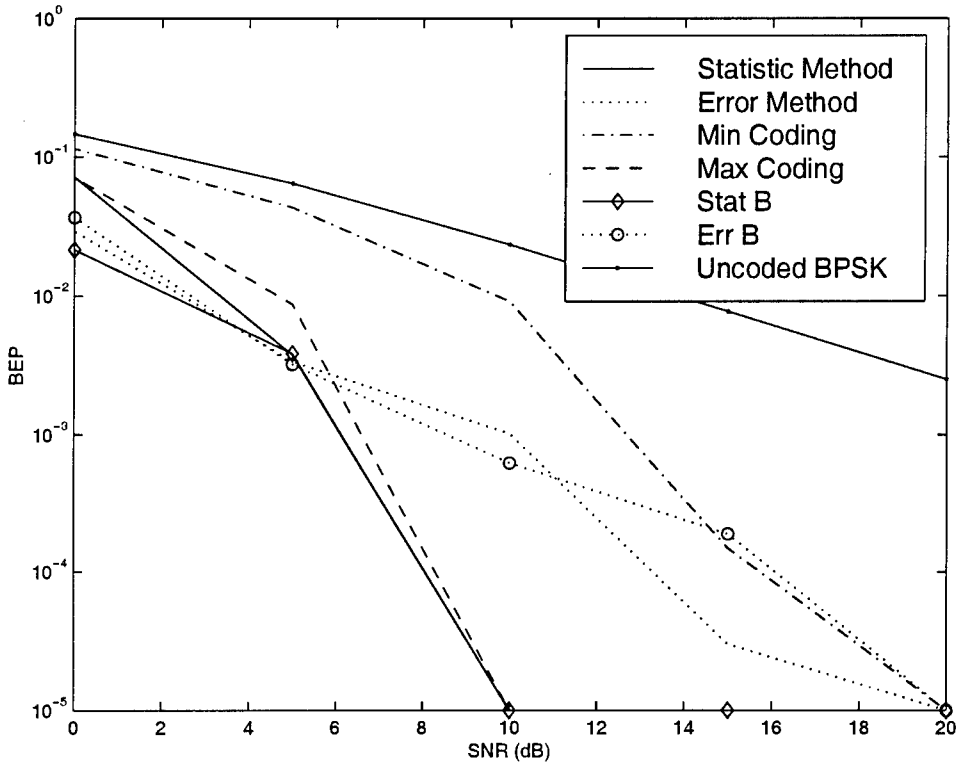

Figure 10. BEP vs. SNR

at a BEP of 10^{-4}. Both methods would perform better if the BCH codes could be extended to operate in a lower dB range.

The major problem encountered was that the BEP at low Eb/No was unacceptable, often approaching 10^{-1}. Rayleigh attenuation in addition to the large amounts of AWGN creates a quickly changing wireless environment. The adaptive system does not provide any benefit when encountering such conditions.

Two types of errors exist in this simulation: threshold errors and transition errors. Threshold errors occur when too many errors occur for the maximum level of redundancy. This occurs with fades below -5dB or during smaller fades when noise levels are high. The inability of the system to adjust quickly creates a transition error. The current code is then

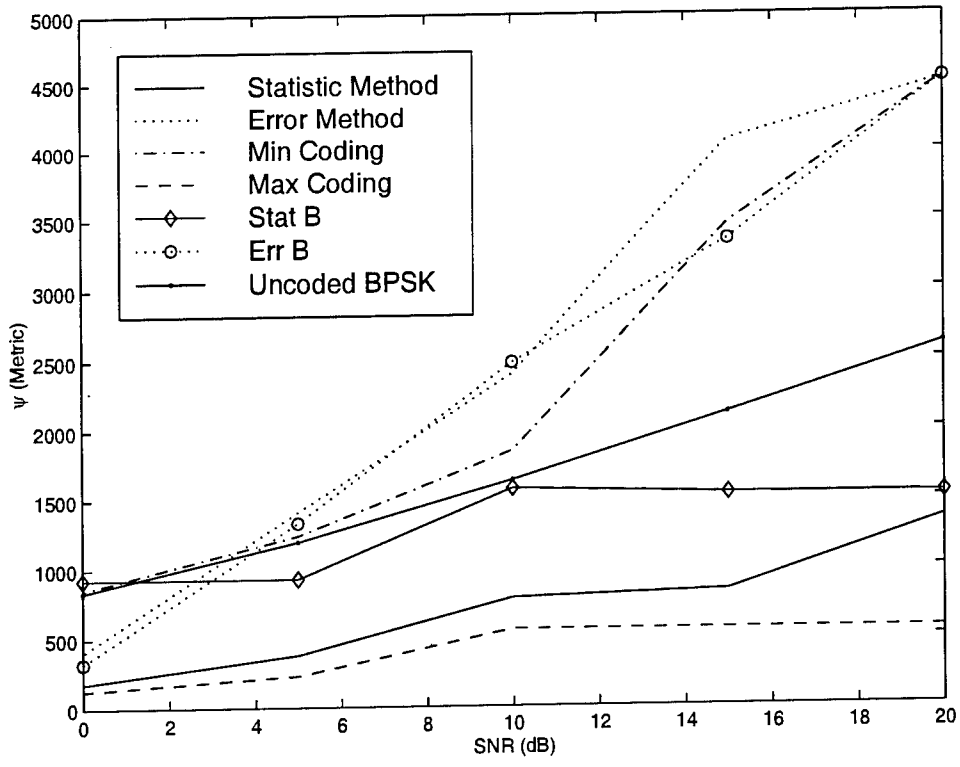

Figure 11. Metric vs. SNR

caught using less redundancy than necessary for the errors encountered. Both methods have a limitation as to how quickly they can react to a changing channel. They both only change one coding level at a time. Therefore, this simulation system is designed for slowly varying channels.

The Type B systems shown in the three figures above are attempts to improve the performance of the original adaptive systems. For "Stat B," the different levels in the look-up table were all shifted by a constant of -2.5 dB. This was an attempt to increase the system's efficiency and metric. In "Error B," the upper threshold level was moved from 75% to 50% of the maximum correction ability. This was an attempt to improve the adaptation response and reduce the system's BEP.

4. Conclusions

The goal of this research was to develop and implement an adaptive system that could efficiently and effectively balance throughput and data integrity. Overall, the statistical and error based adaptive methods show benefits compared to fixed coding, but the metric demonstrates the error evaluation method to be superior over a wide range of SNR. From 10 dB to 15 dB this adaptive system maintains a 32% gain over minimum fixed encoding. This method is only surpassed by the statistical type B method at very low SNR.

The benefits of this error-based decision method include simplicity and ruggedness. The adaptation decision is made from existing BCH decoder outputs and does not rely on further signal analysis, which makes the system simpler. The statistical method shows considerable problems, creating a channel estimate in severe conditions and giving higher than expected values. The efficiency is therefore increased at these low SNR levels, giving this method better metric values than the error method. The error method is rugged since severe noise and attenuation do not seriously affect the system's decision-making ability. The current problem with the error system is its lag in responding to quick changes of the channel, which lead to transition errors. This lag is due to the fact the redundancy is changed after evaluating the performance of the last block received, always keeping the system one block behind the current conditions.

The two Type B systems show that performance of the original designs can be modified in order to enhance certain characteristics. In future investigations, the performance needs to be maximized for the requirements of specific applications and their operational levels of SNR.

4.1 Challenges

A few difficulties were encountered while working on this project. Much time at the beginning of the project was spent getting the system designed and programmed properly. MATLAB® commands and the proper construction of a communications simulation had to be learned. The mathematics used in encoding and decoding BCH codes are very specialized and are extremely complex; these procedures took patience and diligence to understand. Once the system was first operational, the most noticeable problem encountered was with the decoder. When the error levels exceeded the code's capability, the encoder sent out unreliable information about the number of errors that were corrected. The error adaptation method then reduced the level of redundancy, further worsening the problem, causing catastrophic failure of the system. To correct this output, the core decoder function was modified to produce the proper output.

4.2 Future Work

With greater amount of time to complete further research, many related problems could be investigated. Keeping the system as designed, optimal parameters for the error evaluation system could be found. Another performance enhancement that could be implemented is the ability to adapt between codes more than one level apart.

The addition of other types of codes (Reed Solomon, Convolutional or Turbo) may lead to gain in system performance. Also an automatic request for repeat (ARQ) system could be implemented when the capability of the BCH code is exceeded and threshold errors

are occurring. This would ensure a much lower probability of error than shown in this simulation.

