Adaptive Equalization for Indoor Wireless Channels

John M. Morton

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Dr. Brian D. Woerner, Chairman
Dr. Amy E. Bell
Dr. William H. Tranter

August 3, 1998
Blacksburg, Virginia

Keywords: Adaptive Equalization, Indoor Channels, Wireless Communications

Adaptive Equalization for Indoor Wireless Channels

John M. Morton

ABSTRACT

This thesis describes the use of adaptive equalization techniques to compensate for the inter-symbol interference (ISI) that results when digital data is transmitted over a multipath radio channel. The equalization structures covered in this work are the linear transversal equalizer (LTE), the fractionally spaced equalizer (FSE), the decision-feedback equalizer (DFE), and the maximum-likelihood sequence estimation (MLSE) equalizer. This work also covers adaptive algorithms for equalization including both the least mean squares (LMS) and the recursive least squares (RLS) algorithm. All these equalizer structures and algorithms will be modeled using various simulation modules. Equalization for both stationary and mobile radio channels is considered. Stationary channels are modeled with a simple exponentially decaying profile. The mobile radio channel is represented using a two-ray Rayleigh fading model for an outdoor environment. The SIRCIM channel modeling tool is used to create channel profiles for an indoor mobile radio channel. Adaptive arrays and their similarities to linear equalizers are also studied in this thesis. The properties and performance of simple adaptive array systems using the LMS and RLS algorithms are examined through simulation. This thesis concludes with an in-depth study of the use of adaptive equalization for high-speed data systems operating in an indoor environment. Both stationary and slowly varying radio channels are examined. Simulations of DFE and MLSE equalizers operating in such a system show that both equalizer structures provide better BER performance over a system with no equalization. These simulation results also show that the MLSE equalizer provides better performance than the DFE in almost all cases, but requires a great deal more computations.

Acknowledgments

I would like to express my deepest thanks to my advisor, Dr. Brian D. Woerner, for his invaluable guidance, encouragement, and support. I would also like to thank the rest of my committee members, Dr. Amy E. Bell and Dr. William H. Tranter, for reading over this thesis. I am very grateful for their advice and suggestions.

I would also like to thank Dr. Virginia Stonick and Dr. John Stonick, formerly of Carnegie Mellon University, for introducing me to the topic of adaptive equalization and allowing me to work on an independent project relating to the subject during my undergraduate studies.

I extend special thanks to Rennie Givens and Hilda Reynolds for helping me with all the administrative details during my time here at the MPRG.

I am very grateful to the late Marion Bradley Via and the H. Lynde Bradley Fellowship for providing me with financial support during my stay at Virginia Tech.

Thanks to all my friends who have put up with me for the last year and a half.

Finally, I wish to thank my parents who have provided me with never-ending support and encouragement during all my years in academia.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Intersymbol interference (ISI) is one of the largest impediments to the efficient transmission of digital data. ISI results when the symbols contained within a data stream interfere with one another. This causes the signal to become distorted. To combat the distortion introduced by ISI, a process known as equalization is often employed by a communication system. Equalizers try to reconstruct the original signal and undo the ISI through the use of filters or other techniques [1] [2].

Intersymbol interference occurs in all types of communication systems and has many different causes. This thesis will focus on the ISI that results from different types of wireless channels. Wireless channels are often multipath channels. In this type of channel delayed versions of the signal entering the channel are added with the original signal at its output. An equalizer can be used in all these cases to combat the signal distortion caused by the ISI.

The wireless channel that results from an urban environment is a good example of a multipath channel [3]. A wireless digital communication system operating in such an environment will require an equalizer to operate effectively. In a city, a radio signal will bounce off of buildings and other obstacles in the environment. These reflected signals travel a slightly longer distance than the line-of-sight signal and will lag behind the direct signal when they are received. This will cause adjacent symbols to interfere with one another which causes

signal distortion which in turn causes bit errors. An equalizer can compensate for the distortion caused by the multipath channel allowing for more accurate data reception.

Wireless channels are always changing. Movement of the transmitter or receiver and movement of obstacles in the environment will cause the channel to change with time. For this reason the equalizer is often made adaptive. An adaptive equalizer is able to track the varying interference and remove it.

Co-channel interference is another hindrance to the reliable transmission of digital data. This type of interference results when two users are operating on the same radio frequency. Depending on the strength of each users' signal, this interference has the potential to severely limit the accurate reception of the desired user's data.

An adaptive array is a device used in wireless communication systems to compensate for co-channel interference. The adaptive array is made up of an array of antennas whose radiation pattern can be electronically steered. The fact that a desired user and its interferers are most likely located at different angles from the array can be used as an advantage in fighting co-channel interference. By placing the nulls of the antenna array at the angles of the interferers, a desired user's signal can be received with little or no interference from the other users operating on the same frequency. This spatial filtering technique becomes very powerful when the antenna array is made adaptive. This way the location of the nulls and lobes of the antenna array can change with time such that the array is able to track the user and its interferers as they move about the environment.

The two techniques of adaptive equalization and adaptive arrays are often treated together because of their many similarities. Both systems operate on an array of data. Equalization uses an array of samples in time to reduce ISI, while an adaptive array uses an array of samples in space to null out signals arriving from certain angles. Both techniques also use similar algorithms to adapt themselves. The signal processing structures used by both systems are also similar.

There are currently many references and resources available on the subject of adaptive filtering. One of the key papers on adaptive equalization was written by Qureshi in 1985

[1]. Previous to this time, there had been many papers and articles written about different aspects of adaptive filtering and its benefits. Qureshi was able to bring together many of these ideas into one paper. The paper also serves as a tutorial, presenting the information in a heuristic manner and does not delve into many lengthy derivations. Though the paper is concerned mostly with equalization for telephone channels, the structures and algorithms presented are still relevant to the equalization of modern wireless systems.

Another excellent reference is the book *Digital Communication* [2] by John G. Proakis. Chapters 10 and 11 cover equalizer structures and algorithms quite thoroughly. Proakis provides much of the same background information that Qureshi did in his paper but also provides many comparisons between the performance of different structures and algorithms. Many of these performance comparisons were researched and developed since Qureshi's original paper.

Simon Haykin's *Adaptive Filter Theory* [4] serves as another key source in the subject of adaptive equalization. Because it is a book and not just a paper it is able to provides information on much of the same aspects of adaptive filtering that Qureshi did, but discusses many of these topics in much greater depth. The book is also a much more theoretical reference, concentrating on the theory behind adaptive filtering, and rarely discusses the application of adaptive filtering to real-world systems.

This thesis will differ from the previously mentioned references and other sources. It is intended to be a supplement to resources already available and is not meant to act as a replacement. This thesis will introduce many of the basic concepts of adaptive equalization but will not examine many deeper theoretical concepts associated with the topic. Along with the basic principles of equalization, this thesis will also provide many examples of practical applications where equalization can be used. While many of the other references are oriented toward equalization of phone lines for high speed data communication or only concentrate on theory and not practical applications, this thesis will concentrate mainly on the operation of adaptive filters within wireless communication systems. Many of these applications are illustrated with computer simulation. These simulations, or modules, are written in MATLAB. Using these modules allows students to have a "hands on" understanding of

the different applications where equalization is used. Source code for these simulationsis provided to further their understanding of the different techniques and algorithms.

Using computer simulation to model real world systems provides many benefits. For one, simulation provides great insight into how the system works. Equations and block diagrams provide a general idea of the operation of the system, but do not provide much detail on the inner workings of the system. Through the use of simulation, persons are able to delve deeper into the system to see how every level works. Users can look at different points within the system as the signal propagates through it. By examining these signals and seeing how they change as they pass through each block, a user can gain a deeper understanding of each function and its influence on the overall system. Computer simulation is also useful when experimenting with system parameters. Through simulation a user can easily adjust certain system parameters and see the effects on the system performance almost immediately. This saves much time and resources over developing the system in hardware and experimenting with it after it has already been built.

The power of simulation as a design tool has been proven time and again. However, a user should be cautioned against relying on simulation to provide hard and fast answers to all design problems. Using a discrete time computer to model continuous time systems can prove to be problematic. A user must be sure to model the system carefully. There are many parameters a user needs to be concerned with in such a simulation. For example, the sampling rate must be chosen with several factors in mind. Failure to use a high enough rate can introduce aliasing into the system which will cause simulation error. Using a rate that is too large is inefficient and will make a simulation run slower than it needs to be. The choice of sampling rate is also important when modeling systems that contain feedback. In these systems it is impossible to perfectly model the instantaneous feedback path using sampled signals. Therefore, the sample rate needs to be large enough such that the error between the delayed signal used to model the feedback path and the original signal is negligible.

These issues are not a problem in the adaptive equalization simulations included with this thesis. Since adaptive equalization is a digital process and is implemented using simple digital add and multiply chips or more advanced digital signal processing (DSP) chips,

using a digital computer is the ideal platform on which to simulate it.

Another trap that a user can fall into with simulation is to incorrectly assume that they know all there is to know about the performance of the system after running only a few simulations. Simulations are most often created to model specific characteristics of a system. Users can run into problems if they try to infer system characteristics that the simulation was not designed to measure. For this reason users need to be careful when interpreting the results of their simulations.

The MATLAB program, version 5.0, was used to develop the simulations that support the material contained within this thesis. MATLAB is one of the most widely used tools for developing simulations of communication and other types of systems and for good reason. Its toolboxes provide many high-level commands meant for specific applications. Many functions can be accomplished using only one line of code instead of the many lines of code that a C program would require. MATLAB also provides excellent presentation commands that can be used to graphically display numerical data. Graphical results are very helpful in developing a clear understanding of the simulated system. They provide the user with insight into the workings of the system. Another benefit is that MATLAB is available in a reasonably-priced student addition. While this version does have some vector size limitations, it is still powerful enough to simulate simple systems.

1.1 Outline of this thesis

This thesis is divided into several chapters. Each chapter describes a specific attribute of adaptive equalization or adaptive arrays and how it can be modeled witnin computer simulation.

Chapter 2 provides information on the fundamentals of adaptive equalization. It is intended as an introduction to equalization and does not provide great detail about any of the many different topics that are covered in later sections. The chapter begins by describing the basic digital communication system. It describes the parts that make up this system: the data source, the transmitter, the channel, the receiver, and the data sink. Next, the more

specific wireless communication system is discussed. The idea of spectrum conservation is introduced included the use of Nyquist filters to limit the bandwidth needed by a digital transmission. Also included is a discussion of wireless channels and how to model these channels. The difference between indoor and outdoor channels is addressed here. The basic principles of equalization are also introduced. This section brings up the zero-forcing equalizer and how it works. The chapter concludes with a description of a simulation that models a wireless communication system broadcasting over a multipath channel.

The different signal processing structures that are used to construct equalizers is the focus of Chapter 3. The chapter begins with a description of linear equalization techniques. This includes the linear transversal equalizer (LTE) which is very similar to an FIR filter and the fractionally spaced equalizer which oversamples the signal. Both the benefits and drawbacks of these structure are addressed. Also included with the chapter are discussions of nonlinear equalization techniques. These techniques include the decision-feedback equalizer (DFE) which uses both a feedforward and a feedback filter to equalize the received signal. The feedback filter uses previously detected symbols to remove the ISI. Another nonlinear equalization technique is the powerful maximum-likelihood sequence estimation (MLSE) equalizer. This equalizer compares the received signal with different hypothesised signals to determine the most likely sequence of data. Also included with this chapter are simulations that model all the different equalization techniques. These simulations are described in detail at the end of the chapter.

Chapter 4 introduces the concept of adaptive equalization. This technique is used to equalize channels that are either unknown, time-variant, or both. To equalize these types of channels, adaptive algorithms are used to determine the best equalizer parameters. The first algorithm talked about in this chapter is the least mean square (LMS) algorithm. The LMS algorithm is a simple algorithm that requires many iterations to determine the optimal equalizer taps. The faster but more complex recursive least squares (RLS) is also discussed. The chapter also deals with how to combine these algorithms with the different equalizer structures given in Chapter 3. All four structures are examined. Included with the discussion of the adaptive MLSE equalizer is a description of the channel estimator. This device is used to determine

information about the channel that can be used by the MLSE equalizer. The chapter ends with a description of the many different system simulations included with this chapter.

The next chapter explores the idea of equalization for mobile radio channels. It begins with an in-depth look at how the channel changes due to motion of the receiver or transmitter. This includes the reasons for the time variations and how quickly these variations occur. A few of the techniques used to model these types of channels are also introduced. This includes simple two-ray models as well as some of the more complicated statistical techniques that have recently been developed. Like the other chapters, this one will also conclude with a section describing the simulation programs used to illustrate the topics covered in this chapter. In this case the programs model a wireless communication system employing adaptive equalization techniques to combat time-varying ISI.

Many of the structures and algorithms used by adaptive equalizers can also be used in adaptive array processing. Chapter 6 describes systems that use adaptive arrays and their role in a communication system. It also describes the many parallels between adaptive equalization and adaptive arrays. These parallels includes similar signal processing techniques and adaptive algorithms. Simulations of a system employing adaptive arrays are included with the chapter. A discussion of these simulation concludes the chapter.

Chapter 7 discusses the use of decision-feedback and MLSE equalization with wireless systems operating in indoor environments. The results of several simulations are given which compare the performance of each of these structures for different channel types. The computational complexity of each of these structures is also examined in detail.

Chapter 8 concludes the thesis by summarizing all the information that has been discussed and also provides some suggestions for future research.

Chapter 2

Fundamentals of Equalization

Equalization is defined as any technique that can be used to compensate for the intersymbol interference (ISI) caused by a dispersive channel. The dispersion within the channel can be caused by a number of factors as mentioned in the previous chapter. In a wireless system, the radio channel can prove to be quite hostile. The reflection of the tranmitted radio signal off of objects in the environment such as buildings or mountains causes several versions of the same signal to add together at the receiver. This causes the signal to be highly distorted. Because reflected signals take more time to reach the receiver, they will lag behind the line-of-site signal that travels directly form the transmitter to the receiver. The receiver will see one strong signal followed by delayed and attenuated versions of the same signal.

In a digital communication system, data symbols are modulated onto a carrier and then transmitted. When this signal is sent through a multipath channel, the delays in the signal cause adjacent symbols to be received overlapping in time with one another. This is what is meant by intersymbol interference. This interference can severely degrade the performance of a digital communication system. Removing this interference will allow for much more accurate data detection.

Equalizers can use several different techniques to compensate for ISI. Linear equalizers filter the received signal to remove the ISI. Decision-feedback equalizers also filter the received

data but also use previously detected symbols to remove the ISI they places on the current symbol. Maximum- likelihood sequence estimation is an equalization technique that finds the most likely sequence of transmitted symbols based on how closely the received data comes to a set of hypothesized signals. All these techniques will be discussed in detail in Chapter 3.

In this chapter, the concepts behind equalization are introduced. It begins with a review of a simple digital communications system and how this system is modeled. After this general communication system is discussed, a more specialized model of a wireless system will be studied. Included in this section will be an in-depth look into the multipath wireless channel and how it causes multipath. In the following section, the concept of equalization will be examined in detail. This will include the strategies used by equalizers in removing the intersymbol interference from the received signal and the limitations that exist. The next section will introduce the concept of channel modeling. This includes the development of simple time-invariant channel models that can be used to model different types of wireless channels. The chapter will conclude with a discussion of the different simulation programs that accompany this chapter. These programs simulate a wireless communication system using BPSK modulation to transmit data over a multipath channel.

2.1 Digital Communication System

The digital communication model can be separated into three main parts: the transmitter, the channel, and the receiver [2] [5] [6] [7]. A diagram of the system is shown below.

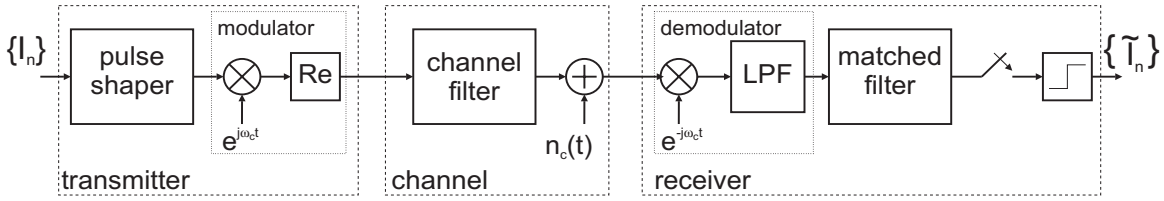

Figure 2.1: Block Diagram of a Digital Communication System

The transmitter takes digital data from a data source, passes it through a pulse-shaping

filter, and then modulates the output of the filter with a carrier signal. The modulated signal is then passed to the channel. The channel bandlimits the signal and adds noise to it. At the receiver, the output of the channel is demodulated with the same carrier to form the baseband signal. This signal is then passed through a matched filter and then sampled at the symbol rate. These samples are then sent to a decision device which determines the received symbols.

Fortunately, since the modulator and demodulator can be approximated as linear devices, they can both be represented in baseband notation using the complex envelope. Since the equalizer is almost always implemented at baseband, representing the modulator and demodulator with this notation will greatly simplify the system model. With this notation, the modulator, and demodulator can be combined with the channel to form one system block. This block is modeled as a filter followed by a baseband noise generator.

When an equalizer is included with a digital communication system, it will operate on the sampled output of the receiver's matched filter. Most equalizer techniques operate at the symbol rate. Others operate at a fraction of the symbol rate. In these systems, the matched filter is often combined with the equalizer, so the equalizer will be placed at the beginning of the receiver.

2.2 The Wireless Communication System

The wireless communication system is a specialized case of the general digital communication system. Since the focus of this thesis is on equalization for wireless systems, this case will be studied in detail.

One major concern in designing a wireless communication system is how much spectrum it requires to transmit its data. Spectrum is very valuable, so the less spectrum is needed the better. For this reason, particular attention is paid to the transmit pulse-shape filter. The filter should bandlimit the signal but should not cause any additional interference. For this reason, Nyquist filters are often used. These filters are able to bandlimit the signal without the addition of any intersymbol interference. One of the most popular Nyquist filters used

in wireless communications is the raised cosine filter [6]. The frequency response for this filter is given by

$$P_{RC}(f) = \begin{cases} 1 & |f| < f_N(1-\alpha) \\ \left\{\frac{1}{2} + \frac{1}{2}\sin\left(\frac{\pi}{2f_N}\left[\frac{f_N - |f|}{\alpha}\right]\right)\right\} & f_N(1-\alpha) \leq |f| \leq f_N(1+\alpha) \\ 0 & |f| > f_N(1+\alpha) \end{cases} \tag{2.1}$$

where f_N is the Nyquist rate equal to $1/2T$ where T is the symbol rate. The parameter α is called the *rolloff factor* and takes the range $0 \leq \alpha \leq 1$. This rolloff factor controls the bandwidth beyond the Nyquist frequency of $1/2T$ called the excess bandwidth. When $\alpha = 0$, there is no excess bandwidth, when $\alpha = 0.5$, there is 50% excess bandwidth, and when$\alpha = 1.0$, there is 100

Usually, in a wireless system, this raised-cosine filter is split into two filters called *square-root raised-cosine filters* [3]. The frequency response of these filters is the square root of the frequency response of the normal raised-cosine filter. The combined response of these two filters will therefore look like the response of a raised-cosine filter. The two filters are divided between the transmitter and receiver so that the combined responses form a Nyquist filter. The impulse response of the raised cosine filter is infinite in length. For this reason, the response is often truncated to exist only between $-3T \leq \tau \leq 3T$ where $\tau = 0$ is the center point of the filter.

A diagram of a wireless communication system that includes these square-root raised-cosine filters is shown below in Figure 2.2. The system is represented in baseband.

The system begins by taking the data symbols from the data source, I_n, and pulse shaping them with the square root raised-cosine filter, $p(\tau)$, to form the transmitted signal, $v(t)$ [1].

$$v(t) = \sum_n I_n p(t - nT) \tag{2.2}$$

This signal is then passed to the baseband channel model. First the signal is filtered by the channel filter, $g(\tau)$. Then noise, $n(t)$, is added to the signal to form the received signal input to the receiver, $y(t)$. The response of the transmit filter and channel filter are often combined and represented as one filter $h(\tau)$. The frequency response of this filter will be

$$H(f) = P(f)G(f). \tag{2.3}$$

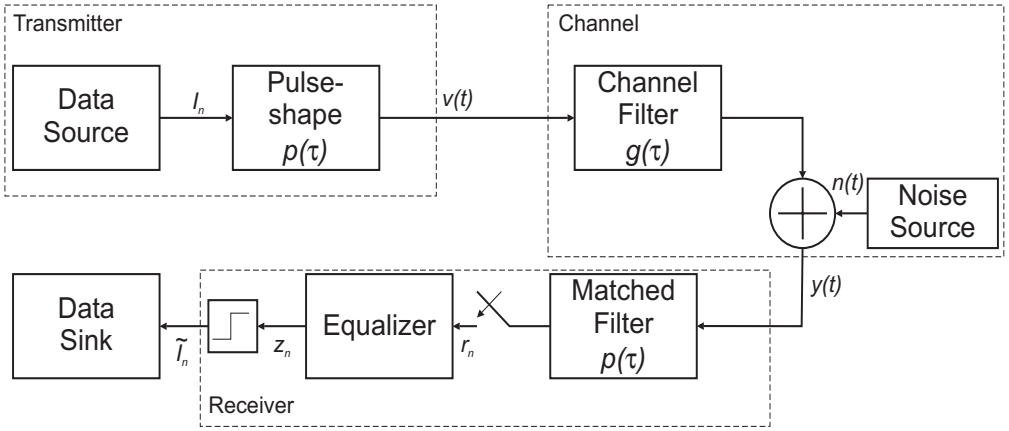

Figure 2.2: Block Diagram of a Wireless Communication System Utilizing an Equalizer

The input to the receiver will then be of the form

$$y(t) = \sum_n I_n h(t - nT) + n(t). \tag{2.4}$$

The receiver then takes this signal and passes it through the matched-filter to form the signal $r(t)$.

$$r(t) = \int y(t)p(t - \tau)d\tau \tag{2.5}$$

This signal is then sampled at the symbol rate to form the discrete time signal $r_n = r(nT)$.

Of particular interest is the overall response of the system, $q(\tau)$. This response is found from convolving the combined transmit and channel responses with the receiver filter.

$$q(\tau) = h(\tau) * p(\tau) \tag{2.6}$$

The received signal samples can then be found directly from the original data symbols.

$$r_n = \sum_k I_k q(nT - kT) + \eta_n \tag{2.7}$$

The noise component of the received symbol is found by passing the noise through the receive filter.

$$\eta_n = \int n(t)p(nT - t)dt \tag{2.8}$$

If the sampled overall response $q_k = q(kT)$ is not a delta function at time zero, then intersymbol interference exists. The frequency response of this sampled response is given by

$$Q'(f) = \sum_k Q(f - k/T) \tag{2.9}$$

This frequency response should be flat across all frequencies. If this is not true, it is the role of the equalizer to make it so.

2.3 Wireless Channel Models

Included in the wireless communication system is a channel block. This block contains two parts: the channel filter and the noise generator. In most cases, the noise can be modeled as a zero-mean, stationary Gaussian random process, $n(t)$, with a flat spectral density. This models the thermal noise generated in the resistors and semiconductors used in the receiver.

The channel filter, on the other hand, is used to model the effective transfer function of the medium between the transmitter and receiver. In the case of a wireless communication system, it is used to model the response of the wireless channel. The distinguishing characteristic of the wireless channel is multipath. In a wireless environment, the radio signal that is broadcast from the transmitting antenna bounces off objects in the environment such as buildings and airplanes in an outdoor environment or walls and furniture in an indoor environment. These reflections add together with the original, direct signal at the receive antenna. Because the reflections take longer to reach the receiver, they are delayed compared to the direct ray. A wireless multipath channel refers to a channel that causes these multiple reflected signals.

Modeling the wireless channel is a difficult task. A different channel exists for every type of wireless environment. Reflected rays will be delayed by a much longer time in an outdoor environment compared to an indoor environment. An outdoor urban environment with lots of large buildings and other obstructions is much different than a rural environment where there are much fewer obstacles. Even indoor channels can vary. The environment of a large factory where the walls are very far apart is very different than an office building, where the

main obstacles could be cubicle walls or even people. All these things must be accounted for when designing an accurate channel model [3] [8] [9] [10] [11].

A multipath channel can be made up of either a continuum of different paths or can have several discrete paths separated by some delay. Most models of multipath channels model the channel as discrete paths because this is the form many of the channels take and because this method is easier to simulate. There are two ways to model this channel. The first is to use a technique called channel sounding to measure the response of an actual channel. The response can than be stored and used in applications where it is needed. The other technique is to create a statistical model of a type of channel. This is the most common method used. The most basic statistical multipath channel model is described by the following equation.

$$g(\tau) = \sum_k \alpha_k e^{-j\theta_k} \delta(\tau - \tau_k) \tag{2.10}$$

The index, k, is the index for each discrete path. Each path has its own characteristics. The variable α_k describes the magnitude of each multipath component, θ_k is the phase of the component, and τ_k is the delay of the component. Each of these path characteristics is a random variable. Their statistical distribution depends on the type of environment that is being modeled. These parameters are also time-varying. How quickly they vary depends on the environment.

A measurement that is very useful in characterizing a channel response is the *rms delay spread* . This measure describes how dispersive the channel is. It provides some idea of how hostile the channel will be. It is defined as

$$\sigma_\tau = \sqrt{\bar{\tau^2} - (\bar{\tau})^2} \tag{2.11}$$

where

$$\bar{\tau^n} = \frac{\sum_k \tau_k^n \alpha_k^2}{\sum_k \alpha_k^2}. \tag{2.12}$$

The typical rms delay spread is on the order of microseconds for an outdoor channel and on the order of nanoseconds for an indoor channel. The delay spread is often normalized with respect to the symbol rate of the tranmitted symbol for convenience.

$$\sigma_N = \sigma_\tau / T_s \tag{2.13}$$

Time variations of the path statistics are dependent on many factors. Movement of the receiver, transmitter, or both can cause time variations. Movement of obstacles in the environment can also cause the channel to change. When modeling and outdoor channel these variations become a major concern. Many wireless systems used outdoors involve movement of the receiver or transmitter, e. g. a mobile phone systems. For this reason, most outdoor channel models account for these rapid time variations. On the other hand, most indoor applications have transmitters and receivers that are static, e. g. a wireless LAN system, or that move very slowly. In this case the time variations of the channel are slow compared to the symbol rate and are not a major concern.

The first few chapters in this thesis will use channel models very similar to those of an indoor channel because time-invariant channels are easier to model and understand. The more complicated time-variant mobile radio channel model will be discussed in Chapter 5.

The indoor channel is usually modeled as several rays with an exponentially decaying profile. The amplitude of the individual path elements can be modeled as a complex Gaussian random variable. Using this model, the individual path magnitude will have a Rayleigh distribution and its phase will be uniformly distributed. The path delays can be modeled several different ways. The most accurate method is to describe the path arrival as a modified Poisson distribution. This seems to provide the most accurate match to measured channel profiles. A less complex method is to have the paths arrive at fractions of the symbol rate. For example, the response could be modeled as

$$g(\tau) = \sum_k A_k \delta(\tau - kT_s/8) \tag{2.14}$$

where A_k is the complex Gaussian path gain and T_s is the symbol rate. In this case there will be eight uniformly spaced paths per symbol. Though this model is not as accurate as one that uses the Poisson distribution, it still provides reasonable approximations of indoor channel profiles [12]. The channel models used in the simulations included with the first few chapters of this thesis will use this simple indoor model. Ten different channel models are included. The characteristics of these models are shown below in Table 2.1.

Table 2.1: Table of Channel Characteristics

Num.	σ_N	Type	Num.	σ_N	Type
1	0.23	LOS	6	1.20	LOS
2	0.34	LOS	7	2.22	LOS
3	0.57	LOS	8	0.85	OBS
4	0.64	LOS	9	0.76	OBS
5	1.18	LOS	10	1.69	OBS

2.4 Equalizer Operation

When the normalized rms delay spread of a channel becomes greater than around 0.1, the ISI introduced by the channel starts to become a major impediment to the accurate reception of data. At this point, the system must employ some sort of technique to compensate for the ISI in order to improve the BER. Equalization is one such technique. It removes the ISI introduced by the channel so that better symbol estimates can be made from the received data signal.

The goal of the equalizer is to have the combined response of the overall channel, q_k, and the impulse response of the equalizer, c_k, equal to a delta function as shown in Equation 2.15. This completely satisfies the Nyquist criterion for distortionless transmission. The equalizer will remove all the intersymbol interference that is caused by the channel [2] [5].

$$d_k = \sum_{j=-\infty}^{\infty} c_j q_{k-j} = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases} \tag{2.15}$$

This type of equalizer is known as a *zero-forcing equalizer* because it forces the impulse response to zero in all but one instant. In the frequency domain this is represented as

$$D(f) = Q'(f)C(f) = 1. \tag{2.16}$$

It then follows that

$$C(f) = \frac{1}{Q'(f)} \tag{2.17}$$

The equalizer tries to become the inverse of the channel. It boosts the frequency elements that are attenuated by the channel and limits the stronger frequency elements such that the combined frequency response of the channel and equalizer is the same over the desired frequency range.

There are several problems with the zero-forcing equalizer. First of all, it may require the equalizer to have a very large number of taps. This would be impossible to implement in a real system. The second problem is that no consideration has been paid to the noise component of the received signal. When the equalizer amplifies the spectral components that were attenuated by the channel, it also amplifies the noise at these frequencies. This is highly problematic for transmission of signals over radio channels because of the deep signal fades that these channels exhibit. In this case the equalizer could cause more harm than good. Finally and maybe most importantly, in most cases the equalizer will not know the exact channel response at each point in time. The channel response will also be time-varying. The method used by the zero-forcing equalizer above will not be of much use because of this. It requires both knowledge of the channel response and assumes that this response does not change.

Fortunately, many of these problems have been solved. Equalizers are able to approximate the infinite tap filter with a truncated, finite tap version, and are still able to remove much of the ISI. Though these equalizers are not as effective, they are realizable in a real system. To solve the problem of noise amplification, more advanced equalizers use both the channel response and the noise characteristics to determine their filter taps. It is done in such a way as to minimize the mean-square error (MSE) of the output of the equalizer. It determines this error by comparing the output of the equalizer with a desired output. These different equalization techniques will be discussed in detail in Chapter 3.

There are several methods that an equalizer can use to gain knowledge of the channel. Probably the most widely used method is to use adaptive algorithms to find the most effective equalizer. Most of these algorithms work by trying to minimize the error between the output of the equalizer and some desired output. These algorithms and how they work with the different equalization techniques will be described in Chapter 4.

2.5 Wireless System Simulation

Included with each chapter in this thesis are simulation programs based on the material covered within that chapter. These simulations are given to help readers further understand the concepts presented in the chapter and to help them gain insight into the simulation process [13]. There is one simulation that will be included with this chapter. It models a digital wireless communication system degraded by intersymbol interference. This is probably one of the most important simulations included with the thesis. Most of the other simulation included with this thesis will be based on this one. In this simulation, no equalizer is used to compensate for the interference. Simulations that do include an equalizer will be included with following chapters.

The simulation is designed to model a wireless communication system. This implies several system characteristics. First, the matched filters used in the transmitter and receiver will be fixed. These filters will be the square root raised-cosine filters mentioned above. Also, all the channel models will be causal. This does not mean that the combined response of the transmit matched filter, the channel, and the receive matched filter will be causal, though the ISI of the noncausal portion will most often be relatively small.

The channel responses used in this simulations are based on the time-invariant channel models described in the previous section. Instead of generating a random channel model each time, the user of the simulation will be able to pick from ten different models. Each model will have its own special properties. Some will be more severe than others. Some will be longer than others.