4.3 Future Applications

The implementation of this system in hardware is the next major application of this research. This could be done with Digital Signal Processors (DSP) or programmable gate arrays. It would be beneficial to see the entire system working outside of computer simulations. The low cost of modern hardware to perform the tasks of coding and adapting could make the computational overhead associated with such a system worthwhile.

With additional improvements, this adaptive system could be incorporated into innovative communication systems. Two such systems include a satellite link with rain attenuation or ship-to-ship data links in the Navy [4].

References

[1] Benice, R. J. "Adaptive Modulation and Error Control Techniques." IBM Corporation, 1966.

[2] Cress, D. E. and Ebel, W. J. "Turbo Code Implementation Issues for Low Latency, Low Power Applications." *Proceedings of the MPRG Symposium on Wireless Personal Communications*, June 10-12, 1998.

[3] Farrell, P.G. "Coding for Noisy Data Links.' Ph. D. dissertation. University of Cambridge, 1969.

[4] Ha, Tre. *Digital Satellite Communications.* New York: Macmillan Publishing Company, 1986.

[5] Lidl, Rudolf. *Introduction to Finite Fields and Their Applications.* Cambridge: Cambridge University Press, 1994.

[6] Lin, Shu. *Introduction to Error-correcting Codes.* Englewood Cliffs: Prentice Hall. 1970.

[7] McFalls, Frank, ATC, USN. Interview by author. Annapolis, Md., October 13, 1998.

[8] Proakis, John G. *Digital Communications*, Second Edition. McGraw-Hill Book Company, New York, 1989.

[9] Sklar, Bernard. *Digital Communications: Fundamentals and Applications.* Englewood Cliffs: Prentice Hall, 1988.

[10] Stremler, Ferrel G. *Communications Systems*, Third Edition. New York: Addison-Wesley Publishing Company, 1990.

[11] Weng, L.G. "Soft and Hard Decoding Performance Comparisons for BCH Codes," *Proceedings of the International Conference on Communications*, 1979.

Appendix

BCH Code Parameters

N	K	T	N	K	T	N	K	T
7	4	1	255	199	7	511	358	18
15	11	1		191	8		349	19
	7	2		187	9		340	20
	5	3		179	10		331	21
31	26	1		171	11		322	22
	21	2		163	12		313	23
	16	3		155	13		304	25
	11	5		147	14		295	26
	6	7		139	15		286	27
63	57	1		131	18		277	28
	51	2		123	19		268	29
	45	3		115	21		259	30
	39	4		107	22		250	31
	36	5		99	23		241	36
	30	6		91	25		238	37
	24	7		87	26		229	38
	18	10		79	27		220	39
	16	11		71	29		211	41
	10	13		63	30		202	42
	7	15		55	31		193	43
127	120	1		47	42		184	45
	113	2		45	43		175	46
	106	3		37	45		166	47
	99	4		29	47		157	51
	92	5		21	55		148	53
	85	6		13	59		139	54
	78	7		9	63		130	55
	71	9	511	502	1		121	58
	64	10		493	2		112	59
	57	11		484	3		103	61
	50	13		475	4		94	62
	43	14		466	5		85	63
	36	15		457	6		76	85
	29	21		448	7		67	87
	22	23		439	8		58	91
	15	27		430	9		49	93
	8	31		421	10		40	95
255	247	1		412	11		31	109
	239	2		403	12		28	111
	231	3		394	13		19	119
	223	4		385	14		10	121
	215	5		376	15			
	207	6		367	16			