The simulation is partitioned into three main programs: the preprocessor, the processor, and the postprocessor. Each program has different functions related to the simulation [14] [15]. The preprocessor, is used to set the different simulation parameters. When the program is executed the program will prompt the user for several of these parameters. These parameters will be used by the processor program to simulate the system. There are two sets of data that the preprocessor will ask for. The first set of requested data describe the workings of the simulation itself. These will include parameters such as the number of

symbols to run and the number of samples per symbol there should be. The second set of data deal with the actual system that is being simulated. These may include such things as the E_b/N_0 ratio or the type of channel to use. The advantage of the preprocessor is that a user does not have to edit the simulation program to change different settings, but can instead enter them from the command prompt.

The processor program contains the main simulation code. It uses the settings saved in the workspace by the preprocessor program to simulate the system. The processor will need no extra information from the user. Once the program is executed, it will not require any input or display any output. All the data it computes will be saved to the workspace to be used with the postprocessor.

The postprocessor takes the data generated by the processor and graphically displays them. When the postprocessor program is executed, a graphical menu pops up that a user can use to select between all the different available plots. Some of the available displays are plots of the transmitted data, plots of the received data, and plots of the impulse and frequency responses of different filters in the system, to name a few.

2.5.1 Preprocessor

The preprocessor for the simulation of the wireless communication system is called **wsyspre.m**. It requests all of the information that is needed by the processor program. An example of this program is shown below.

```
>> wsyspre
Enter number of symbols to simulate [1000]:
Enter number of samples (5-20) per symbol [10]:  15
Enter Eb/N0 value [8]:  12
Enter the rolloff factor (0 - 1) of the raised cosine filter [0.3]:
Enter the Number (1-10) of the Channel to Simulate [1]:  4
>>
```

Each request for information is followed by a number in brackets. This number is the default value for the selection. It can be selected by just pressing the return key. These are just suggested values and should not be considered absolute. The user is encouraged to experiment with the different settings to get a feel for the system and the simulation.

Some requests have numbers in parentheses like the"Number of Samples" selection above. These numbers are the limits of the selection. Users are required to enter a value within these limits. If they do not, the preprocessor will reject the value and request a new value.

Once all the values are entered, they are saved to the workspace to be used by the processor.

2.5.2 Processor

The processor program of this simulation is called **wsysproc.m**. It takes the parameters set by the preprocessor and models the specified system. The processor begins by creating the square root raised-cosine filter used for pulse-shaping in the transmitter and match-filtering in the receiver. The MATLAB communication toolbox comes with a function called **rcosine** that creates the filter automatically. It requires information about the number of samples per symbol and the rolloff factor of the filter. Both of these parameters come from the preprocessor. The filter is created using the following commands.

$$\mathbf{MF} = \mathbf{rcosine}(\mathbf{1}, \mathbf{sps}, '\mathbf{sqrt}', \mathbf{alpha}); \mathbf{delayMF} = \mathbf{3} * \mathbf{sps}; \tag{2.18}$$

The constant **sps** is the number of samples per symbol and **alpha** is the rolloff factor of the filter. The second command sets the variable **delayMF** to the delay of the matched filter. This delay signifies the middle or "time zero" tap of the filter. This information will be necessary when the output of the received filter is sampled.

The next step is to find the combined response of the pulse-shape filter and the channel. This is done using the **conv** command.

$$\mathbf{H} = \mathbf{conv}(\mathbf{MF}, \mathbf{G}); \tag{2.19}$$

The overall channel response is found by including the receiver's matched filter.

$$\mathbf{Q} = \mathbf{conv}(\mathbf{H}, \mathbf{MF}); \tag{2.20}$$

This response is then sampled to find the discrete overall response.

$$\mathbf{Q2} = \mathbf{Q}(\mathbf{1} : \mathbf{sps} : \mathbf{length}(\mathbf{Q})); \tag{2.21}$$

The next step is to normalize the channel filter such that the received signal will have unit energy. This is done for both convenience and consistency. To normalize the channel response we divide the channel filter by the square root of the sum of the sampled response taps.

$$\mathbf{Gn} = \mathbf{G}/\mathbf{sqrt}(\mathbf{Q2} * \mathbf{Q2}'); \tag{2.22}$$

Once the channel has been normalized, the signal vectors are created. This simulation operates using block-by-block method. This means that a system block operates on an entire vector of data before its output is passed to the next block. This method can be used in this case because the main objective of these simulations is to produce the many different waveforms that are possible. If the objective was to count bit errors in order to create accurate BER plots, the symbol-by-symbol method, where a symbol is passed through the entire system before the next one is sent, would be more desirable.

With the block-by-block method, the MATLAB commands **filter** and **conv** can be used to perform all the filtering operations that are required in this system model. The **filter** command takes the filter coefficients and the vector of data to be filtered as its input and produces the filtered data vector as its output. It operates well when the filter is causal. For this reason it is used to simulate the channel filter. The **conv** command works better when the filter is not causal. That is why it is used to perform the matched filtering operations. Both commands are very quick and efficient when filtering vectors of data. They are often used in block-by-block simulations. Because of the nature of the symbol-by-symbol simulation method, the filter command is not usable and the simulation can become slow. This is why the block-by-block method is often advantageous in these types of simulations.

It should be noted that almost always, the impulse responses of the various filters have been capitalized, e.g. **MF, Gn, Q**. Most of the signal variables will be in lower-case such as **y, r2, noise**. The only difference will be the vector of data bits, **I**. This convention is used to

keep the two types of variables separate and the program easier to understand.

The first signal vector that is created is the noise vector. The vector is created using the normal random number generator supplied by MATLAB called **randn()**. The output of the random number generator will have a Gaussian distribution of zero mean and unity variance. To create a noise vector with the correct power for the simulation, this vector is scaled using the E_b/N_0 ratio set by the preprocessor.

$$\mathbf{nvar} = \mathbf{1}/(\mathbf{2} * \mathbf{10\hat{}(EbNo/10)}); \tag{2.23}$$

The next signal to be created is the vector of data bits, **I**. This is done by creating a vector of uniform random variables between zero and one. This vector is then rounded and manipulated such that the vector will hold equally-likely values of ± 1. Once this vector is created, it is lengthened such that the data values will look like pulses at the symbol periods. This new vector is called **d** and is the input to the pulse shape filter. This filter takes this input vector and computes the output of the transmitter, **v**, using the following line.

$$\mathbf{v} = \mathbf{conv}(\mathbf{MF}, \mathbf{d}); \tag{2.24}$$

The signal is filtered by the normalized channel response.

$$\mathbf{y} = \mathbf{filter}(\mathbf{Gn}, \mathbf{1}, \mathbf{v}); \tag{2.25}$$

The noise vector created above is added to the output of the channel filter to form the input to the receiver, **y2**.

The received signal is then passed through the matched filter. This filter is the same as the pulse-shape filter used in the transmitter. The output of the matched filter is the vector **r**. This vector is then sampled at the symbol rate.

$$\mathbf{r2} = \mathbf{r}(\mathbf{2} * \mathbf{delayMF} + \mathbf{1} : \mathbf{sps} : \mathbf{num\_sym} * \mathbf{sps} + \mathbf{2} * \mathbf{delayMF}); \tag{2.26}$$

In most of the simulations that will follow in this thesis, this is where the equalizer would come in to combat the ISI affecting the signal. Since no equalizer is used in this simulation,

the signal is passed directly the decision device which computes the vector of estimated bits.

$$\mathbf{Iest} = \mathbf{2} * (\mathbf{real(r2)} > \mathbf{0}) - \mathbf{1}; \tag{2.27}$$

This vector is then compared with the original vector of bits, **I**, to determine if any errors occured. These errors are then counted and a bit-error rate is computed for this block of data. This bit-error rate is called **BERisi** and is printed to the MATLAB screen.

This simulation also models the same system, but without the channel filter. The only limiting factor of the system is the additive noise. This system is easily simulated. Its output can be compared with the output of the system that suffers from ISI. This will determine how much ISI degrades the performance of a digital system.

This system is modeled exactly the same as the other one, except that the input of the receiver is the addition of the **v** signal vector with the noise vector instead of the addition of the **y** signal vector with the noise vector. The vector of bit estimates for the output of this system is called **Iestprime** and the BER associated with these estimates is called **BERnoisi**. A theoretical BER calculated from the E_b/N_0 ratio is also calculated and is displayed as **BERnoisitheory**.

While the processor runs, it saves the vectors it computes in the workspace. These values can then be viewed graphically with the postprocessor.

2.5.3 Postprocessor

The postprocessor program, called **wsyspost.m**, is used to display the data produced by the processor program in a way that is both informative and provides insight to the system that has been simulated. There are several different displays that are available with the postprocessor. On most systems once the program is executed, a graphical menu will pop up on the screen. The menu for this program is shown in the figure below.

The first selection of the menu, "Waveforms of Transmitted and Received Data", creates two plots. The top plot shows the signal representing the first 50 symbols that are sent. The bottom plot shows the same signal after it has passed through the channel filter and

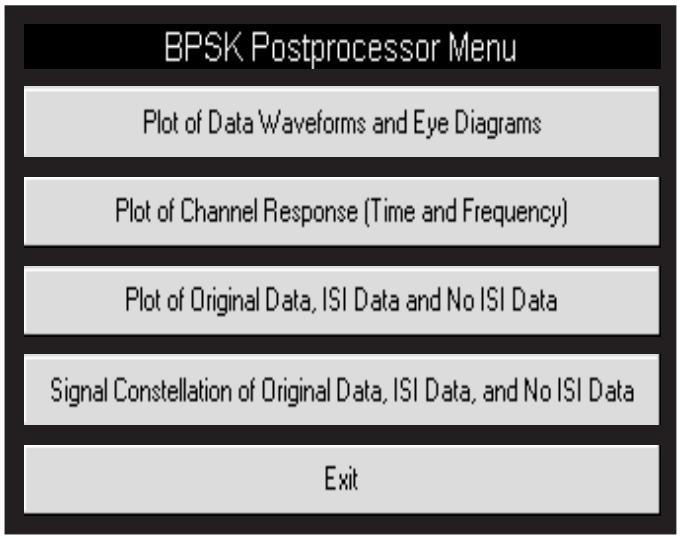

Figure 2.3: Menu for the Wireless System Simulation Postprocessor

has been degraded by noise.

The next selection plots the eye-diagram of the received signal for the system degraded by ISI and the received signal of the system free of ISI. An example of this plot is shown below in Figure 2.4.

This plot gives a good indication of how ISI affects the received signal. In the top plot that shows the eye diagram of a system with no ISI, the value of the signal at the sampling points (in this case the multiples of ten) is two distinct values. In eye diagram of the received signal with ISI, many different values exist at the sampling points.

The next menu selection is a plot of the channel response. This displays both the continuous time impulse response and the sampled response. The frequency response of both impulse responses are also shown.

The fourth selection displays a stem plot of the first 25 values of the original data, the sampled received signal with ISI, and the sampled received signal with no ISI. This plot allows the user to compare the signal symbol-by-symbol.

The final selection plots the scatter diagram of the original signal and both possible received signals. A scatter plot takes the samples of the signal and plots their real parts on the x-

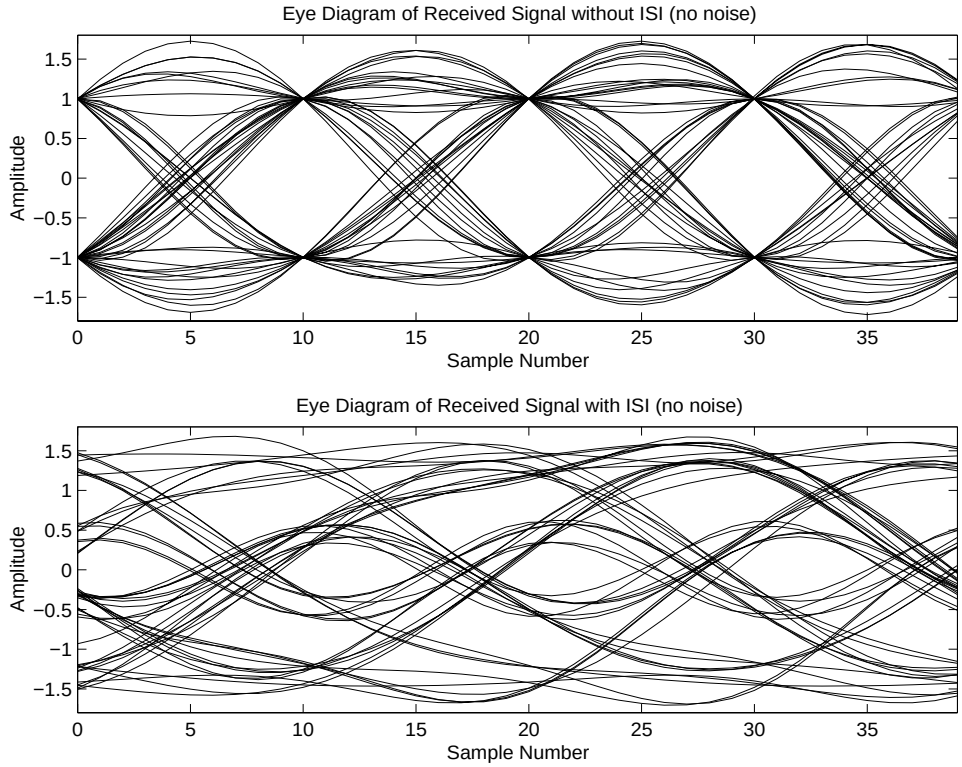

Figure 2.4: Eye Diagrams for Received Signal of Wireless System Model

axis and their imaginary parts on the y-axis. This gives a good indication of how the noise affects the signal along with the ISI. Also included with these plots are histograms of the real part of the signal. This will provide a statistical distribution that describes the received signal.

Chapter 3

Equalization Techniques

There are several different techniques that are used to compensate for ISI in a digital communication system. All of these equalization techniques differ in complexity and performance. Four different equalization types are proposed in this chapter. The first is the *linear transversal equalizer* (LTE). This equalizer is the same as a simple linear FIR filter with tap spacings at the symbol rate but has coefficients that are adjustable based on the channel response. A second equalization technique that is similar to the linear transversal equalizer is *fractionally spaced equalization* (FSE). A fractionally-spaced equalizer also uses a linear filter with adjustable coefficients but has tap spacings at a fraction of the symbol rate. Another equalization technique is *decision-feedback equalization* (DFE). In this technique, the equalizer uses previously detected symbols from the output of the equalizer to combat the ISI imposed on the current symbol. The final equalization technique that will be presented is *maximum-likelihood sequence estimation* (MLSE). MLSE is a very powerful and complex equalization technique that determines the most likely sequence of transmitted data based on the received signal and knowledge of the channel.

A chart of different equalization techniques is given below in Figure 3.1. The techniques and algorithms covered in this thesis are in boldface. The adaptive algorithms shown will be covered in chapter 4.