N: code word length; K: message length; T: error-correction capability

MATLAB Files

```
% 'Reall.m'
% Real implementation of Raleigh Channel
% John Waterston
% February 21, 1999
% Trident Scholar Project
%

clear; clc;

%Code Parameters
n=63;
k_list=[7,10,16,18,24,30,36,39,45,51,57];
t_list=[15,13,11,10,7,6,5,4,3,2,1];

%Initial Parameters
  Nb=4;                                          % Samples per bit in the transmission channel
  klen=length(k_list); %Number of possible values of K; Limits values of adaption position
  x=1;                                           %Prepare for worst case scenario initially
%Create Data Source
  fprintf(1,'\n');                               % Output blank line
  source_len=1000*input('What is the length of the Data Source(in thousands)? ');
  smooth=input('What is the smoothing factor wanted? (1=no smoothing): ');
  randn('state',1);                  % Sets Random number generator to initial state
  source=randn(1,source_len)>0;
  datptr=1;                          % Pointer to position in the data array
  maxlen=ceil(source_len/k_list(1)); %Determine the number of blocks for fixed case
  minlen=ceil(source_len/k_list(klen));      %Rounds up with ceil function
%Initialize Variables
  c_errmax=zeros(1,maxlen);   % Errors that are corrected at the receiving end
  c_errmin=zeros(1,maxlen);
  c_err=zeros(1,maxlen);
  errorsmax=zeros(1,maxlen);  % Actual errors that occur between trans and recv
  errorsmin=zeros(1,maxlen);
  errors=zeros(1,maxlen);
  errmax=zeros(1,maxlen);     % Amount of errors detected by decoding process
  errmin=zeros(1,maxlen);
  err=zeros(1,maxlen);
  mem=zeros(1,5);             % Prepare storage of errors to calculate average
  mhat=zeros(1,maxlen);
  EbNo_est=zeros(1,maxlen);   % Values of Estimated SNR
  EbNo_avg=zeros(1,maxlen);   % Average of the SNR occuring over X number of blocks
  mem=zeros(1,smooth);        % Prepare storage of errors to calculate average
  uk=zeros(1,maxlen);         % Used values of k <- Information Bits
  ut=zeros(1,maxlen);         % Used values of t <- Correction Capability
  change=zeros(1,maxlen);     % Allows for count of level changes
  change(1)=1;                % Change = 1 to trigger change
  stop=0;                     % Shows when adaptive has stopped
% Determine the signal to noise parameters of the Channel
  limit=input('Do you want to limit fades below a level? (-5 dB) ');
  EbNo=input('At what EbNo level (db) do you want to transmit? ');
  G=0;
  %G=input('What amplifier gain in DB do you want? ');
  G=10.^(G/10);        %Convert from dB
  speed=input('What speed(km/hr) parameter do you want to use for the doppler
       parameter(5-100)? ');
  [vc,len]=varchan1(1.2*maxlen,speed,881,9600);      %Raleigh Channel from CDR Welch;
  vc=vc(1:maxlen);
  vc_lim=vc;
  for i=1:maxlen
     if 10*log10(vc(i)) < (limit)
        vc_lim(i)=10^(limit/10);
     end
  end

tic                                    % Begin Stopwatch for timing entire process
pg = bch_gen(n,k_list(1));             % Create Systematic Generator Matrix for MAX case
m  = length(pg) - 1;
b=[];
for i = 0:k_list(1)-1
```

```
   [q, tmp] = gfdeconv([zeros(1, n-(k_list(1))+i), 1], pg); % tmp is remainder of division
   tmp = [tmp zeros(1, m -length(tmp))];
   b=[b; tmp];
end;
gmmax = [b eye(k_list(1))];              % Multiply by the identity matrix

pg = bch_gen(n,k_list(klen));            % Create Systematic Generator Matrix for MIN case
m  = length(pg) - 1;
b=[];
for i = 0:(k_list(klen)) - 1
   [q, tmp] = gfdeconv([zeros(1, n-(k_list(klen))+i), 1], pg);
   tmp = [tmp zeros(1, m -length(tmp))];
   b=[b; tmp];
end;
gmmin = [b eye(k_list(klen))];                    % Multiply by the identity matrix

   % Transmitter transmits at a constant Ec/No(Channel Bit Energy)
   % Attenuation of the signal changes the effect of noise level.
   %sigma=sqrt((A^2*Nb)/(2*EbNo))
   sigma=sqrt(Nb./(2*10.^(EbNo/10)));             % RMS Voltage of Noise7

                                                  % Begin outside loop
for z=1:maxlen
   if change(z) == 1          % Change is non-zero when asking for a change; usually +/- 1
      k=k_list(x);            % Define current code parameters
      t=t_list(x);
      pg = bch_gen(n,k);      % Create Systematic Generator Matrix for adaptive case
      m  = length(pg) - 1;
      b=[];
      for i = 0:k - 1
         [q, tmp] = gfdeconv([zeros(1, n-k+i), 1], pg);
         tmp = [tmp zeros(1, m -length(tmp))];
         b=[b; tmp];
      end;
      gm = [b eye(k)];        % Multiply by the identity matrix
   end;

   uk(z)=k;                   % Record Current values of k,t
   ut(z)=t;
   data1=getdata(source,datptr,k);                % Get Data for block from data source
   data1max=getdata(source,((z-1)*k_list(1))+1,k_list(1));
   data1min=getdata(source,((z-1)*k_list(klen))+1,k_list(klen));
   datptr=datptr+k;                               % Advance pointer in data

   if (datptr > source_len)&(stop==0)             % Acknowledge end of adaptive transmission
      stop=z;
   end;

   codemax = rem(data1max * gmmax,2);             % Encode Data with BCH
   codemin = rem(data1min * gmmin,2);
   code = rem(data1 * gm,2);

   a_codemax=(2*codemax-1).*vc_lim(z);            % Make BPSK(antipodal) +/- 1 Volt and*Channel
   a_codemin=(2*codemin-1).*vc_lim(z);
   a_code=(2*code-1).*vc_lim(z);

   noise=sigma*randn(Nb,n);                       %Create AWGN dependant on EbNo of signal

   chanmax=((a_codemax'*ones(1,Nb))'+noise)*G;    % Combine Noise and coded data
   chanmin=((a_codemin'*ones(1,Nb))'+noise)*G;
   channel=((a_code'*ones(1,Nb))'+noise)*G;

   % Create Scaled PDF Histogram
   l=length(channel);
   vmax=max(max(channel));                        % Voltage Max in mV
   vmin=min(min(channel));                        % Voltage Min in mV
   a=511/(vmax-vmin);                             % 1/(Bin Size)
   hg=zeros(1,512);                               % Define Histogram Array and initialize with zeros
   y=fix((channel-vmin)*a)+1;
   step=1/(size(channel,1)*size(channel,2));

   for i=1:size(channel,1)
      for j=1:size(channel,2)
         hg(y(i,j))=hg(y(i,j))+step;
```

```
   end;
end;

     v=vmin:((vmax-vmin)/511):vmax;
     E=sum(v.*hg);                                    % Mean - Expected Value
     m2=sum((v-E).^2.*hg);                            % Second Moment -
     m4=sum((v-E).^4.*hg);                            % Fourth Moment
     mhat=((3/2)*(m2)^2-(1/2)*m4)^(1/4);
%sigma2=sqrt(abs((m2)-mhat(z)^2));
if (3*m2^2>m4)|(m2>mhat^2)
   mhat=sum(abs(v).*hg);
end;

%Noise_est(z)=10*log10(Nb/(2*sigma2^2));  % in dB
EbNo_est(z)=10*log10(mhat);               % in dB
mem(rem(z,smooth)+1)=EbNo_est(z);         % Rotate through memory block
EbNo_avg(z)=sum(mem)/smooth;              % Calculate current amount of average errors

%plot(v,hg);

recvmax=sum(chanmax)>0;                   % Receive Data with an integrator detector
recvmin=sum(chanmin)>0;
recv=sum(channel)>0;

     % Decode received signal MAX
[data2max,ccodemax,errmax(z)]=debch3(recvmax,n,k_list(1),t_list(1));
c_errmax(z)=sum(abs(recvmax-ccodemax));          % Receiver Compare
errorsmax(z)=sum(abs(data2max-data1max));        % Overall Compare
     % Decode received signal MIN
[data2min,ccodemin,errmin(z)]=debch3(recvmin,n,k_list(klen),t_list(klen));
c_errmin(z)=sum(abs(recvmin-ccodemin));          % Receiver Compare
errorsmin(z)=sum(abs(data2min-data1min));        % Overall Compare
     % Decode received adaptive signal
[data2,ccode,err(z)]=debch3(recv,n,k,t);
c_err(z)=sum(abs(recv-ccode));            % Receiver Compare (possible in real world)