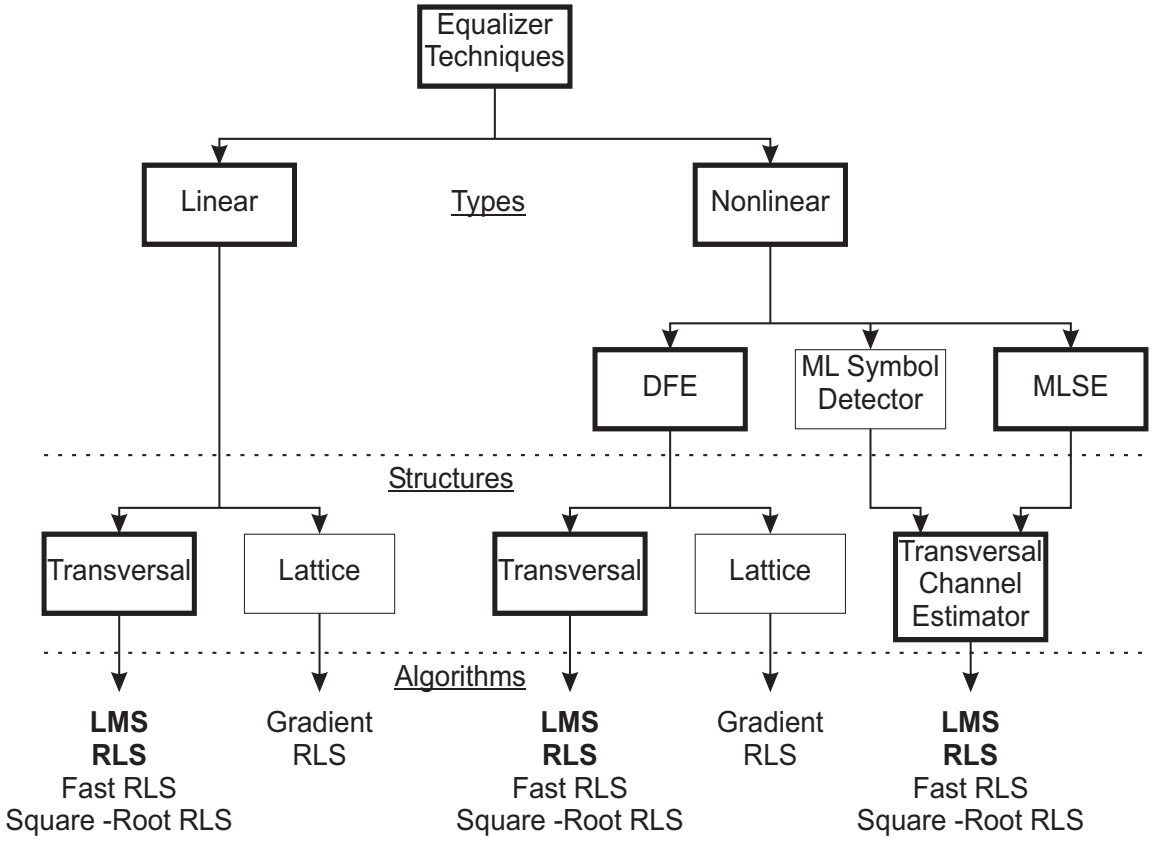

Figure 3.1: Equalizer Types, Structures, and Algorithms [16]

3.1 Linear Equalizers

The linear transversal equalizer and the fractionally spaced equalizer are examples of linear equalization techniques. Linear equalizers use only received signal samples in their calculations and do not use any previously detected symbols.

3.1.1 Linear Transversal Equalizer

The linear transversal equalizer [17] is the simplest equalization technique available. It is made up of a tapped-delay line with tap spacings equal to the symbol rate. The equalizer input consists of the sampled output of the matched filter that precedes the equalizer. All synchronization and timing issues are assumed to be addressed before the signal reaches the input of the equalizer. These samples are placed in a shift register and shifted once every symbol period. The contents of each register is multiplied by a tap gain and added together to form the output of the equalizer. This output is the estimate of the current symbol. This operation can be described by the following equation.

$$z_k = \sum_{k=-N_1}^{N_2-1} c_k r_{n-k} \tag{3.1}$$

In equation 3.1, r_n is the input sequence to the equalizer, c_k is the set of complex-valued tap weights, N_1 is the number of non-causal equalizer taps, and N_2 is the number of causal taps. The total number of equalizer taps is therefore $N_1 + N_2 = N$. The output of the equalizer is represented as z_n. This is the metric used in estimating the value of the original symbol, I_n.

The LTE contains both a causal and non-causal portion. The non-causal portion is used to combat the ISI that occurs when a symbol interferes with previous symbols. This type of ISI results from transmission of data over a dispersive channel or from transmission of data over an obstructed radio channel where the strongest signal element is not the first to arrive. For most radio channels, however, the first received signal element is the strongest. In this case the non-causal equalizer taps are not needed and N_1 is set to zero.

Figure 3.2 shows the structure of a linear transversal equalizer. The T_s blocks indicate a

delay of one symbol period.

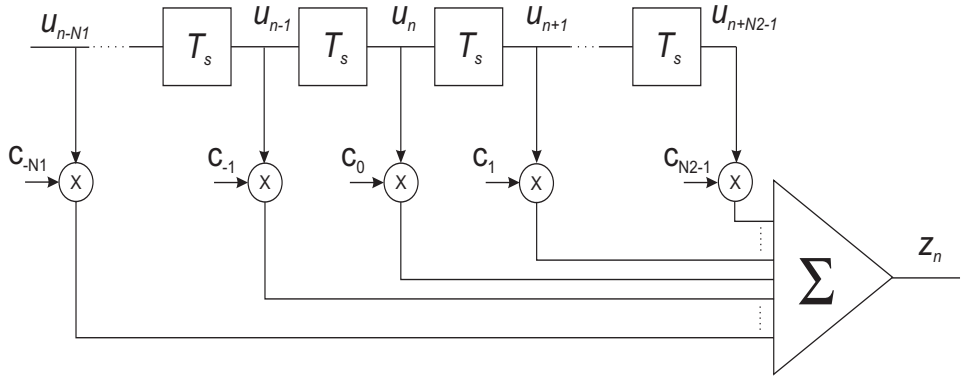

Figure 3.2: Block Diagram of a Linear Transversal Equalizer

The optimization of the coefficients of the linear equalizer is a difficult process. There are many criteria that can be used to choose the coefficients. In digital communications performance is most often measured by the bit error rate (BER). However, BER is a considerably nonlinear function of the filter taps, c_k. This nonlinearity makes optimizing the coefficients based on this performance almost impossible. For this reason the coefficients are usually chosen based on other criteria that are related to the BER. Two performance measures that are often used are the peak distortion criterion and the mean squared error criterion.

3.1.2 Peak Distortion Criterion

Peak distortion refers to the maximum amount of ISI that occurs at the output of the equalizer [2] [17]. Under the peak distortion criterion the filter taps of the linear equalizer are chosen such that this ISI at the output of the equalizer is minimized. An equalizer using this criterion is a type of zero-forcing equalizer. The peak-distortion equalizers, however, use a finite number of taps so that they can be realizable.

The goal of the peak distortion criterion is to set the ISI equal to zero over the range of the filter taps. For the case where the equalizer has L noncausal taps and $L+1$ causal taps the

following should hold true.

$$b_k = \sum_{l=-L}^{L} c_l q_{k-l} = \begin{cases} 1 & k = 0 \\ 0 & k = \pm 1, \pm 2, \ldots \pm L \end{cases} \tag{3.2}$$

Here, b_k is the combined impulse response of the channel and equalizer and $2L+1$ is length of the equalizer.

To find the set of equalizer taps that satisfies this criterion, it is easiest to view equation 3.2 in vector form.

$$\begin{bmatrix} q_0 & \cdots & q_{-L+1} & q_{-L} & q_{-L-1} & \cdots & q_{-2L} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ q_{L-1} & \cdots & q_0 & q_{-1} & q_{-2} & \cdots & q_{-L-1} \\ q_L & \cdots & q_1 & q_0 & q_{-1} & \cdots & q_{-L} \\ q_{L+1} & \cdots & q_2 & q_1 & q_0 & \cdots & q_{-L+1} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ q_{2L} & \cdots & q_{L+1} & q_L & q_{L-1} & \cdots & q_0 \end{bmatrix} \begin{bmatrix} c_{-L} \\ \vdots \\ c_{-1} \\ c_0 \\ c_1 \\ \vdots \\ c_L \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.3}$$

This matrix multiplication can be represented as $\mathbf{Qc} = \boldsymbol{\delta}$. To find the equalizer coefficients that solve this equation, both sides of the equation are multiplied by the inverse of the channel matrix. The coefficients can then be found from

$$\mathbf{c}_{pd} = \mathbf{Q}^{-1}\boldsymbol{\delta}. \tag{3.4}$$

The problem here is that the ISI is not guaranteed to be zero outside the range of the equalizer taps, i.e. $b_k \neq 0$ for $|k| > L$. However, in most cases, if the equalizer is made long enough, this excess ISI should be small and should not hurt the performance of the system.

The peak-distortion criterion works well for applications such as wireline modems where the ISI it the major impediment to the system and the noise is relatively small. However, in wireless systems, the additive noise is a major concern. An equalizer that uses the peak-distortion criterion tends to amplify the noise as well as the signal. This greatly decreases the signal-to-noise ratio. This is why most wireless systems use criteria other than peak distortion that account for the noise component of the received signal. The mean-square

error criterion accounts for the noise when determining the optimal equalizer coefficients. It is discussed below.

3.1.3 Mean Square Error Criterion

Under the mean square error criterion [18], the tap weights of the equalizer are adjusted to minimize the mean-square error between the original data symbol and the output of the equalizer. This error includes both the ISI as well as the additive noise. The mean squared error is defined as

$$J = E\left|I_n - \hat{Z}_n\right|^2 \tag{3.5}$$

where E is the expectation operator . To find the set of equalizer coefficients that minimize the mean squared error for this linear equalizer, the following set of computations are made.

The first step is to find the correlation matrix, $\mathbf{A}$, of the input of the equalizer, u_n.

$$\mathbf{A} = E[\mathbf{u}_n \mathbf{u}_n^H] \tag{3.6}$$

The vector, $\mathbf{u}_n$ is the N-length vector of data in the shift register at sample instant n. The correlation matrix can also be found by using the impulse response vector, $\mathbf{q}$, and the noise spectral density, N_0. First we find the autocorrelation, $\mathbf{x}$, of the impulse response vector.

$$x_k = \sum_l q_l^* q_{l+k} \quad k = -N, \ldots, -1, 0, 1, \ldots, N \tag{3.7}$$

The sampled autocorrelation, $\mathbf{w}$, of the matched filter in the receiver with impulse response, $p(t)$, is also needed.

$$w_k = \int p^*(t) p(t + kT) \quad k = -N, \ldots, -1, 0, 1, \ldots, N \tag{3.8}$$

With these autocorrelation vectors defined, the correlation matrix can now be found.

$$A_{ij} = \quad x_{i-j} + N_0 w_{i-j} \quad i, j = -N_1, \ldots, -1, 0, 1, \ldots, N_2 - 1 \tag{3.9}$$

Next, the cross-correlation vector between the tap inputs, $\mathbf{u}_n$, and the transmitted data, I_n must be computed. This cross-correlation vector is denoted by the symbol $\boldsymbol{\alpha}$ and is of length N.

$$\alpha = E[\mathbf{u}_n I_n] \tag{3.10}$$

Much like the correlation matrix, this vector can be determined from the impulse response vector, **q**.

$$\alpha_i = \quad q^*_{-i} \quad i = -N_1, \ldots, -1, 0, 1, \ldots, N_2 - 1 \tag{3.11}$$

The equalizer tap coefficients that provide the minimum MSE, c_{opt}, are then computed from the correlation matrix and the cross-correlation vector.

$$\mathbf{c}_{opt} = \mathbf{A}^{-1}\boldsymbol{\alpha} \tag{3.12}$$

The MMSE that results from a system using the optimal tap weight vector, $\mathbf{c}_{opt}$, is equal to

$$J_{min} = 1 - \boldsymbol{\alpha}^{*T}\mathbf{A}^{-1}\boldsymbol{\alpha}. \tag{3.13}$$

Theoretically, an equalizer with an infinite number of taps will provided the minimum mean squared error (MMSE). This MMSE is found from

$$J_{min} = T\int_{-1/2T}^{1/2T} \frac{N_0}{N_0 + |Q_{eq}(f)|^2/S_{pp}(f)} df. \tag{3.14}$$

In this equation,

$$Q_{eq}(f) = \sum_n Q(f - n/T) \tag{3.15}$$

and

$$S_{pp}(f) = \sum_n |P(f - n/T)|^2 \tag{3.16}$$

where $Q(f)$ is the frequency response of the system to be equalized and $P(f)$ is the frequency response of the matched filter of the receiver.

For a more in depth look into the calculation of the optimal filter coefficients and how they are derived based on the peak distortion and MMSE criteria see [2] or [1].

3.1.4 Fractionally Spaced Equalizers

In the LTE the taps of the equalizer are spaced at the symbol rate. For a system using a receiving filter that is matched to the transmitted pulse and the channel response, this T-spaced equalizer would be ideal. In practical systems, however, the channel characteristics are not known before hand. In these systems the receiver filter is only matched to the

transmitted pulse shape and the system is synchronized to this filter. Since this method is suboptimal, the performance of the receiver can suffer. Often the frequency response of the channel will have components at frequencies greater than $1/2T$. When the received signal is then samples at a rate of $1/T$, aliasing will occur and these higher frequency components will overlap with the lower frequency components. The sampled signal at the input of the equalizer will have the spectrum

$$Q_T(f) = \frac{1}{T}\sum_n Q\left(F - \frac{f}{T}\right) e^{2\pi(f-n/T)\tau_0} \tag{3.17}$$

where τ_0 is the sampling delay. When the sampling delay is not zero, the effects of the sampling become more severe. The equalizer is only able to operate on this aliased spectrum and not the frequency response itself. The aliased spectrum will often have sharp transitions within it because of the overlapped spectrum. A T-spaced equalizer will have a difficult time compensating for these sharp transitions as it tries to flatten out the overall response.

The fractionally spaced equalizer (FSE) [19] [20] has the same structure as a LTE but spaces its taps at a fraction of the symbol rate. In a digital system the tap spacing is expressed as KT/M, where K and M are relatively prime integers and $K < M$. The received signal is sampled every T/M seconds and passed into the equalizer. The equalizer then operates on every Kth sample. The variables K and M are often chosen such that the received signal spectrum is totally contained within the sampling bandwidth. The input to the equalizer will have the spectrum

$$H' = \begin{cases} H(f) & |f| \leq KT/M \\ 0 & KT/M < |f| \leq /M. \end{cases} \tag{3.18}$$

The equalizer will now be able to operate on the original spectrum of the received signal instead of the aliased spectrum. This allows the equalizer to better compensate for channel distortion. An FSE will also be less affected by changes in the sampling phase because it operates on the full signal spectrum.

In the receiver, the FSE operates without a matched filter preceding it. Instead, the FSE operates as both an adaptive matched filter as well as equalizer. This makes it closer to the optimal receive filter that is matched to both the channel and transmit filter.

Finding the equalizer taps that minimizes the MSE of the output of the equalizer is much like finding the optimal coefficients of the T-spaced LTE. Again, $\mathbf{c}_{opt} = \mathbf{A}^{-1}\boldsymbol{\alpha}$. However, since the FSE does not use a matched filter, $\mathbf{A}$ and $\boldsymbol{\alpha}$ are determined from the sampled input to the receiver, y_n.

$$\mathbf{A} = E[\mathbf{y}_n \mathbf{y}_n^H] \tag{3.19}$$

$$\boldsymbol{\alpha} = E[\mathbf{y}_n I_n] \tag{3.20}$$

The covariance matrix and cross-correlation vector can also be computed from the channel response $h(t)$.

$$A_{ij} = h^*(kT - iKT/M)h(kT - jKT/M) + N_0\delta(i-j) \tag{3.21}$$

$$\alpha_i = h^*(-iKT/M) \tag{3.22}$$

In both equations i and j range from $-N_1$ and $N_2 - 1$.

The only disadvantage to an FSE is that is uses more taps than a T-spaced equalizer to cover the same amount of time. However, it has been found that a $T/2$-spaced equalizer with the same number of taps as a T-spaced equalizer, covering half the time span, performed just as well as and sometimes better than the T-spaced equalizer [1].

3.2 Nonlinear Equalizers

Nonlinear equalizers are equalizers that use more than just the received data in its computations. Many of these equalizers use previously detected symbols in their calculations. Using these detected symbols helps the equalizer to better compensate for the ISI contained by the received signal. Two important nonlinear equalization techniques are *decision-feedback equalization* and *maximum-likelihood sequence estimation*. These two techniques are discussed below.

3.2.1 Decision-Feedback Equalization

The decision-feedback equalizer (DFE) [21] [22] is made up of two parts, the feedforward and the feedback section. The feedforward section is the same as the linear transversal

equalizer discussed above. It can have either T-spaced taps or fractionally spaced taps. The feedback section is also a LTE, but operates on previously detected symbols instead of the received signal. The output of each filter is added to form the equalized signal. By using previously detected symbols the interference caused by these symbols can be removed from the received signal. A diagram of a DFE is shown below.