errors(z)=sum(abs(data2-data1));          % Overall Compare (used for evaluation)

     %SNR Adaption Method
adj=0;  %for 10-4 PE case

x_old=x;

if EbNo_avg(z) >= 6.8 +adj
  x=11;
elseif EbNo_avg(z) >= 5.7 +adj
  x=10;
elseif EbNo_avg(z) >= 4.8 +adj
  x=9;
elseif EbNo_avg(z) >= 4.0 +adj
  x=8;
elseif EbNo_avg(z) >= 3.5 +adj
  x=7;
elseif EbNo_avg(z) >= 2.8 +adj
  x=6;
elseif EbNo_avg(z) >= 2.0 +adj
  x=5;
elseif EbNo_avg(z) >= 1.2 +adj
  x=4;
elseif EbNo_avg(z) >= .65 +adj
  x=3;
elseif EbNo_avg(z) >= -.3 +adj
  x=2;
else
  x=1;
end;
if x ~= x_old
   change(z+1)=1;
end
if mod(z,20) == 0                 % Output number to show progress every 100 blocks
  disp(z);
end;
end;                              % End Main Loop
et=toc;                     % End Stopwatch
```

```
%Performance Calculation Phase
Nu=sum(uk(1:stop))/(n*stop);
Nu_total=(sum(uk))/(n*maxlen);
f=diff(10*log10(vc));
  clc
  disp(['DATA SHEET / PERF1.M / ', datestr(now)]);
  disp(' ');
  disp(['Efficiency of adaptive code =',num2str(Nu),'  The total efficiency =
',num2str(Nu_total)]);
  disp(['Total K bits = ',num2str(sum(uk))]);
  disp(['Total T bits = ',num2str(sum(ut))]);
  disp(['The adaptive process transmitted ',num2str(source_len),' bits in
',num2str(stop),' blocks.']);
  disp(['The number of times the code was changed is ',num2str(sum(change)-1),' times
overall and ',num2str(sum(change(1:stop))-1), ' times while transmitting data']);
  disp(['The Average of the channel is ',num2str(sum(vc)/length(vc)),' dB and
',num2str(sum(vc(1:stop))/stop),' dB before stopping']);
  disp(['The Average Rate of the channel is ',num2str(sum(abs(f))/length(f)),' dB/block
and ',num2str(sum(abs(f))/(stop-1)),' dB/block before stopping']);
  disp(['The number of blocks transmitted with EbNo < -0.3 dB =
',num2str(length(find(vc<-0.3))),' Before stopping =
',num2str(length(find(vc(1:stop)<-0.3)))]);
  disp(['The speed used in the Raleigh Channel is ',num2str(speed),' km/hr.']);

  %Performance Output Phase
  %Process Graphing - Design and Run Function to allow changes after running program.
figure(1)
subplot(3,1,[1 2 3])
plot(10*log10(vc_lim),'r-.');
hold on;
plot(EbNo_avg,'g-');
plot(ut,'k-');
axis([1 maxlen -20 16]);
hold off;
%title({'John Waterston - Trident Project',['reall.m /',datestr(now)]});
xlabel({['Blocks'],['EbNo = ',num2str(EbNo)]});
%xlabel({['maxlen=',num2str(maxlen),' minlen=',num2str(minlen),'
stop=',num2str(stop)],['Total K bits = ',num2str(sum(uk)),'  Total T bits =
',num2str(sum(ut))]});
ylabel(['Channel Attenuation (dB) & Error Correction Level']);
legend('Rayleigh Channel','Channel Estimation','Coding Level')

% Efficency and Elapsed Time and Smoothing TEXT
%gtext({['\eta = ',num2str(Nu),'  \eta Total=',num2str(Nu_total)],['Elapsed
Time=',num2str(et),'  Smoothing Factor=',num2str(smooth)]});
%gtext({['\eta = ',num2str(Nu_total)],'  Smoothing Factor=',num2str(smooth)]});

BEP=',num2str(sum(errorsmin(1:minlen))/(minlen*k_list(klen)))],['Adaptive
BEP=',num2str(sum(errors(1:stop))/sum(uk(1:stop))),'  Overall
BEP=',num2str(sum(errors)/sum(uk))]});

%gtext({['The adaptive process transmitted ',num2str(source_len),' bits in
',num2str(stop),' blocks.'],...
%       ['The code changed ',num2str(sum(change)-1),' times overall and
',num2str(sum(change(1:stop))-1), ' times before stopping'],...
%       ['The avg EbNo of the channel is ',num2str(sum(10*log10(vc))/length(vc)),' dB and
',num2str(sum(10*log10(vc(1:stop)))/stop),' dB before stopping'],...
%       ['The avg EbNo Rate of the channel is ',num2str(sum(abs(f))/length(f)),' dB/blk and
',num2str(sum(abs(f))/(stop-1)),' dB/blk before stopping'],...
%       ['The number of blocks transmitted with EbNo < -0.3
dB=',num2str(length(find(vc<10^(-0.3/10)))),' Before stopping
=',num2str(length(find(vc(1:stop)<10^(-0.3/10))))],...
%       ['The speed used in the Raleigh Channel is ',num2str(speed),' km/hr.  The limit was
set at ', num2str(limit),' dB.']});
```

```
% 'Real2.m'
% Real implementation of Raleigh Channel
% Error Decision process
% John Waterston
% March 19, 1999
% Trident Scholar Project
%

clear; clc;

%Code Parameters
n=63;
k_list=[7,10,16,18,24,30,36,39,45,51,57];
t_list=[15,13,11,10,7,6,5,4,3,2,1];

%Initial Parameters
Nb=4;                                   % Samples per bit in the transmission channel
klen=length(k_list);                    % Number of possible values of K;
x=1;                                    % Prepare for worst case scenario initially
%Create Data Source
fprintf(1,'\n');                        % Output blank line
source_len=1000*input('What is the length of the Data Source(in thousands)? ');
smooth=input('What is the smoothing factor wanted? (1=no smoothing): ');
randn('state',1);                       % Sets Random number generator to initial state
source=randn(1,source_len)>0;
datptr=1;                               % Pointer to position in the data array
maxlen=ceil(source_len/k_list(1));      % Determine the number of blocks for fixed case
minlen=ceil(source_len/k_list(klen))    % Rounds up with ceil function
%Initialize Variables
c_errmax=zeros(1,maxlen);               % Errors that are corrected at the receiving end
c_errmin=zeros(1,maxlen);
c_err=zeros(1,maxlen);
errorsmax=zeros(1,maxlen);              % Actual errors that occur between trans and recv
errorsmin=zeros(1,maxlen);
errors=zeros(1,maxlen);
errmax=zeros(1,maxlen);                 % Ammount of errors detected by decoding process
errmin=zeros(1,maxlen);
err=zeros(1,maxlen);
mem=zeros(1,5);                         % Prepare storage of errors to calculate average
mhat=zeros(1,maxlen);
EbNo_est=zeros(1,maxlen);               % Values of Estimated SNR
Err_avg=zeros(1,maxlen);                % Average of the SNR occuring over X # of blocks
mem=zeros(1,smooth);                    % Prepare storage of errors to calculate average
uk=zeros(1,maxlen);                     % Used values of k <- Information Bits
ut=zeros(1,maxlen);                     % Used values of t <- Correction Capability
change=zeros(1,maxlen);                 % Allows for count of level changes
change(1)=1;                            % Change = 1 to trigger change
stop=0;                                 % Shows when adaptive has stopped

%Determine the signal to noise parameters of the Channel
limit=input('Do you want to limit fades below a level? (-5 dB) ');
EbNo=input('At what EbNo level (db) do you want to transmit? ');
G=0;
%G=input('What amplifier gain in DB do you want? ');
G=10.^(G/10);  %Convert from dB
speed=input('What speed(km/hr) parameter do you want to use for the doppler
parameter(5-100)? ');
[vc,len]=varchan1(1.2*maxlen,speed,881,9600);       %Raleigh Channel ((kph), (kHz), baud)
vc=vc(1:maxlen);
vc_lim=vc;
for i=1:maxlen
   if 10*log10(vc(i)) < (limit)
      vc_lim(i)=10^(limit/10);
   end
end

tic                                     % Begin Stopwatch for timing entire process

pg = bch_gen(n,k_list(1));              % Create Systematic Generator Matrix for MAX
case
m  = length(pg) - 1;
b=[];
for i = 0:k_list(1)-1
```

```
   [q, tmp] = gfdeconv([zeros(1, n-(k_list(1))+i), 1], pg); % tmp is remainder of division
   tmp = [tmp zeros(1, m -length(tmp))];
   b=[b; tmp];
end;
gmmax = [b eye(k_list(1))];          % Multiply by the identity matrix
pg = bch_gen(n,k_list(klen));        % Create Systematic Generator Matrix for MIN case
m  = length(pg) - 1;
b=[];
for i = 0:(k_list(klen)) - 1
   [q, tmp] = gfdeconv([zeros(1, n-(k_list(klen))+i), 1], pg);
   tmp = [tmp zeros(1, m -length(tmp))];
   b=[b; tmp];
end;
gmmin = [b eye(k_list(klen))];       % Multiply by the identity matrix

% Transmitter transmits at a constant Ec/No(Channel Bit Energy)
% Attenuation of the signal changes the effect of noise level.
%sigma=sqrt((A^2*Nb)/(2*EbNo))
sigma=sqrt(Nb./(2*10.^(EbNo/10)));   % RMS Voltage of Noise7

for z=1:maxlen                       % Begin outside loop
   if change(z) == 1                 % Change is non-zero when asking for a change;
      k=k_list(x);                   % Define current code parameters
      t=t_list(x);
      pg = bch_gen(n,k);      % Create Systematic Generator Matrix for adaptive case
      m  = length(pg) - 1;
      b=[];
      for i = 0:k - 1
         [q, tmp] = gfdeconv([zeros(1, n-k+i), 1], pg);
         tmp = [tmp zeros(1, m -length(tmp))];
         b=[b; tmp];
      end;
      gm = [b eye(k)];               % Multiply by the identity matrix
   end;

   uk(z)=k;                          % Record Current values of k,t
   ut(z)=t;

   data1=getdata(source,datptr,k);   % Get Data for block from data source
   data1max=getdata(source,((z-1)*k_list(1))+1,k_list(1));
   data1min=getdata(source,((z-1)*k_list(klen))+1,k_list(klen));

   datptr=datptr+k;                  % Advance pointer in data

   if (datptr > source_len)&(stop==0)% Acknowledge end of adaptive transmission
      stop=z;
   end;

   codemax = rem(data1max * gmmax,2);        % Encode Data with BCH
   codemin = rem(data1min * gmmin,2);
   code = rem(data1 * gm,2);

   a_codemax=(2*codemax-1).*vc_lim(z);       % Make BPSK(antipodal) +/- 1 Volt * channel
   a_codemin=(2*codemin-1).*vc_lim(z);
   a_code=(2*code-1).*vc_lim(z);

   noise=sigma*randn(Nb,n);                  % Create AWGN dependant on EbNo of signal

   chanmax=((a_codemax'*ones(1,Nb))'+noise)*G;       % Combine Noise and coded data
   chanmin=((a_codemin'*ones(1,Nb))'+noise)*G;
   channel=((a_code'*ones(1,Nb))'+noise)*G;

   recvmax=sum(chanmax)>0;           % Receive Data with an integrator type dectector
   recvmin=sum(chanmin)>0;
   recv=sum(channel)>0;

       % Decode received signal MAX
   [data2max,ccodemax,errmax(z)]=debch3(recvmax,n,k_list(1),t_list(1));
   c_errmax(z)=sum(abs(recvmax-ccodemax));          % Receiver Compare
   errorsmax(z)=sum(abs(data2max-data1max));        % Overall Compare
       % Decode received signal MIN
   [data2min,ccodemin,errmin(z)]=debch3(recvmin,n,k_list(klen),t_list(klen));
   c_errmin(z)=sum(abs(recvmin-ccodemin));          % Receiver Compare
```

```
    errorsmin(z)=sum(abs(data2min-data1min));        % Overall Compare
        % Decode received adaptive signal
    [data2,ccode,err(z)]=debch3(recv,n,k,t);
    c_err(z)=sum(abs(recv-ccode));            % Receiver Compare (possible in real world)
    errors(z)=sum(abs(data2-data1));          % Overall Compare (used for evaluation)

    if err(z) == -1
       err(z)= t;
    end

    mem(rem(z,smooth)+1)=err(z);              % Rotate through memory block replacing old
                                              error values
    Err_avg(z)=ceil(sum(mem)/smooth);
    BEPguess=sum(errors)./sum(uk);

    x_old=x;

    change(z+1)=0;
    %75%
    if Err_avg(z) > .35*t_list(x);
       x=x-1;
       change(z+1)=1;
       if x < 1
          x=1;
       end
    elseif Err_avg(z) < .25*t_list(x);
       x=x+1;
       change(z+1)=1;
       if x > length(k_list)
          x=length(k_list);
       end
    end

    if mod(z,20) == 0                         % Output number to show progress every 100
blocks
       disp(z);
       disp(BEPguess);
    end;

end;                                          % End Main Loop
et=toc;                                       % End Stopwatch

%Performance Calculation Phase
Nu=sum(uk(1:stop))/(n*stop);
Nu_total=(sum(uk))/(n*maxlen);
f=diff(10*log10(vc));
clc
disp(['DATA SHEET / PERF1.M / ', datestr(now)]);
disp(' ');
disp(['Efficiency of adaptive code =',num2str(Nu),'  The total efficiency =
',num2str(Nu_total)]);
disp(['Total K bits = ',num2str(sum(uk))]);
disp(['Total T bits = ',num2str(sum(ut))]);
disp(['The adaptive process transmitted ',num2str(source_len),' bits in ',num2str(stop),'
blocks.']);
disp(['The number of times the code was changed is ',num2str(sum(change)-1),' times
overall and ',num2str(sum(change(1:stop))-1), ' times while transmitting data']);
disp(['The Average of the channel is ',num2str(sum(vc)/length(vc)),' dB and
',num2str(sum(vc(1:stop))/stop),' dB before stopping']);
disp(['The Average Rate of the channel is ',num2str(sum(abs(f))/length(f)),' dB/block and
',num2str(sum(abs(f))/(stop-1)),' dB/block before stopping']);
disp(['The number of blocks transmitted with EbNo < -0.3 dB =
',num2str(length(find(vc<-0.3))),' Before stopping =
',num2str(length(find(vc(1:stop)<-0.3)))]);
disp(['The speed used in the Raleigh Channel is ',num2str(speed),' km/hr.']);

%Performance Output Phase
%Process Graphing - Design and Run Function to allow changes after running program.
figure(1)
subplot(3,1,[1 2 3])
%for i=1:maxlen
%     if errors(i) ~= 0
%       stem(i,errors(i),'filled');
```

```
%       hold on;
%    end;
% end;
%plot(10*log10(vc),'r:');

plot(10*log10(vc_lim),'r-.');
hold on;
plot(Err_avg,'g.');
plot(ut,'k-');
%plot(err,'d');

axis([1 maxlen -20 16]);
hold off;

%title({'John Waterston - Trident Project',['real1.m /',datestr(now)]});
%xlabel({['maxlen=',num2str(maxlen),' minlen=',num2str(minlen),'
stop=',num2str(stop)],['Total K bits = ',num2str(sum(uk)),'  Total T bits =
',num2str(sum(ut))]});
xlabel({['Blocks'],['EbNo = ',num2str(EbNo)]});
ylabel(['Channel (dB) & Error Correction Level']);
legend('Rayleigh Channel','Errors Corrected','Coding Level')

% Efficency and Elapsed Time and Smoothing TEXT
%gtext({['\eta = ',num2str(Nu),'  \eta Total=',num2str(Nu_total)],['Elapsed
Time=',num2str(et),'  Smoothing Factor=',num2str(smooth)]});

% BEP TEXT
%gtext({['Max BEP=',num2str(sum(errorsmax(1:maxlen))/(maxlen*k_list(1))),'  Min
BEP=',num2str(sum(errorsmin(1:minlen))/(minlen*k_list(klen)))],['Adaptive
BEP=',num2str(sum(errors(1:stop))/sum(uk(1:stop))),'  Overall
BEP=',num2str(sum(errors)/sum(uk))]});

%gtext({['The adaptive process transmitted ',num2str(source_len),' bits in
',num2str(stop),' blocks.'],...
%      ['The code changed ',num2str(sum(change)-1),' times overall and
',num2str(sum(change(1:stop))-1),' times before stopping'],...
%      ['The avg EbNo of the channel is ',num2str(sum(10*log10(vc))/length(vc)),' dB and
',num2str(sum(10*log10(vc(1:stop)))/stop),' dB before stopping'],...
%      ['The avg EbNo Rate of the channel is ',num2str(sum(abs(f))/length(f)),' dB/blk and
',num2str(sum(abs(f))/(stop-1)),' dB/blk before stopping'],...
%      ['The number of blocks transmitted with EbNo < -0.3
dB=',num2str(length(find(vc<10^(-0.3/10)))),' Before stopping
=',num2str(length(find(vc(1:stop)<10^(-0.3/10))))],...
%      ['The speed used in the Raleigh Channel is ',num2str(speed),' km/hr.  The limit was
set at ', num2str(limit),' dB.']});
```

```
function [b, a, gN, fd, L, fc] = raychan(vkph, fc, Rs, fs)
%       [b, a, gN, fd, L, fc] = raychan(vkph, fc, Rs, fs): Designs a
%       channel noise filter to have the normalized doppler frequency fd.
%       The inputs are:
%              vkph = vehicle velocity in kilometers per hour
%                fc = carrier frequency in Mhz
%                Rs = symbol rate
%                fs = samples per symbol
%       The vectors b and a contain the direct form filter
%       coefficients. A simple fixed-point algorithm is used to
%       find an fc value that gives the desired fd. gN is the
%       noise scaling factor used to scale the noise variance at
%       the filter output to unity via z[n] = sqrt(gN)*y[n].
%       by Mark A. Wickert 7/94.