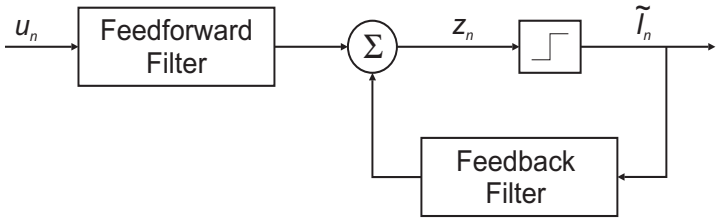

Figure 3.3: A Decision-Feedback Equalizer

The output of the equalizer is expressed as:

$$z_n = \sum_{k=-N_1}^{0} c_k u_{n-k} - \sum_{j=1}^{N_3} b_j \tilde{I}_{n-j} \tag{3.23}$$

where $\tilde{I}_n$ is the previously detected symbol at time n, b_j are the tap coefficients of the feedback filter, and N_3 is the number of taps in the feedback filter. Notice that the feedforward filter uses only noncausal taps. This is because the feedback portion will remove all the ISI caused by the previous symbols; the causal portion of the feedforward filter is not needed. If possible N_3 should be chosen such that the feedback filter includes all the symbols that interfere with the current symbol.

The MMSE criterion is used most often in practical systems to find the optimal taps for both the feedback and feedforward filters. The feedforward filter taps are found using the $\mathbf{c}_{opt} = \mathbf{A}^{-1}\boldsymbol{\alpha}$ calculation similar to that used for the LTE. There are some differences however. The first is that the indices of the $\mathbf{A}$ matrix, i and j, only range between $-N_1$ and 0. Also, the values of x_k that make up the individual elements of the $\mathbf{A}$ matrix are computed as follows.

$$x_k = \sum_{l \leq 0} q_l^* q_{l+k} \quad k = -N_1, \ldots, -1, 0, 1, \ldots, N_1 \tag{3.24}$$

The $\boldsymbol{\alpha}$ vector is computed from

$$\alpha_i = \begin{cases} q_0^* & i = 0 \\ 0 & -N_1 \leq i < 0 \end{cases} \tag{3.25}$$

The coefficients of the feedback filter are then found from the feedforward coefficients, $\mathbf{c}_{opt}$, from

$$\mathbf{b}_{opt} = \mathbf{Q}\mathbf{c}_{opt} \tag{3.26}$$

where $\mathbf{Q}$ is an N_3 x $N_1 + 1$ matrix with elements

$$Q_{i,j} = q_{i-j} \quad i = 1, 2, \ldots, N_3; j = -N_1, \ldots, -1, 0 \ . \tag{3.27}$$

Basically, the decision-feedback equalizers works as follows. The feedforward filter removes the ISI caused by future symbols on the current symbol through linear filtering. The feedback portion subtracts the ISI from past symbols by using the estimates of previously detected symbols. The benefit of the DFE is that it can subtract all the ISI caused by past symbols with a finite number of taps. It only needs as many taps as there are interfering symbols. To achieve the same results with an LTE requires an infinite number of taps.

A DFE with infinite taps in the feedforward filter and the same number of taps as there are past interfering symbols in the feedback section will completely remove all ISI in the case of no noise. When noise is present, the equalizer will be able to achieve a minimum mean squared error (MMSE) equal to

$$J_{min}(DFE) = \exp\left[T \int_{-1/2T}^{1/2T} \ln \frac{N_0}{N_0 + |Q_{eq}(f)|^2 / S_{pp}(f)} df\right]. \tag{3.28}$$

This computation assumes that all bit decisions are made correctly and does not consider the possibility of errors propagating through the feedback loop. For most systems this is a reasonable assumption. When there is a relatively high SNR and the ISI is moderate the probability of error is quite small so the effect of error propagation will be negligible. In the case of low SNR or severe ISI the probability of error is much higher. In these cases error propagation has the potential to greatly degrade the performance of the equalizer. This problem with error propagation through the feedback path is the biggest drawback to decision-feedback equalization.

3.2.2 Maximum-likelihood Sequence Estimation

All the equalization techniques discussed previously have relied on either the peak distortion or the minimum mean square error criterion to equalizer the received data. However, neither criterion provides the minimum symbol error probability that is desired in a digital communication system. The peak distortion minimizes the ISI but amplifies the noise in doing so. An equalizer that minimizes the mean-square error does not amplify the noise as much as peak distortion but only provides minimum error probability when no amplitude distortion is incurred by the channel. However, it is this exact type of channel condition that warrants an equalizer, therefore the MMSE criterion also does not provide minimum error probability.

An equalization technique that provides minimum symbol error in the form of minimum sequence error is maximum-likelihood sequence estimation (MLSE). MLSE is a non-linear equalization technique that differs from the other equalization types in that it does not filter the received data to counter the effects of ISI. Instead, MLSE uses information about the channel to determine all the possible noiseless data sequences that could possibly be received. It then compares these noiseless hypotheses with the noisy received symbol sequence. The sequence that comes closest to the received sequence is then chosen to be the estimated data sequence. The measure or metric that determines how close the sequence hypotheses come to the received signal is given by

$$J_H = \int_t |y(t) - y_H(t)|\, dt \tag{3.29}$$

where the received signal hypotheses are given by

$$y_H(t) = \sum_n a_n h(t - nT). \tag{3.30}$$

In these equations, $y(t)$ is the received signal, $\{a_n\}$ is the set of hypothesized transmitted sequences, and $h(t)$ is the channel response.

The complexity of this metric increases by a factor the size of the symbol alphabet every time a new symbol is received. This is because J_H will hold a value for every possible sequence of $\{a_1, \ldots, a_n\}$ that is possible up until this point. This complexity is given as M^n

where M is the size of the symbol alphabet and n is the number of symbols that have been received. Obviously, this exponentially growing complexity makes this metric computation impractical. A process known as the Viterbi algorithm [23] can reduce this complexity.

The Viterbi algorithm makes use of the fact that in almost all cases the channel response is of finite length, i.e. there exists a t_1 and t_2, with $t_1 < t_2$, such that $h(t) = 0$ for $t < t_1$ and $t > t_2$. With this in mind, the algorithm uses only the current transmitted symbol and the symbols that interfere with it at the receiver in calculating the metric. A metric is computed for each combination of symbols that are part of the current received signal. There will exist M^{L+1} metrics for each received symbol, where L is the number of symbols interfering with the current transmitted symbol. Once a past symbol does not add to the interference of the current received symbol it is no longer used in the metric calculations. After all the metric values are calculated based on the received symbol, the algorithm separates the metric values based on different states. The state is based on the latest L hypothesized values, $\{a_n, \ldots, a_{n-L+1}\}$. This will divide the metrics between the M^L different states with M metric values in each one. The algorithm then finds the maximum metric value for each state and records this value as well as the hypothesized symbol value corresponding to this metric value. These are called the survivor metrics. The other metric values and corresponding symbol values are discarded. The saved metric values will be used in the calculation of the next set of metrics. This process continues until the entire symbol sequence has been received. The complete sequence of symbols that has made it through all the iterations of the algorithm are then given as the most likely sequence. A detailed example of the Viterbi algorithm will be given soon.

The metric calculation as given in equation 3.29 is not realizable for a practical system because of the complexity problem mentioned above and the need for continuous time convolution. An MLSE receiver using the Viterbi algorithm and discrete convolution is a much more practical solution. Two classic MLSE receivers that operate in the discrete time domain were proposed by Forney [24] and Ungerboeck [25]. The receivers proposed by Forney and Ungerboeck are fundamentally equivalent [26] but differ in their structure. The Forney receiver uses a Euclidean distance calculation in computing the metrics used by the

Viterbi algorithm and requires a noise whitening filter. The Ungerboeck receiver forgoes the whitening filter but uses a modified metric calculation. Both these receivers as well as other MLSE receivers based on these approaches are discussed below.

Forney Receiver

The Forney receiver is made up of a matched filter, a sampler, a whitening filter, and a sequence estimator. A block diagram is shown below.

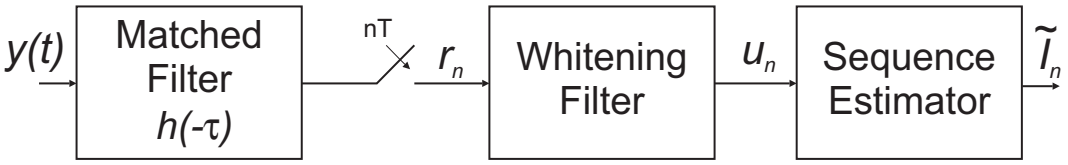

Figure 3.4: The Forney Receiver

In this receiver the matched filter is matched to the channel's response to the transmitted pulse, $h(\tau)$. The matched filter will also filter the noise component of the received signal. This will cause correlation between the noise samples. The whitening filter is used to flatten the power spectral density of the noise and will create a channel response, f_k, that is purely causal. When the signal at the input of the sequence estimator is corrupted by white noise, the standard Euclidean distance metric may be used by the sequence estimator. The sequence estimator finds the most likely sequence from the set of $\{a_n\}$ that maximizes the following metric.

$$\Psi_H(a_1, a_2, \ldots, a_N) = -\sum_{n=0}^{N} \left| u_n - \sum_{j=0}^{L} f_j a_{n-j} \right|^2 \tag{3.31}$$

This metric finds the Euclidean distance between each received symbol and a set of received symbol hypotheses. The symbol hypotheses are formed by filtering the estimated channel response with each possible set of symbols that could have been transmitted. Because a_n can take on up to M values, the size of the symbol alphabet, the calculation of the metric grows exponentially at M^n with each new received symbol, u_n. For this reason the receiver uses the Viterbi algorithm to limit its complexity. The Viterbi algorithm uses a recursive

equation to calculate each metric. It is given by

$$\tilde{\Pi}_{n-L}(a_{n-L},\ldots,a_n) = \max_{a_{n-L}} \left[\tilde{\Pi}_{n-L-1} - \left| u_n - \sum_{j=0}^{L} f_j a_{n-j} \right|^2 \right]. \tag{3.32}$$

Here, $\tilde{\Pi}_n$, is the set of metric values at time n, and f_j is the overall channel response. This channel response includes the transmit filter, the channel, the receiver matched filter, and the whitening filter.

Example of Viterbi Algorithm using the Euclidean Distance Metric

To better understand how the Viterbi algorithm works the following example is given. In this example the symbol alphabet is binary ($M = 2$). The sequence produced by the transmitter is $I_n = \{1, -1, 1, -1, -1, \ldots\}$. The whitened channel response is given by $f_0 = 0.79$, $f_1 = 0.51$, and $f_2 = 0.34$ ($L = 2$). No noise is added to the system so the received signal is $u_n = \{0.79, 1.30, 1.64, 0.06, \ldots\}$.

The first step that the Viterbi takes is to initialize itself. It does this by calculating the metric for the first L possible symbols.

$$\Pi_1(a_3, a_2, a_1) = -\sum_{k=1}^{3} \left| u_k - \sum_{j=0}^{2} f_j a_{k-j} \right|^2 \tag{3.33}$$

Π_1 has 8 different values at this point. One for each of the possible combinations of $\{a_3, a_2, a_1\}$. For the next metric computation, n will equal 4 and I_1 is no longer an interfering symbol. Therefore the hypothesis of I_1, a_1, no longer needs to be included in the metric calculation. For this reason each combination of a_3, a_2 is scanned through and the value of a_1 that maximizes the metric value is recorded. All the others are discarded. Table 3.1 lists the results of equation 3.33 for this example.

Table 3.1: Metric Calculations

state	Π_1	state	Π_1	state	Π_1	state	Π_1
-1,-1,-1	-8.6	-1,1,-1	-4.3	1,-1,-1	-2.5	1,1,-1	-2.8*
-1,-1,1	-4.0*	-1,1,1	-2.9*	1,-1,1	0.0*	1,1,1	-3.5

The metric values marked with an asterisk are the maximum values for each set of $\{a_3, a_2\}$. These metric values are saved for the next step along with the corresponding value of a_1.

The next step is to include the next received symbol, u_4. The algorithm will take the current metric and add to it the two new distance calculations. This will form eight new metrics.

$$\tilde{\Pi}_2(I_4, I_3, I_2) = \tilde{\Pi}_1(I_3, I_2, I_1) - \left| u_4 - \sum_{j=0}^{2} f_j I_{4-j} \right|^2 \tag{3.34}$$

Once these metrics are computed, the four different combinations of $\{I_4, I_3\}$ are scanned through, and the value of I_2 that maximizes that particular set is recorded. The value of I_1 that goes along with that particular value of I_2 is kept as well.

This process continues for the remainder of the received data.

The progression of the Viterbi algorithm is often viewed in form of a trellis. The trellis indicates which states survive after each symbol is received. Figure 3.5 shows the trellis diagram for the example given above.

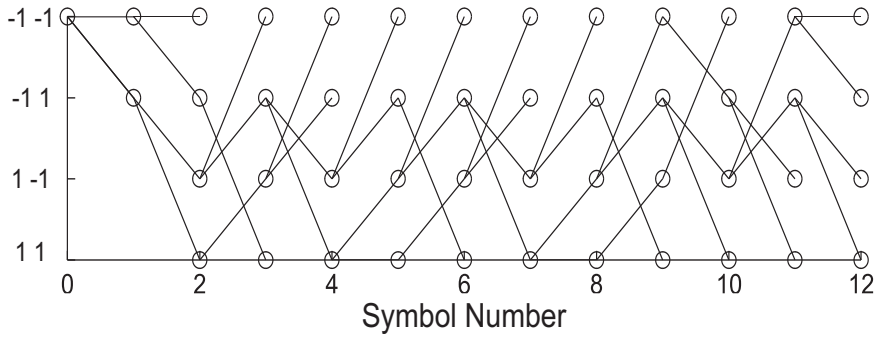

Figure 3.5: The MLSE Trellis

In theory the MLSE equalizer only produces a most-likely sequence estimate after all the symbols have been received. This will introduce a substantial delay into the system and requires a great deal of memory. In most communication systems this delay and memory requirement is not tolerable. For most signals, the Viterbi algorithm will converge to one sequence estimate after a few more symbols are received. In most cases the maximum delay will be around $5L$ symbol periods. In the example given above this means that a guess on

I_1 can be made when u_{11} is received. All the metrics should hold only one value of a_1 at this time. This value is given as the estimate of I_1.

Ungerboeck Receiver

Another of the classic MLSE receivers was developed by Ungerboeck. It is shown below.

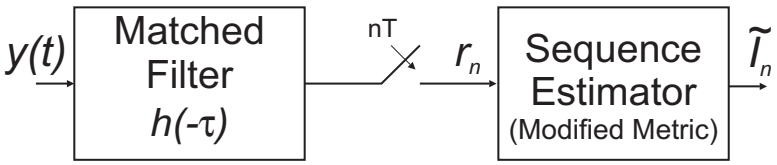

Figure 3.6: The Ungerboeck Receiver

Ungerboeck's receiver differs from Forney's in that it does not use the whitening filter before the sequence estimator. Without the whitening filter, the noise entering the sequence estimator will be correlated or "colored". Also, the channel response, $h(t)$, may contain noncausal elements. For these reason the metric used by the sequence estimator will differ from that used in the Forney receiver. This modified metric is expressed as

$$\Phi_H(a_1, a_2, \ldots, a_N) = \sum_{n=1}^{N} \mathrm{Re}\left[a_n^* \left(2r_n - s_0 a_n - 2\sum_{j=1}^{L} s_j a_{n-j}\right)\right] \tag{3.35}$$

where the sequence $\{a_n\}$ that maximizes the metric is the most likely one. The sampled overall response, s_k, is made up of the transmit filter, the channel, and the received matched filter.

$$s_k = h^*(-\tau) * h(\tau)|_{\tau=kT} \tag{3.36}$$

Because the receiver matched filter is matched to the channel response, the overall response will be symmetric about zero. However, only the causal elements are needed in the metric calculation.

Much like in the Forney receiver this metric calculation grows exponentially. Again, the Viterbi algorithm is used to keep the number of metrics to a manageable level. The algorithm is the same as it was with the Forney receiver except that the metrics are calculated

differently. The iterative metric calculation is given by

$$\tilde{\Gamma}_{n-L}(a_n, \ldots, a_{n-L}) = 2\ \mathrm{Re}(a_n^* r_n) + \max_{a_{n-L}} \left[\tilde{\Gamma}_{n-L-1} - a_n^* s_0 a_n - \mathrm{Re}(a_n^* \sum_{j=1}^{L} s_j a_{n-j}) \right]. \quad (3.37)$$

The main benefit of the Ungerboeck receiver over the Forney receiver is that it requires no squaring operation in its metric calculation [27]. Instead it only requires only a few complex multiplications which are simpler operations than the squaring. Another advantage of the Ungerboeck receiver is that it does not require a whitening filter which will add more complexity to the receiver.