%************************************************************
% Begin by convertiing the input parameters to the normalized doppler, fd:
fd = vkph*1e3/60^2/(fs*Rs)*fc*1e6/3e8;

% The noise filter cutoff frequency will be very close to fd, but
% at slow vehicle velocity may be too small for filter realization.
% To compensate for this a lower fd cuttoff of 0.025 is set. Interpolation
% (upsampling) by the factor L is invoked as needed to keep fc_up >= 0.25.
% This is equivalent to increasing the vehicle velocity by factor L, thus
% the filtered noise sequences must be 'slowed' back down by using an interpolation
% routing such as MATLABs interp1() or the DSP tool box function interp().
%************************************************************

% Find fd_up, the upsampled normalized doppler frequency.
L = 1;
fd_up = fd;
while fd_up < 0.025,
       L = L + 1;
       fd_up = L*fd;
end;
%disp('Normalized doppler and upsampled doppler:')
%disp(['fd = ' num2str(fd) ', fd_up = ' num2str(fd_up) ', L = ' num2str(L)])
%disp('Filter cutoff frequency search status:')
%************************************************************
% Design the noise filter
%tol = 0.0005;
tol = .02 * fd_up; % 2% of fd_up tolerance
trials = 1;
% initial guess for fc is just the normalized doppler itself
p0 = fd_up;
fd_hat = doppler(p0);
% display iteration status at start:
%disp(['trial #' num2str(trials) ', fc_hat = ' num2str(p0) ...
%       ', fd_hat = ', num2str(fd_hat) ', error = ' num2str(abs(fd_up-fd_hat))])
p = p0 - fd_hat + fd_up;

% search for fc value - no checks for not converging
% included at present.
while abs(p - p0) > tol,
       % try a new value
       trials = trials + 1;
       p0 = p;
       fd_hat = doppler(p0);
%       display iteration status at each trial:
%       disp(['trial #' num2str(trials) ', fc_hat = ' num2str(p0) ...
%       ', fd_hat = ', num2str(fd_hat) ', error = ' num2str(abs(fd_up-fd_hat))])
       p = p0 - fd_hat + fd_up;
end;
%************************************************************
% Obtain final design filter coefficients
[b,a] = chan_mod(p0,10);
fc = p0;
% Solve for the noise (power) scaling factor
gN = 1/(2*quad8('smom0',0,.5,[],[],b,a));
%************************************************************
end;
```

```
function [rwarm,Zf1,Zf2] = mk_warm(b, a, gN, L, N)
%        [rwarm,Zf1,Zf2] = mk_warm(b, a, gN, L, N)
%       Used to warm-up the Rayleigh channel
%       (i.e. runs enough noise through the filters to get
%       past the initial transients)
%       b,a    = Rayleigh filter coefficients from raychan.m
%       gN     = noise scaling factor from raychan.m
%       L      = interpolation factor L from raychan.m
%       N      = # samples (typically ~ Ns*2000)to get transients out
%       rwarm  = Rayleigh weights
%       Zf1    = final condition of the filter for use in mk_wgts.m
%       Zf2    = final condition of the filter for use in mk_wgts.m

% find required length of noise vector prior to interpolation
nx = ceil(N/L)+1;

%disp('Generating white sequence....')
% create complex noise vector for input to filter
x = randn(nx,1) + j*randn(nx,1);pause(1)

%disp('Filtering the white sequence....')
% filter and gain scale to unity variance the complex white noise
[x,Zf1] = filter(b,a,x);
x=x*sqrt(gN/2);
%disp('Upsampling using linear interpolation..........')
% Begin by placing L-1 zeros between each point of x
ny = L*nx;
rwarm = zeros(1,ny);
rwarm(1:L:ny) = x;
% Now filter with h[n] = 1 - |n|/L for |n| <= L-1
nh = (1:L)/L;
h = [nh fliplr(nh(1:L-1))];
[rwarm,Zf2] = filter(h,1,rwarm);
%       trim length for non integer N/L values
rwarm = rwarm(1:fix(N));
```

```
function [rw,Zf1,Zf2] = mk_wgts1(b, a, gN, L, N, zf1, zf2)
%        [rw,Zf1,Zf2] = mk_wgts1(b, a, gN, L, N, zf1, zf2)
%       Creates a complex vector of length N containing Rayleigh
%       channel weights for the channel noise filter
%       coefficients b and a and interpolation factor L.
%       Zf1 & Zf2 are the final filter states originally from mk_warm.m
%       The variance of yy = {wI + j wQ} is one,
%       thus in a comm channel application no additional scaling
%       by sqrt(2) is required.
%       by Mark Wickert 8/94.

% find required length of noise vector prior to interpolation
%nx = ceil(N/L) +1;
nx = round(N/L);

%disp('Generating Rayleigh channel weights....')
% create complex noise vector for input to filter
x = randn(nx,1) + j*randn(nx,1);


% filter and gain scale to unity variance the complex white noise
[x,Zf1] = filter(b,a,x,zf1);
x=x*sqrt(gN/2);


% Begin by placing L-1 zeros between each point of x
ny = L*nx;
rw = zeros(1,ny);
rw(1:L:ny) = x;
clear x

% Now filter with h[n] = 1 - |n|/L for |n| <= L-1
nh = (1:L)/L;
h = [nh fliplr(nh(1:L-1))];
clear nh
[rw,Zf2] = filter(h,1,rw,zf2);

%       trim length for non integer N/L values
%rw = rw(1:N);
```

```
function fd=doppler(fc)

% fd = doppler(fc): returns the normalized doppler frequency
% for an input filter cutoff frequency, fc. Filter parameters
% are fixed at 10 dB ripple using the function chan_mod.m. Numerical
% integration is performed using the MATLAB function quad8().
% by Mark A. Wickert 7/94.

[b,a]=chan_mod(fc,10);

fd=sqrt(2*quad8('smom2',0,.5,[],[],b,a)/quad8('smom0',0,.5,[],[],b,a));

end;
```

```
function I2 = smom2(F, b, a)

%       I2 = smom2(F, b, a): Produces a frequency response
%       magnitude vector times f^2 corresponding to
%       frequency values in F. This form is suitable for use
%       with the numerical integration formula quad8().
%       by Mark A. Wickert 7/94.

W = 2*pi*F;
I2 = freqz(b, a, W);
I2 = (abs(I2).^2).*(F.^2);
end;
```

```
function bch_pg = bch_gen(n,k)
%returns generator polynomial for the bch code with length n and data k.
%little error trapping and correction is implemented
%using MATLAB's Comm library GF(2^m) functions.
%John Waterston - USNA '99
%24 SEP 98

%create primitive polynomial for GF(2^m)
dim = ceil(log2(n));
m = gfprimdf(dim);
%compute minimum polynomial
if k == n - dim
   %trivial case where primitive poly is generator poly
   bch_pg = m;
else
   %generate cyclotomic cosets and use to create min poly's
   cs = gfcosets(dim);
   [n_cs, m_cs] = size(cs);
   pl = gfminpol(cs(2:n_cs,1),m); %does multiplication

   n_terms = 0;
   n_i = 0;
   pm = [];
   while ((n_terms < (n-k)) & (n_i+1 < n_cs))
      n_i = n_i + 1;
      n_terms = n_terms + sum(~isnan(cs(n_i+1,:)));
      pm = [pm; pl(n_i, :)];
   end;

   bch_pg = gftrunc(pm(1,:));
   [n_pm, m_pm] = size(pm);
   if n_pm > 1
      for i = 2 : n_pm
         bch_pg = gfconv(bch_pg, gftrunc(pm(i,:))); %multiplies the min poly's
      end;
   end;
end;

%end of bch_gen.m
```

```
function [msg,ccode,err]=debch3(rec_code,n,k,t);
%
% debch - Decodes a block of data that is BCH encoded
% John Waterston
% October 20, 1998
%

[n_code, m_code] = size(rec_code);
dim = ceil(log2(m_code));

ord = gfprimdf(dim);   %the complete list of all element in GF(2^dim)
tp = gftuple([-1:m_code- 1]', ord);

%Initialize error vector at the very beginning
err = zeros(n_code,1);
msg = zeros(n_code,k);
ccode = zeros(n_code,m_code);

for n_i = 1:n_code  %passes each row to core, one at a time.
   %profile johncore
   [msg(n_i,:), err(n_i), ccode(n_i, :)]=...
      johncore(rec_code(n_i,:), m_code, dim, k, t, tp);
   %profile report, profile done
   %break
end;
```

```
function [msg, err, ccode] = johncore(code, pow_dim, dim, k, t, tp)

tp_num = tp * 2.^[0:dim-1]';
tp_inv(tp_num+1) = 0:pow_dim;
% syndrome computation.
% initialization, find all non-zeros to do the calculation
non_zero_itm = find(code > 0) - 1;
len_non_z_itm = length(non_zero_itm);
syndrome = -ones(1, 2*t);

% syndrome number is 2*t where t is error correction capability
if len_non_z_itm > 0
    tmp = 1:2*t;
    syndrome(tmp) = non_zero_itm(1) * tmp;
    if len_non_z_itm > 1
        for n_k = 2 : len_non_z_itm
            syndrome(tmp) = gfplus(syndrome(tmp), non_zero_itm(n_k) * tmp, tp_num,
tp_inv);
        end;
    end;
end;
% complete syndrome computation

% Determine the error-location polynomial.
%profile jwwelocp
[sigma, err] = jwwelocp(syndrome, t, tp, pow_dim, 0);
%profile report, profile done

% computation of error-location numbers.
loc_err = zeros(1, pow_dim);

% in case of failed or no error, skip.
num_err = length(sigma) - 1;
if (~err) & (num_err > 0)
    cnt_err = 0;
    pos_err = [];
    er_i = 0;
    while (cnt_err < num_err) & (er_i < pow_dim * dim)
        test_flag = sigma(1);
        for er_j = 1 : num_err
            if sigma(er_j + 1) >= 0
                % The following 6 lines is equivelent to
                % tmp = gfmul(er_i * er_j, sigma(er_j+1), tp);
                tmp = er_i * er_j;
                if (tmp < 0) | (sigma(er_j+1) < 0)
                    tmp = -1;
                else
                    tmp = rem(tmp + sigma(er_j + 1), pow_dim);
                end;
                test_flag = gfplus(test_flag, tmp, tp_num, tp_inv);
            end;
        end;
        if test_flag < 0
            cnt_err = cnt_err + 1;
            pos_err = [pos_err, rem(pow_dim-er_i, pow_dim)];
        end;
        er_i = er_i + 1;
    end;
    pos_err = rem(pow_dim+pos_err, pow_dim);
    pos_err = pos_err + 1; % shift one location because power zero is one.
    loc_err(pos_err) = ones(1, cnt_err);
    err = num_err;
else
    if err
        err = -1;
    end;
end;
% completed error location detection

% correct the error
ccode = rem(code + loc_err, 2);
msg = ccode(pow_dim-k+1 : pow_dim);
%-- end of johncore---
```

```
function [sigma, err] = jwwelocp(syndrome, t, tp, pow_dim, err)
%John Waterston - USNA '99
%2 October 1998


[tp_n, tp_m] = size(tp);
tp_num = tp * 2.^[0 : tp_m-1]';
tp_inv(tp_num+1) = 0:pow_dim;

% use simplified algorithm
mu = [-1/2, 0:t]';
sigma_mu = [zeros(t+2,1), -ones(t+2, t)];
d_mu = [0; syndrome(1); zeros(t, 1)];
l_mu = [0 0 2*(1:t)]';
mu2_l_mu = 2*mu - l_mu;
% iterative start with row three. The first two rows are filled.
for de_i = 3:t+2
   % no more effort to failed situation
   if (d_mu(de_i - 1) < 0) | err
      sigma_mu(de_i, :) = sigma_mu(de_i-1, :);
   else
      % find another row proceeding to row de_i -1
      % d_mu equals to zero
      % and 2*mu - l_mu is the largest.
      indx = find(d_mu(1:de_i - 2) >= 0);
      rho  = find(mu2_l_mu(indx) == max(mu2_l_mu(indx)));
      rho = indx(rho(length(rho)));

      % by (6.28)
      % shifted = gfmul(d_mu(de_i - 1), pow_dim - d_mu(rho), tp);
      % shifted = gfmul(shifted, sigma_mu(rho, :), tp)';
      % multiply inreplace the above two lines.
      shifted = -ones(1, t+1);
      if (d_mu(de_i - 1) >= 0) & (pow_dim - d_mu(rho) >= 0)
         tmp = rem(pow_dim - d_mu(rho) + d_mu(de_i - 1), pow_dim);
         indx = find(sigma_mu(rho,:) >= 0);
         for de_k = 1 : length(indx)
            shifted(indx(de_k)) = rem(tmp + sigma_mu(rho, indx(de_k)), pow_dim);
         end;
      end;
      % end multiply

      shifting = (mu(de_i - 1) - mu(rho)) * 2;

      % calculate new sigma_mu
      %if ~isempty(find(sigma_mu(1:de_i-2,max(1,t-shifting+2):t+1) >=0))
        % disp('A potential for error is posible')
        % disp(['de_i=',num2str(de_i), ' shifting=',num2str(shifting)]);
        % disp(sigma_mu)
        % disp(['shifted=',mat2str(shifted)]);
      %end;
      if ~isempty(find(shifted(max(t-shifting+2, 1) : t+1) >= 0))%possibly not needed- jww
         % more than t errors, BCH code fails.
         err = 1;
      else
         % calculate the new sigma
         shifted = [-ones(1, shifting) shifted(1:t-shifting+1)];
         sigma_mu(de_i, :) = gfplus(sigma_mu(de_i-1,:), shifted, tp_num, tp_inv);
      end;
   end;
   l_mu(de_i) = max(find(sigma_mu(de_i,:) >= 0)) - 1;

   % calculate d_mu. It is not necessary to do so if mu(de_i) == t
   if de_i < t+2
      % the constant term
      d_mu(de_i) = syndrome(mu(de_i) * 2 + 1);
      indx = find(sigma_mu(de_i, 2:t) >= 0);

      for de_j = 1 : length(indx)
         de_j_tmp = indx(de_j);
         % Before the "end", it is equivalent to
         % d_mu(de_i) = gfadd(d_mu(de_i), ...
         %              gfmul(sigma_mu(de_i, de_j_tmp+1), ...
```

```
            %              syndrome(mu(de_i) * 2 - de_j_tmp + 1), tp), tp);
            tmp = syndrome(mu(de_i) * 2 - de_j_tmp + 1);
            if (tmp < 0) | (sigma_mu(de_i, de_j_tmp + 1) < 0)
               tmp = -1;
            else
               tmp = rem(tmp + sigma_mu(de_i, de_j_tmp + 1), pow_dim);
            end;
            d_mu(de_i) = gfplus(d_mu(de_i), tmp, tp_num, tp_inv);
         end;
      end;

      % calculate 2*mu-l_mu
      mu2_l_mu(de_i) = mu(de_i) * 2 - l_mu(de_i);
   end;

   % the error polynomial
   sigma = sigma_mu(t+2, :);
   % truncate the redudancy
   indx = find(sigma >= 0);
   sigma = sigma(1:max(indx));
   % completed constructing error polynomial

   %-- end of errlocp(jww) --
```

```
%BPSKCAL.m
%Generates the BPSK output and expected waterfall graph
%John Waterston - Trident Project
%October 7, 1998

EbNo=-5:1:12;
       %in DB
semilogy(EbNo,commq(sqrt(2*10.^(EbNo/10)))); %plot and compute Pe
grid
ylabel('P_e');
xlabel('E_b/N_o (dB)');
%title('BPSK Signal');
axis([-5 15 1e-8 1]);
hold on;

EbNo = linspace(.5,11,10);
N_bits=10000;
Nb=4;
BEP=zeros(1,length(EbNo));

for i = 1:length(EbNo)
   errors=0;
   loop=0;
   tic
   pe=commq(sqrt(EbNo(i)));
   while (N_bits*loop) < (10000/pe)
      loop=loop+1;
      sigma=sqrt(Nb/(2*EbNo(i)));
      data=(randn(1,N_bits)>0);
      a_data=data*2-1;
      noise=sigma*randn(Nb,N_bits);
      channel=(a_data'*ones(1,Nb))'+noise;
      recv=sum(channel)>0;
      errors=errors+sum(abs(data-recv));
      BEP(i)=errors/(N_bits*loop);
   end
   t=toc;
   fprintf(1, '%5g  %12g    %12.10f       %5g    \n',errors,loop*N_bits,BEP(i),t)
   %plot(10*log10(EbNo(i)),BEP, 'rd');
end
plot(10*log10(EbNo),BEP, 'rd');
legend('Expected Curve','Experimental Data');
hold off;
```

```
%
% 'bchbnd2.m'
% BCH comparison to bounds
% BPSK vs. BCH of various (n,k)
% John Waterston
% Trident Project
% February 26, 1999
% This creates a BCH matrix with the BEP (-4 dB to 9 dB) for all 11 63 codes

%Code Parameters
n=63;
k_list=[7,10,16,18,24,30,36,39,45,51,57];
t_list=[15,13,11,10,7,6,5,4,3,2,1];

a=input('What code do you want to simulate? (1-11) ');
b1=input('What dB do you wish to begin at? ');
b2=input('What dB do you wish to end at? ');

EbNo=-4:1:9;                          % This creates the general matrix for all to use.
BEP=zeros(length(k_list),length(EbNo));
tloop=zeros(length(k_list),length(EbNo));
errors=zeros(length(k_list),length(EbNo));
load bnd
for code=a:a;  %length(k_list):-1:5;
   fn=['tst',num2str(a)]
   k=k_list(code);
   t=t_list(code);
   N_bits=500*n;                                          % do not change values...
   K_bits=500*k;
   Nb=4;

   pg = bch_gen(n,k);
   m  = length(pg) - 1;
   b=[];
   for i = 0:k - 1
      [q, tmp] = gfdeconv([zeros(1, n-k+i), 1], pg); % tmp is remainder of division
      tmp = [tmp zeros(1, m -length(tmp))];
      b=[b; tmp];
   end;
   gm = [b eye(k)];

   for i = find(EbNo == b1):find(EbNo == b2)         % Eb is for channel bits
      loop=0;
      tic
      while ((loop*K_bits) < fix(1000/(Pb(code,i))))
         loop=loop+1;