MLSE Receivers for Wireless Systems

The two classical receiver discussed above are optimal for theoretical systems. However, these receivers do not directly fit into the typical wireless communication system model. In a practical wireless communication system the matched filter at the input of the receiver is most often matched only to the transmit filter, not the channel response to the transmitted pulse. For these reasons the receivers used in wireless communication systems are often modified versions of the classical MLSE receivers [27].

One such receiver is based closely on the Ungerboeck receiver. This modified receiver splits the matched filtering operation into two parts. The first part is made up of a filter matched to the transmit pulse. This matched filter can be either a continuous filter or a discrete filter that operates at a sample rate much higher than the symbol rate. The output of this filter is then sampled at the symbol rate and passed to a filter with symbol-spaced taps. This filter matches the overall channel response, q_k. The combination of the filter matched to the transmitted pulse and the filter matched to the channel response is a very close approximation of the single matched filter used in the Ungerboeck receiver. A diagram of this receiver is shown in Figure 3.7.

Other than using two separate filters to act as one matched filter, this receiver works exactly the same as the classic Ungerboeck receiver.

Another type of MLSE equalizer, known as the *direct-form receiver* [27], is based loosely on

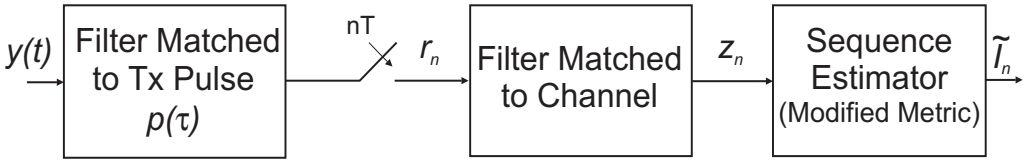

Figure 3.7: The Modified Ungerboeck Receiver

the Forney approach. In this version, only a filter matched to the transmit pulse is used, no other matching is attempted. Also, the Euclidean distance metric is used by the sequence estimator as shown below. This metric is given by

$$\Phi_H = \sum_{n=0}^{N} - |r_n - \hat{r}_n|^2 \tag{3.38}$$

where

$$\hat{r}_n = \sum_{j=-L_1}^{L_2-1} q_j a_{n-j}. \tag{3.39}$$

In this equation, $\mathbf{q}$ is the overall channel response vector and a_n is the hypothesized symbol. The limits L_1 and L_2 represent the number of precursors and postcursors contained in the channel response, respectively.

A diagram of the direct form of the MLSE receiver is shown below.

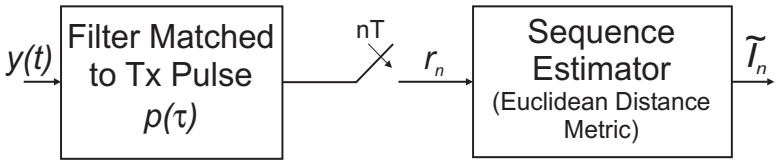

Figure 3.8: The Direct Form Receiver

This approach is suboptimal when compared to the theoretical approach proposed by Forney because the receiver does not match the incoming signal completely. Also, the noise component of the signal at the input to the sequence estimator will be correlated because of the matched filter. When the Euclidean distance metric is used, the noise should be white for optimal performance.

Though this is not an optimal receiver, it still performs very well in a wireless system. The use of a filter matched to the transmit pulse provides some performance increase over no matching at all. Also, when a square-root raised cosine pulse shape is used by the system, the correlation of the noise caused by the matched filter is very small, and affects the performance of the sequence estimator very little.

The advantage to using this equalizer, is that it provides good performance with relatively simple complexity. Removing the filter matched to the channel greatly simplifies the receiver. The only disadvantage is the extra computational complexity of by the squaring operation used by the metric calculation.

3.3 Simulation of Equalization Techniques

To help readers further understand the equalization techniques discussed in the previous sections, several MATLAB simulations have been included with this chapter. These programs simulate a communication system corrupted by ISI and demonstrate how well each of the different equalization techniques compensate for these types of channels. The following sections will discuss the contents of each of these simulation programs and how they operate. These descriptions will include examples of the programs used to set the parameters for the different equalizers and examples of the many different types of graphical output that can be displayed.

All of the simulation programs model a BPSK communication system utilizing an equalizer to compensate for ISI. It is modeled after the system shown with the previous chapter in Figure 2.1. All four of the different simulation programs will use this system design with a few exceptions. First, the FSE simulation does not require the matched filter in the receiver so this block will be left out. Also, the MLSE equalizer makes symbol decision internally so the decision device after the equalizer will not be needed in this simulation.

The simulation programs included with this chapter will use time-invariant channels whose impulse response is known by the receiver. The user will be able select from ten different channel models during the preprocessor program of each simulation. These channel models

are the same as those given with the simulation from the Chapter 2. These channel models are used so that the simulations are simple and straightforward. More complex equalizers that operate on unknown, time-invariant channels or unknown, time-variant channels, will be discussed in the next chapter.

3.3.1 Simulation of a Linear Transversal Equalizer

The first simulation included with this chapter models a wireless BPSK system employing a LTE. It uses the same system model mentioned above. The simulation is the same as the one of the wireless communication system given previously until it reaches the equalizer stage. The equalizer block filters the data just as the other blocks did. However, it needs to compute the filter coefficients that minimize the mean-squared error based on the channel conditions.

The equalizer uses the function **ltetaps.m** to compute the equalizer coefficients. This function takes the sampled channel response, q_k, the autocorrelation of the matched filter response, w_k, and the noise spectral density, N_0, to compute the optimal equalizer coefficients. The function uses the formulas given in Equations 3.9, 3.11, and 3.12 to do this.

The overall channel response is found as it was in the **wsysproc.m** program, by convolving the response of the pulse-shaping filter, the channel filter, and the matched filter. The autocorrelation of the matched filter response, w_k is found by using the convolution command and is represented as **W**.

$$\mathbf{W} = \mathbf{conv}(\mathbf{MF}, \mathbf{fliplr}(\mathbf{MF})) \tag{3.40}$$

The **fliplr** command is used to flip the response vector so the autocorrelation is computed instead of the convolution. The **W** vector is then sampled at the correct time instances to provide w_k, labeled **W2** in the code. The noise spectral density is found from the complex noise vector, **noise**.

$$\mathbf{N0} = \mathbf{0.5} * (\mathbf{noise} * \mathbf{noise}') / \mathbf{length}(\mathbf{noise}) \tag{3.41}$$

All these computations are made in the main simulation file, **lteproc.m**. Once all these values are found, they are passed to the **ltetaps.m** function. This function is used to find

the optimal equalizer coefficients. It begins by finding the autocorrelation of the overall channel response.

$$\mathbf{x} = \mathbf{conv}(\mathbf{q}, \mathbf{fliplr}(\mathbf{q})) \tag{3.42}$$

This vector along with the matched filter autocorrelation vector, **W**, and the noise spectral density are used to create the correlation matrix, **A**, and the crosscorrelation vector, **alpha**, based on Equations 3.9 and 3.11 . The optimal tap coefficients can then be easily calculated from these values using the **inv** command that is used to invert matrices.

$$\mathbf{copt} = \mathbf{inv}(\mathbf{A}) * \mathbf{alpha} \tag{3.43}$$

This vector of optimal equalizer coefficients is then passed back to the processor program. In the processor this vector is used to filter the received data.

$$\mathbf{z} = \mathbf{conv}(\mathbf{copt}, \mathbf{r2}) \tag{3.44}$$

This output vector, **z**, is then shifted to account for the delay caused by the noncausal taps of the equalizer. It is then passed to the decision device which determines symbol values from the vector. A BER is computed by comparing the symbol estimates to the original data. It is then displayed on the screen as **BEReq**. The BER for a system with no equalizer is also computed. It is given as **BERnoeq**. Also included is the theoretical BER for a system with the same E_b/N_0 but no ISI. It is displayed as BERnoisitheory.

Preprocessor/Postprocessor

The preprocessor portion of this simulation is given by **ltepre.m**. It prompts the user for information such as the number of symbols to simulate, the E_b/N_0 ratio, and the number of equalizer taps to use. An example of the execution of this program is shown below.

```
>> ltepre
Enter number of symbols to simulate [1000]:  500
Enter Eb/N0 value [8]:
Enter number of samples (5-20) per symbol [10]:  10
```

```
Enter number of causal equalizer taps [4]:  10
Enter number of noncausal equalizer taps [2]:
Enter the Number (1-10) of the Channel to Simulate [1]:  1
>>
```

Included with the prompt for each parameter value is a default value enclosed in parenthesis. These values can be selected by just pressing the return key when prompted for the parameter value. These default values are just suggestions and are not absolute. Users should feel free to choose whatever reasonable values they wish.

The postprocessor program for this simulation is **ltepost.m**. When this name is entered at the command prompt the menu shown in Figure 3.9 will be displayed.

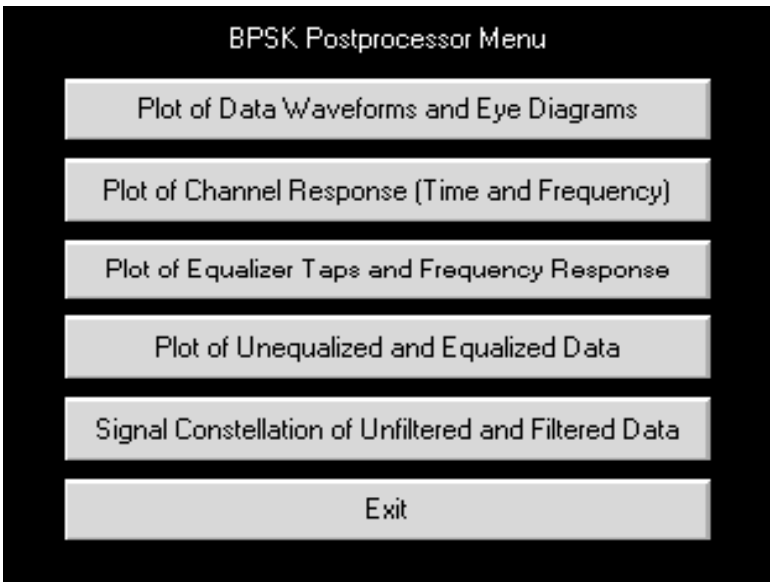

Figure 3.9: Postprocessor Menu for Simulation of LTE

Each menu choice represents a graph or set of graphs that illustrates the requested set of data. For example, the bar labeled *Plot of Data Waveforms and Eye-Diagrams* generates plots of the transmitted and received data waveforms and as well as the eye diagrams of this data. Two of the most interesting plots are the scatter diagrams of the unequalized and equalized data. These plots are shown in Figures 3.10 and 3.11.

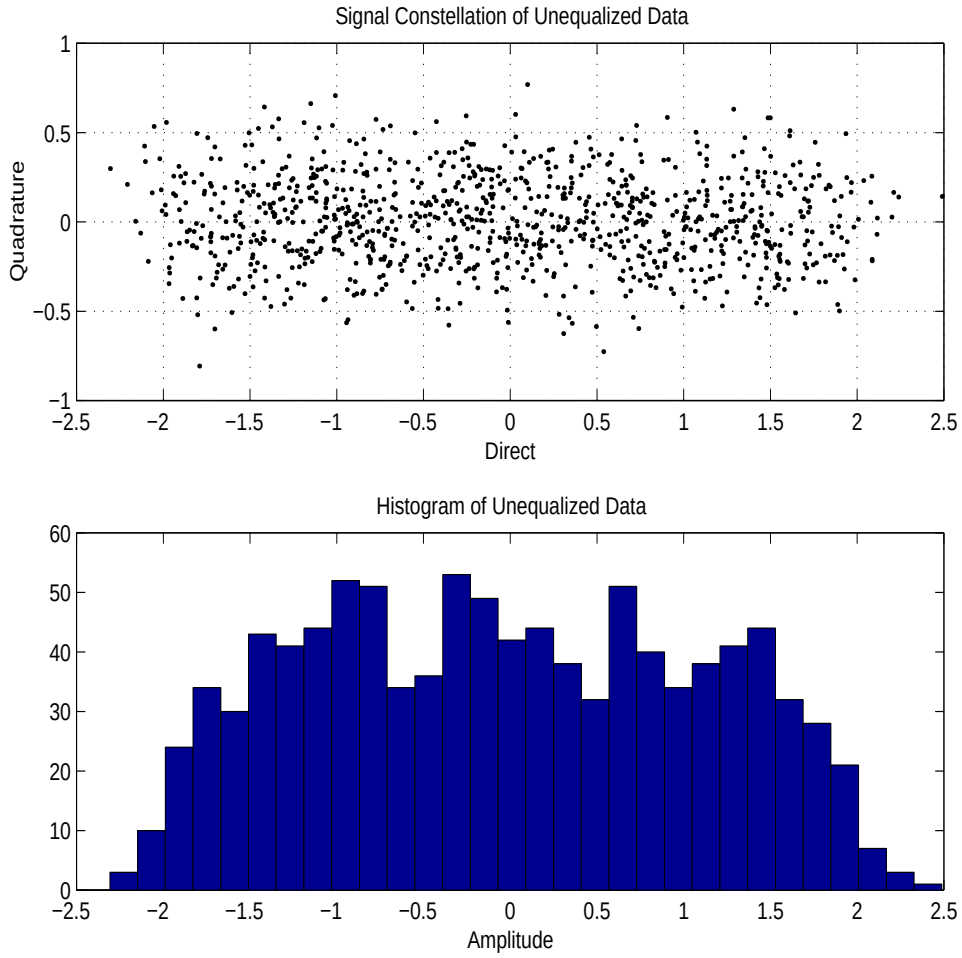

Figure 3.10: Scatter Diagram of the Unequalized Data for a LTE ($E_b/N_0 = 8$ dB, Channel #1)

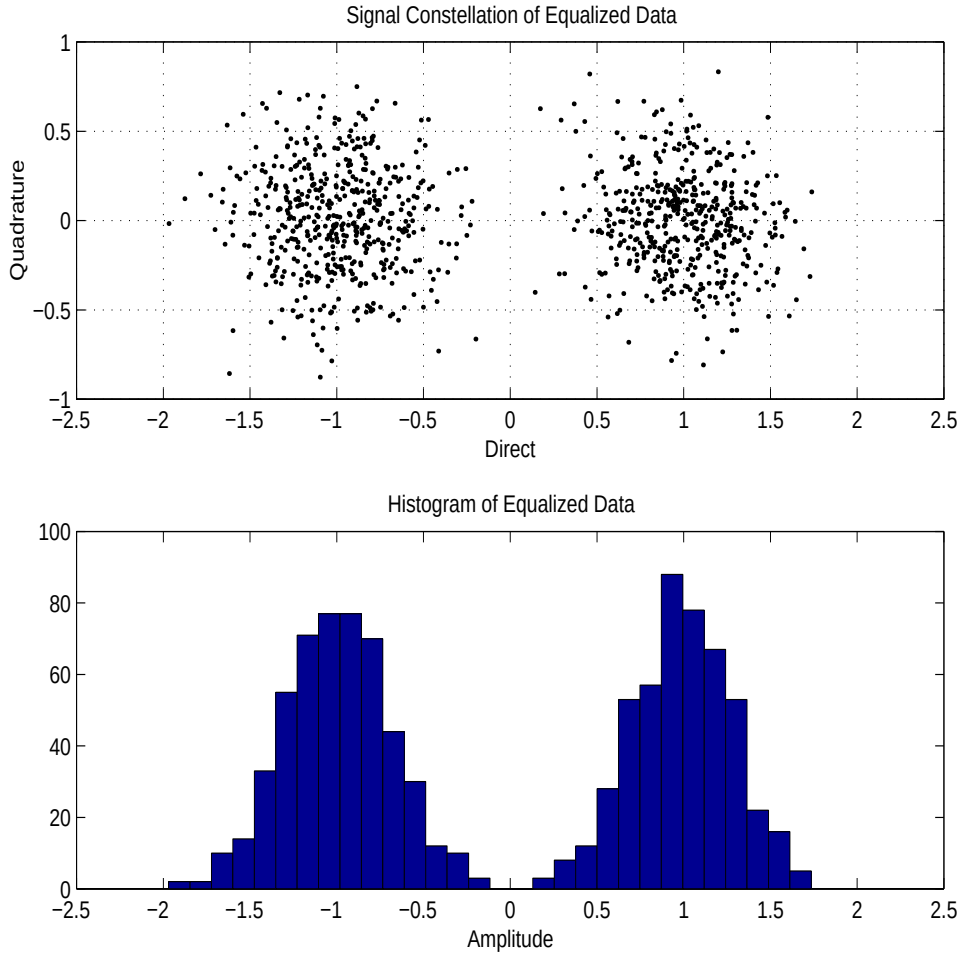

Figure 3.11: Scatter Diagram of the Equalized Data for a LTE

Each of these figures contains two plots. The top plot is the scatter diagram of the signal. A scatter diagram plots the direct part of the signal versus the quadrature portion. In this case the quadrature portion of the signal comes from the additive white Gaussian noise. The scatter diagram gives a good idea of how closely the received signal matches the transmitted signal and how badly the distortion and ISI added by the channel has affected the signal. The bottom plot contained in the figure shows a histogram of the direct part of the received symbol. It illustrates the distribution of this signal.

Figure 3.10 displays a scatter diagram and histogram of the sampled output of the receiver's matched filter. This is the received data before it is equalized. In this case the data appears as one large cloud with no separation between the two possible symbols. The histogram illustrates this fact very clearly. The distribution is approximately uniform across most of the values while dropping off at the edges. These plots indicate how the transmitted data has been spread apart by the ISI caused by the channel as well as corrupted by the noise.

Figure 3.11 displays the scatter diagram and histogram for the data coming out of the equalizer. In this case the scatter diagram displays two distinct clouds centered around the two possible symbols, ± 1. The histogram also shows two distinct clusters of data centered around the two possible symbols. The data distribution appears Gaussian in both cases indicating that most of the distortion caused by the ISI has been removed and only the AWGN is corrupting the signal. The contrast of these figures demonstrates the benefit of using an equalizer in the receiver of a communication system corrupted by ISI.

There are several other possible plots that are available with the LTE postprocessor, but to conserve space they will not all be shown here. It is left up to the user to experiment with the different possible settings and to view the available plots.

3.3.2 Simulation of a Fractionally Spaced Equalizer

The simulation of a Fractionally Spaced Equalizer is very similar to that of the LTE. It models the same BPSK system that was used in the LTE simulation. The main differences between the two are that the FSE simulation does not include a receiver matched filter, the

equalizer operates at a higher sampling rate, and the equalizer taps are determined using a different method.

The preprocessor of the FSE simulation is called **fsepre.m**. It is very similar to the preprocessor of the LTE. An example of the preprocessor is shown below.

```
>> fsepre
Enter number of symbols to simulate [1000]:
Enter Eb/N0 value [8]:
Enter number of samples (5-20) per symbol [12]:  10
Enter number of causal equalizer taps [8]:
Enter number of noncausal equalizer taps [7]:
Enter the number of taps per symbol [4]:  3
Number of taps per symbol must be a multiple of number of samples per
symbol
Enter the number of taps per symbol [4]:  2
Enter the Number (1-10) of the Channel to Simulate [1]:
>>
```

One requirement in setting the parameters for the FSE simulation is that the number of taps per symbol is a multiple of the number of samples per symbol, otherwise the preprocessor will display an error and ask for a new value like in the example above. Default values are also included with this preprocessor. They can be chosen by pressing return when prompted for a value.

The processor program is also similar to the LTE processor. It models a BPSK system using the block-by-block simulation method. It begins by finding the combined response, **H**, of the matched filter, **MF**, and the channel, **G**, using the **conv** command. It then samples this response at the correct points at the rate of M samples per symbol that was selected in the preprocessor. This sampled response is given by **H2**. It then normalizes the overall response such that $\sum_k |\mathbf{H2}(\mathrm{k})|^2 = 1$. The normalized sampled impulse response, the selected noise power, and other parameters are then passed to the program **fsetaps.m**.

This function uses Equations 3.21 and 3.22 to determine the correct equalizer taps. These taps are passed back to the main program.

Once these responses are found, a vector of random bipolar data is created. This data is convolved with the overall response and noise is added. At the receiver the signal is sampled at a rate of M samples per symbol. This sampled response is then filtered with the equalizer taps. The output of the equalizer is then sampled such that there is one sample per symbol. These samples are then passed to a hard limiter to determine the estimated symbol value. A bit error rate is then determined from these estimated values.

A simulation of a typical BPSK system that uses a matched filter in the receiver but does not contain an equalizer is also included in this simulation so that its output can be compared with the output of the system utilizing an equalizer. The different outputs can be analyzed and any performance gains can be observed.

The postprocessor of the FSE simulation is also very similar to the one used by the LTE simulation. It includes many of the same data plots including plots of the transmitted and received data and scatter plots of the equalized and unequalized data. One of the most interesting plots is one where the combined response of the equalizer and channel is displayed. An example of this is shown in Figure 3.12.

This top graph of this figure shows the overall response of the channel and equalizer and the bottom graph displays the sampled version of this response. These graphs reveal how the FSE works. The FSE tries to model the optimal receive filter: one that matches the channel response of the transmitted signal shape. With this optimal filter in place, the combined response will have zeros at each sampling point except at time zero. This is illustrated with the top graph. The bottom graph shows the response after the signal is sampled. In this case the response is a delta function at time equal to zero. This indicates that all the ISI has been canceled and the only distortion in the noise. This is the desired response of the system.

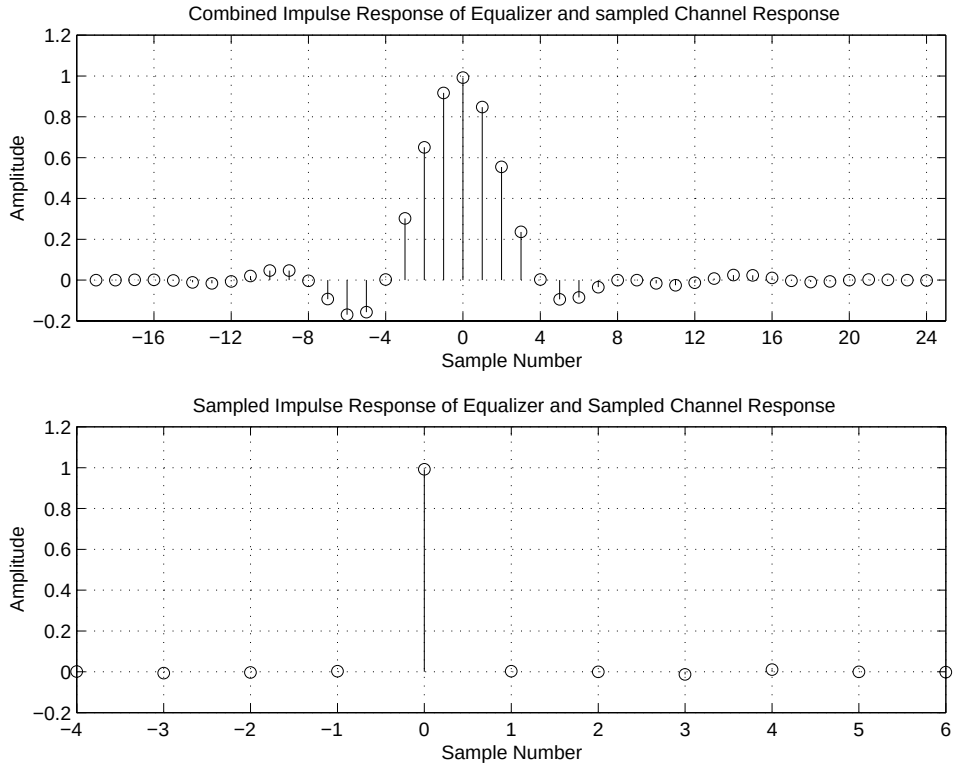

Figure 3.12: Combined Response of FSE and Channel, (Channel #1)

3.3.3 Simulation of a Decision-Feedback Equalizer

The next simulation included with this chapter is a simulation of a BPSK system utilizing a decision-feedback equalizer. This simulation is very similar to the LTE and FSE simulations discussed above.

The preprocessor included with the DFE simulation prompts the user for many of the same parameters that the other preprocessors have asked for such as the number of symbols to run and the E_b/N_0 ratio. In addition to these items the DFE processor will also ask for the number of taps in the feedforward filter and the number of taps in the feedback section. The feedforward filter is set up such that there is one causal tap at time equal to zero. The remaining number of taps will be noncausal taps. The causal part of the channel response will be handled by the feedback filter.

The processor stage of the simulation is based on the standard BPSK system model. The DFE is placed after the matched filter sampler. To find the optimal feedforward and feedback taps for the equalizer, the function **dfetaps.m** is used. This function takes the channel response, the noise power, and the number of taps in each filter and computes **C**, the optimal feedforward taps, and **B**, the optimal feedback taps. It uses equations 3.24,3.25, and 3.26 to computes these vectors.

One difficulty in implementing the DFE is that it does not a operate as a linear filter like the LTE and FSE do. Instead of using the **filter** or **conv** commands in MATLAB, a small algorithm is created to perform the equalization. The algorithm is based on equation 3.23 and is made up of two operations. In the first part a block of data is passed into a register and is then vector multiplied by the feedforward coefficients. This is implemented using the following MATLAB code:

$$\mathbf{z(n) = r2(n + N1 : -1 : n) * C.} \tag{3.45}$$

Here, **z(n)** represents the current output of the filter, **r2** is the block of data being filtered, and **C** is the optimal feedforward coefficients.

In the next part of the algorithm, the ISI caused by past symbols is removed by taking the

value computed in part one and subtracting from it past symbols that have been filtered by the optimal feedback taps. This is implemented using:

$$\mathbf{z(n) = z(n) - Iest(n - 1 : -1 : n - N3) * B.} \tag{3.46}$$

Here, **Iest** is the block of previously estimated output symbols and **B** is the vector of optimal feedback taps. The output of the equalizer, **z(n)**, is then passed to a decision device used to estimate the current symbol, **Iest(n)**. This estimate will then be used in equalizing the next few symbols.

The postprocessor of the DFE is also very similar to the other postprocessors. One plot that is specific to the DFE postprocessor is one that displays the combined response of the channel and the feedforward filter. An example of this plot and a plot of the equalizer coefficients is shown in Figure 3.13. In the plot of the equalizer taps, the positive symbol numbers represent the feedback taps while the rest represent the feedforward taps. A plot of the sampled channel response is shown in Figure 3.14.

These two plots are very good at illustrating how the two filters in the DFE work together to remove the ISI corrupting the signal. Figure 3.14 shows that the channel response has both causal and noncausal elements to it. The convolution of this response and the feedforward filter response of the DFE is displayed in the top plot of figure 3.14. In this case the noncausal portion of the channel response has been greatly reduced. Only the causal elements are a factor now. These elements are removed by the feedback portion of the DFE. The taps of this filter, shown in the top graph of Figure 3.13, match up almost exactly with those of the causal ISI terms of the combined response. By subtracting out these terms, the causal ISI will be removed. In other words, the feedforward section of the DFE compensates for any ISI caused by future symbols while the feedback portion removes any ISI caused by past symbols.

3.3.4 Simulation of a Maximum-likelihood Sequence Estimator

Like the simulation of the other three equalizer, the MLSE simulation also models a wireless BPSK system. In this case an MLSE equalizer is utilized by the system to compensate for

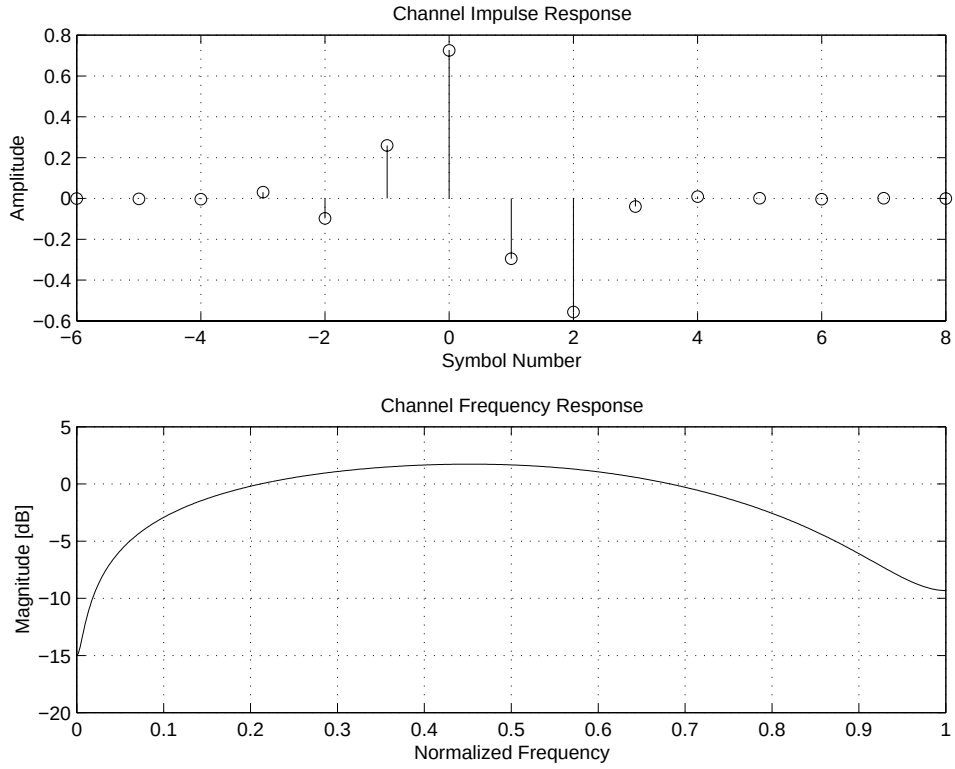

Figure 3.13: Plot of the Sampled Channel Response and Frequency Response (Channel #1)

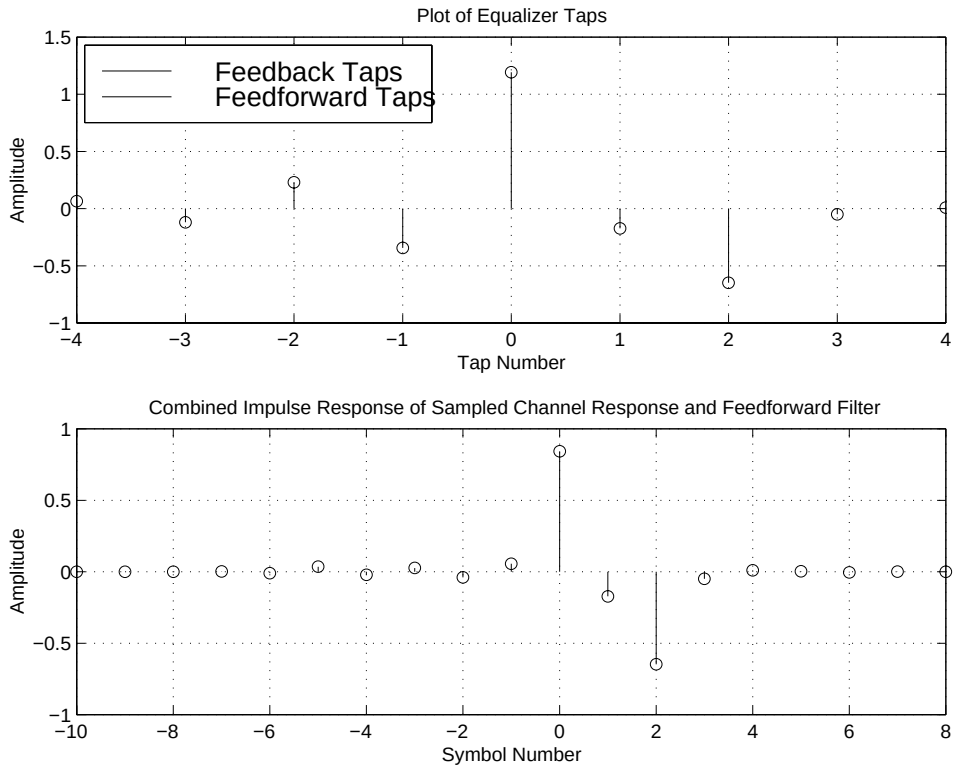

Figure 3.14: Plots of the DFE Equalizer Coefficients and the Combined Response of the FF and Channel

ISI. This equalizer will require different input parameters and will provide output displays different from the other simulations. The structure of this system differs slightly form the others because the decision device that estimates the transmitted symbols is internal to the MLSE equalizer and is not needed in the main system.