         sigma=sqrt(Nb/(2*10^(EbNo(i)/10)));         % RMS Voltage of Noise
         data1_v=(randn(1,K_bits)>0);                          % Generate Random Data
         data1_m = vec2mat(data1_v, k);
         code1 = rem(data1_m * gm, 2);                         % Encode Data with BCH
         code1=code1';code1=code1(:)';                         % Matrix to Vector
         a_code=(2*code1-1);                                   % Make BPSK
         noise=sigma*randn(Nb,N_bits);                         % AWGN
         channel=(a_code'*ones(1,Nb))'+noise;
         recv=sum(channel)>0;
         recv=vec2mat(recv,n);
         data2_m=debch(recv,n,k,t);                            % Decode recv
         data2_m=data2_m';
         data2_v=data2_m(:)';
         errors(code,i)=errors(code,i)+sum(abs(data1_v- data2_v));
      end
      e_time=toc;
      tloop(code,i)=loop;
      BEP(code,i)=errors(code,i)/(K_bits*loop);
      fprintf(1, '%5g  %12g    %12.10f       %5g    %5g \n',...
         errors(code,i),loop*K_bits,BEP(code,i),e_time,EbNo(i));
      z1=['BEP'];
      z2=['tloop'];
      z3=['errors'];
      z4=['code'];
      save(fn,z1,z2,z3,z4);
   end;
end;
```

```
% 'Raycal.m'
% Raleigh Channel calibration
% John Waterston
% February 22, 1999
% Trident Scholar Project
%

%clear;
clc;

Nb=4;
                              % Samples per bit in the transmission channel
loop=1;
len=50000;
errors=zeros(1,11);
z=zeros(1,11);

tic
N_bits=len*1.2;
randn('state',0);
[b,a,gN,Fd,L,fc]=raychan(30,881,2400,1);

for EbNo=0:3.5:35;
   [ray_wgts,Zf1,Zf2]=mk_warm(b,a,gN,L,N_bits+2000);
   while errors(loop) < fix(-12500*log10(1/(4*EbNo)))
      [ray_wgts,Zf1,Zf2]=mk_wgts1(b,a,gN,L,N_bits,Zf1,Zf2);
      vc=abs(ray_wgts(1:len));
      len=length(vc);
      data=randn(1,len)>0;
      a_code=(2*data-1).*vc;
      sigma=sqrt(Nb/(2*10^(EbNo/10)));                          % RMS Voltage
of Noise
      noise=sigma*randn(Nb,len);                                        %
Create AWGN dependant on EbNo of signal
      channel=((a_code'*ones(1,Nb))'+noise);                % Multiplies by
Raleigh
      recv=sum(channel)>0;
      errors(loop)=errors(loop)+sum(abs(recv-data));        % Overall Compare (used for
evaluation and analysis)
      z(loop)=z(loop)+1;
   end
   loop=loop+1;
   disp(loop-1);
end
et=toc;                                                             % End Stopwatch

EbNo=0:3.5:42;
BEP=errors./(len.*z);
BPSK=0.5*(1-sqrt((10.^(EbNo./10))./(1+10.^(EbNo./10))));
                        % From Proakis p.781 diversity=1

%Performance Calculation Phase
disp(['DATA SHEET / raycal.m / ', datestr(now)]);
disp(' ');
%Performance Output Phase

figure(1)
subplot(3,1,[1 2 3])
%semilogy(EbNo(1:11),BEP(1:11);
semilogy(EbNo,BPSK,'b-',EbNo(1:11),BEP,'rd');
legend('Expected Curve (BPSK)','Experimental Data');
grid;
axis([0 40 1e-4 1]);
%title({'John Waterston - Trident Project',['raycal.m /',datestr(now)]});
xlabel('E_b/N_o');
ylabel('P_r');
```

```
%
% 'dat_edit.m'
% Edit BCH data files
% John Waterston
% 1/29/99
%

clear; clc;

n=63;
k_list=[7,10,16,18,24,30,36,39,45,51,57];
t_list=[15,13,11,10,7,6,5,4,3,2,1];


%Code Parameters

EbNo_base=-4:1:9;              % Base Matrix

disp('Data Editor for the BCH (63,K) family of codes');
disp('Written by John Waterston 2/11/99 for Trident Project');

a=0;
while a >= 0
   disp(' ');
   a=input('Do you want to combine new [0] data files [-1] to end? ');
   disp(' ');
   if a==0
      % Take Data from new data file
      what
      a=input('What tst?.mat file do you wish to load? (Enter value of CODE(1-11)) ','s');
      %b=num2str(t_list(find(k_list==str2num(a))));
      %c=input('                                        (Other file apnds)  ','s');
      fname=['tst',a,'.mat'];
      load(fname);
      disp(['File ',fname,' has been loaded...']);
      disp(['The data runs from ',num2str(EbNo_base(1)),' to
',num2str(EbNo_base(length(EbNo_base))),' dB']);
      disp(['Incremented by ',num2str(EbNo_base(2)-EbNo_base(1)),' dB steps.']);
      disp(' ');
      d=input('Do you wish to combine this data with the master BCH data? [1]=yes [0]=no
');
      if d > 0
         load master
         for i=1:length(EbNo_base)
            code=str2num(a);
            master_err(code,i)=master_err(code,i)+errors(code,i);
            master_bits(code,i)=master_bits(code,i)+(tloop(code,i).*k_list(code)*500);
            if master_bits(code,i) ~= 0
               master_BCH(code,i)=master_err(code,i)/master_bits(code,i);
            end
            save master master_BCH master_bits master_err
         end
      end
   end
end

disp('End of Session');
```

```
%
% 'datprint2.m'
% John Waterston
% This prints the current data for the 63 family of codes.
% 1/29/99
%

clear;
clc;

n=63;
k_list=[7,10,16,18,24,30,36,39,45,51,57];
t_list=[15,13,11,10,7,6,5,4,3,2,1];

disp('BCH code printer')
disp('Written by John Waterston');
disp('for Trident Project on March 2, 1999');
disp(' ');

a=0;
a=input('Enter (1) for channel bits (2) for info bits (3) for metric: ');
b=0; c=[];
while b>=0 & b~=99
   b=input('What codes do you wish to print? (-1 to stop, 99 for all) ');
   c=[c b];
end;
c=c(1:length(c)-1);
if isempty(c)
   c=1:1:11;
end

d=input('Would you like to print the upper bounds with your data? (1=yes 0=no) ');

load master2;
load bounds2;
EbNo=-4:1:9;

for i=1:length(c)
   if a == 1
      semilogy(EbNo,master_BCH(c(i),:),'ro:');                    % For a channel bit -
EbNo in dB
      title(['BCH Code Family of (63,M) - Channel Bits']);
      hold on;
      if d == 1
         semilogy(EbNo2,Pb(c(i),:),'b-');                  % For a channel bit - EbNo in
dB
      end
   elseif a == 3
      for j=1:14
         if master_BCH ~= 0
            semilogy(EbNo(j),(n/k_list(c(i)).*-log10(master_BCH(c(i),i))),'rd');
         end
      end

   elseif a == 4

plot(EbNo+(10*log10(n/k_list(c(i)))),(n/k_list(c(i))).*-log10(master_BCH(c(i),:)),'r');
   else
      nebno=EbNo+(10*log10(n/k_list(c(i))));
      semilogy(nebno,master_BCH(c(i),:),'r')
      title(['BCH Code Family of (63,M) - Information Bits']);
   end;
end;

if d==1
   legend('Experimental Data','Upper Bounds');
end
ylabel('P_e');
xlabel('dB');
%clc

axis([-4 9 1e-8 1]);
hold off
```

```
%
% 'varest.m'
% Estimate Variance
% Used in Development of Statistical Adaption Method
% John Waterston
% November 12, 1998
%

clear
clc

n=63;
sigma=2;
Nb=4;

channel=randn(1,n)>0;
channel=channel*2-1;
noise=sigma*randn(Nb,n);      % AWGN
channel=(channel'*ones(1,Nb))'+noise;

l=length(channel);
vmax=max(max(channel));                    % Voltage Max in mV
vmin=min(min(channel));                    % Voltage Min in mV
a=511/(vmax-vmin);                         % 1/(Bin Size)
hg=zeros(1,512);                           % Define Histogram Array and initialize with
zeros
x=fix((channel-vmin)*a)+1;

% Create PDF Histogram (Scaled)
step=1/(size(channel,1)*size(channel,2));
for i=1:size(channel,1)
   for j=1:size(channel,2)
      hg(x(i,j))=hg(x(i,j))+step;
   end;
end;
E=0;
v=vmin:((vmax-vmin)/511):vmax;

% Mean - Expected Value
for i=1:length(hg)
   E=E+v(i)*hg(i);
end;

% Second Moment -
m2=0;
for i=1:length(hg)
   m2=m2+(v(i)-E)^2*hg(i);
end;

% Fourth Moment
m4=0;
for i=1:length(hg)
   m4=m4+(v(i)-E)^4*hg(i);
end;

mhat=((3/2)*(m2)^2-(1/2)*m4)^(1/4);

sig2=sqrt(m2-mhat^2);

disp(['The input value of sigma is: ',num2str(sigma)]);
disp(['The estimated value of sigma is: ',num2str(sig2)]);
disp(['The error is: ',num2str(abs((sigma-sig2)/sigma*100)),'%']);

% Display Data
figure(1)
plot(v,hg);
title(['John Waterston, Variance Estimation']);
xlabel('PDF(x)');
ylabel('Probability');
```