The wireless BPSK model that is used to simulate this system contains transmit and receive matched filters that are symmetric, centered around zero, equal to each another, and form a Nyquist pulse when convolved with one another. For this reason the direct form MLSE receiver will be used. This type of MLSE receiver was discussed in Section 3.2.2. It uses the Euclidean distance metric to compute the decision metrics which makes it fairly easy to understand. In Section 3.2.2 the direct form receiver uses a channel estimator to determine the time-varying channel response. The purpose of this simulation is to help the user better understand the fundamentals of MLSE equalization. For this reason the system will be simplified; the simulated channel will be time-invariant and will be known by the receiver. This will keep the simulation simple and comprehensible.

In theory, an MLSE receiver works on one continuous stream of data and determines metrics for all the different possible paths. This is not possible in a practical system because of computation and memory limitations. Therefore, in a practical system the MLSE equalizer must be designed so that is more efficient and simple. MLSE receivers are often designed one of two ways in order to meet the memory and computation limitations. The first will remember the data path that matches up with each possible state. However, once these paths grow to a length of $5L$ symbols, all paths should lead back to the same symbol estimate. In the first type of MLSE receiver, this common symbol will be given as the estimated symbol for this symbol period and will then be forgotten by the equalizer. The output of the equalizer will therefore be delayed $5L$ symbols compared to its input. This method requires the equalizer to only remember a number of symbols equal to $5L$ times the 2^L number of different states. This method is efficient and does not require any overhead symbols in its calculations, but is complex and can be difficult to implement.

An alternative method is to break the data into frames of a preset length. The MLSE equalizer will then operate on one frame of data at a time. One problem that develops from

this method is how to pick the correct path once the frame of data ends. Just picking the path that maximizes the metric is not always the best choice. A solution to this problem is to include extra symbols at the end of each frame that are known by both the transmitter and receiver. These extra bits will force the MLSE equalizer into a known state at the end of each frame. The most likely path will then be the one that ends with this state.

The second method discussed above is the one used in this MLSE simulation. It was chosen because many wireless communication standards already organize their data into frames and provide overhead symbols between them. It was also chosen because it is simple and is easier to implement than the first method.

Preprocessor/Processor/Postprocessor

The preprocessor of this MLSE simulation requests most of the same information as the preprocessor programs of the other equalizers. Two extra parameter settings this preprocessor requests are the number of symbols in each data frame and the number of ISI terms to account for, L. The frame length parameter is self-explanatory, but the number of ISI terms requires some explaining. Basically, before the MLSE equalizer begins processing the received data it needs to know the channel response. However, because the size of this channel response determines the complexity of the equalizer, i.e. the number of states, using the entire sampled channel response may not be the best choice. The greater the complexity of the equalizer, the longer it will take for the simulation to run. Since most of the ISI is caused by the one or two past symbols, it is better to only account for these and to ignore the smaller more distant ones. Also, the MLSE receiver only works with causal ISI terms. This is because the most severe of the ISI terms are the causal ones and because this receiver is easier to implement.

The main processor stage of this simulation consists of the same BPSK model that was used in all the other equalizer simulations. The main difference is that the data is divided into frames and extra symbols are appended onto the ends of these frames before the data is passed to the channel. This extra data is a set of symbols equal to -1. The number of extra symbols is equal to the number of ISI terms, L, that are accounted for.

Once the data is received, passed through the receive matched filter, and sampled, it is passed to the equalizer block. In the equalizer block, the data is once again divided into frames. Each frame and the truncated channel response, **F**, are then passed to a function called **mlse.m** that takes this data as its input and produces all the different possible paths as well as the most-likely path as its output.

The **mlse.m** function follows the same method as the example given in Section 3.2.2. The first step is to initialize each of the possible states and create a metric value for each one. How these metrics are calculated will be discussed shortly. Once all these values are determined, a vector called **Iconv** is created which represents all the different possible received values based on the given channel response, **F**. The function then begins to loop through each received value and create metrics based on this value and the different possible values. This is performed in MATLAB using the following code which is based on Equation 3.34.

$$\mathbf{PM} = \mathbf{Gam2} - (\mathbf{r(k)} - \mathbf{Iconv})\hat{}\,\mathbf{2}; \tag{3.47}$$

In this equation, **PM** is the matrix of metric values for each state, **r(k)** is the current received symbol value, and **Gam2** is a matrix of the previous metric values. The matrix **Gam2** is made up of two columns of identical metric vectors. These metrics are determined from the previous values of **PM**.

Once the matrix **PM** is determined, the appropriate values are kept or discarded by determining the maximum metric value in each column. Each column holds the possible values held by each state. It is organized much the way Table 3.1 is. To determine what the maximum value in each column is and what row holds this value, the following command is given:

$$[\mathbf{Gam}, \mathbf{idx}] = \mathbf{max}(\mathbf{PM}); \tag{3.48}$$

This command sets the vector **Gamma** to the maximum value in each column and the vector **idx** to the row in which these values fell. The vector **Gamma** is then doubled up to form the **Gam2** matrix and the **idx** vector is then reshaped and appended onto the end of a matrix called **paths** which holds all the possible data paths ending with each state.

This process continues for the entire frame. Once all the data in the frame has been analyzed,

the path that corresponds to the final state of all −1s is chosen as the most likely state since this is the state the equalizer was forced into with the extra data. Both this most likely path and all the different paths that were determined while the algorithm was running are then delivered to the main processor program. All the different paths are provided because they will be used in one of the postprocessor graphs.

The postprocessor of the MLSE simulation provides many of the same plots as the other postprocessors. In addition to plots of the overall channel response and the sampled channel response, the MLSE postprocessor displays a plot of the truncated response, **F**, that is used by the equalizer. A user can compare this response to the total sampled response to determine a good value for the number of ISI terms to account for.

Another plot that is exclusive to the MLSE postprocessor is one that displays a trellis plot of the last frame that was analyzed. An example of this plot is shown in Figure 3.15.

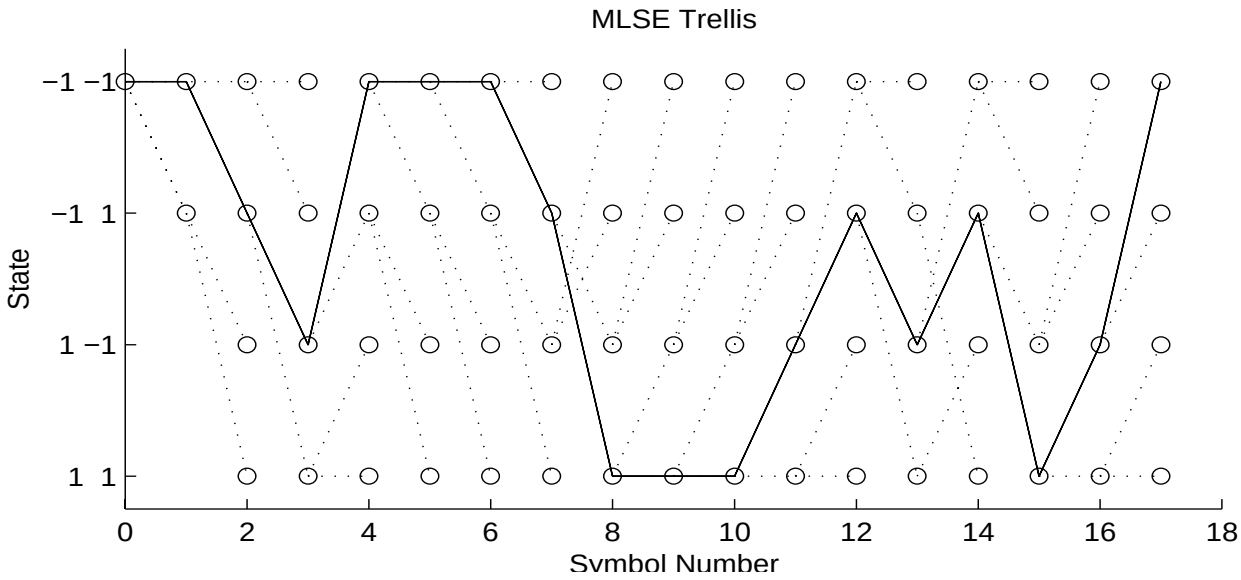

Figure 3.15: An MLSE Trellis

In this plot the dotted lines represent all the different paths that were calculated by the equalizer and the solid line represents the most-likely data path. On the computer the dotted line will be green and the solid line blue. The actual data path is also displayed as a solid red line. However, in this case the estimated path is equal to the actual path so the

red line is not seen. Only when errors occur will the red path be visible.

In this example, the trellis has 2^L states where L equals two and the frame size is fifteen symbols. Two extra symbols were added to force the equalizer into a predefined state. Notice that the estimated path ends in the all negative ones state like it should.

Chapter 4

Adaptive Equalization

The focus of Chapter 3 was on different equalizer structures. During the development and simulation of these structures, it was assumed that the channel was both known by the receiver and was time-invariant. In practical, real-world systems this assumption cannot be made. Wireless channels, in particular, can prove to be extremely volatile in many instances. Every time a receiver is used, different channel conditions exist depending on the environment. This makes exact knowledge of the channel by the receiver impossible. The channel is always changing with time. Movement by objects in the environment, movement of the receiver, and even atmospheric conditions can cause time-variations in the channel response.

For the reasons mentioned above it is necessary to develop equalizers that do not rely on exact knowledge of the channel conditions, but can instead adjust themselves to the channel. Using the received data, the equalizer will adapt itself based on a certain performance index. These equalizers will be able to adjust themselves to all sorts of different channels and should be able to track these channels as they vary with time [17] [28].

In this chapter the concept of adaptive equalization is introduced. Adaptive equalizers use algorithms that determine the best set of equalizer parameters based on the received data. The theory behind these algorithms and their performance will be described in great detail. Also presented in this chapter will be the use of these algorithm with each of the equalization

techniques developed in Chapter 3. Simulations of these different adaptive equalizers in a BPSK system are included with this chapter. These simulations will be presented towards the end of the chapter.

4.1 Adaptive Algorithms

There are many different types of adaptive algorithms that can be used with an equalizer. These algorithms can adapt themselves based on any number of criteria. Two widely used adaption criteria are peak distortion and mean-square error. These criteria describe the type of cost function used in the adaptive algorithm. The cost function is found by comparing the actual output of the algorithm with a desired output.

The performance of these different algorithms are based on a number of factors. One important factor is the algorithm's rate of convergence. This describes how quickly an algorithm can adapt itself to the optimum solution. A fast rate of convergence allows the receiver to make accurate bit decisions more quickly than a slow rate of convergence. A fast rate of convergence also allows for better tracking of quickly changing channel conditions by the equalizer.

Another performance measure is the adjustment error. This indicates how close the adaption criterion, such as the mean-square error, comes to the theoretical optimal value. The smaller the difference, the more accurate the data estimates. The computational complexity another issue to consider when examining adaptive algorithms. Some algorithms require many computations per iteration while other only require a small amount. In applications such as portable radios, where power efficiency and size require that computations be kept at a minimum, algorithm complexity can become a major concern.

Adaptive algorithms based on both the peak distortion and mean-square error criteria are controlled by the prediction error of the equalizer. This error is found by comparing the output of the equalizer with a desired value at each iteration of the algorithm. This desired value can be either a scaled version of the transmitted symbol or some known property of the signal. This error is directly related to the cost function being used. The goal of the

algorithm is to adapt itself to minimize the cost function. The most common method in determining the error is to subtract the output of the equalizer, z_n, from the transmitted symbol, I_n, as shown in the equation below.

$$\varepsilon_n = I_n - z_n \tag{4.1}$$

The problem here is that the desired symbol, I_n, must be known by the receiver. For this to take place, a short sequence of data, known by both the receiver and transmitter, is used to train the equalizer such that it converges to the optimal solution before any data is sent. This set of known data is commonly referred to as a *training sequence* and is used in many adaptive equalization systems. Another technique used in calculating the error signal is to use the receiver's estimate of the transmitted signal as the desired symbol. This mode of equalizer operation is called the *decision-directed* mode. Often this mode will be used after a training sequence has been sent. The training sequence will allow the equalizer to converge to an appropriate state for the current channel conditions, then the decision-directed mode can be used to track small channel variations that occur afterwards. Though incorrect symbol estimates may hurt the performance of the equalizer in this mode, the probability of error should be so small that the few errors that do occur will have little effect.

As mentioned before, the most common cost functions used to adjust an equalizer are peak distortion and mean-square error. While the peak distortion technique is a useful technique and is often used in applications such as wireline modems, it does not work well when applied to wireless applications because of its tendency to amplify noise. That is why most wireless systems employing adaptive equalizers use the mean-square error to adapt themselves. The mean-square error is defined as the average of the magnitude of the error, squared.

$$J = E\left[|\varepsilon|^2\right] \tag{4.2}$$

Two of the most common adaptive algorithms that use the mean-square error cost function are the Least Mean Square (LMS) algorithm and the Recursive Least Squares (RLS) algorithm. LMS is the most widely used adaptive algorithms because of its simplicity, but has a relatively slow convergence rate. The RLS algorithm, on the other hand, is more complex

than the LMS algorithm, but provides a much faster rate of convergence. Both of these algorithms will be discussed in detail below.

4.1.1 The Least Mean Square Algorithm

The LMS algorithm is an iterative procedure that continuously updates a vector of equalizer coefficients [18]. It updates these coefficients based on the mean-square error cost function given in Equation 4.2. This cost function is dependent on the output of the equalizer which is dependent on the tap coefficients. Each vector of equalizer coefficients will have a certain mean-square error associated with it. One such vector will produce the minimum mean-square error. The LMS algorithm attempts to find this desired vector.

The mean-square error performance of the equalizer can be thought of as an N-dimensional bowl-shaped function of the tap coefficients. This is best visualized when there are two coefficients. The value of each tap is on the x- and y-axis, respectively, while the MSE of the equalizer is on the z-axis. The function will look like a three-dimensional parabolic bowl with the minimum mean-square error located at its bottom. Each point on the bowl will have a gradient vector associated with it that points away from the bottom of the bowl. This vector can be computed from the derivative of the MSE with respect to the equalizer coefficients.

$$\mathbf{G}_n = \frac{1}{2}\frac{\partial J}{\partial \mathbf{c}_n} \tag{4.3}$$

The vector of coefficients that minimize the mean-square error can then be found by continuously moving along the bowl in the opposite direction of the gradient vector. The point on the bowl where the gradient vector is zero corresponds to the vector of optimal equalizer taps, $\mathbf{c}_{opt}$.

This technique of using the gradient vector to find the MMSE is called the steepest-decent method. This method is illustrated by the following iterative equation.

$$\mathbf{c}_{n+1} = \mathbf{c}_n - \mu\mathbf{G}_n \tag{4.4}$$

In this equation, the parameter μ is called the step size and is used to control the stability and convergence speed of the algorithm.

The gradient vector can also be found from the cross-correlation between the error and the vector of received data points, $\mathbf{u}_n$, held by the equalizer.

$$\mathbf{G}_n = \frac{1}{2}\frac{\partial J}{\partial \mathbf{c}_n} = -E\left[\varepsilon_n \mathbf{u}_n^*\right] \tag{4.5}$$

However, this cross-correlation vector is only known when the channel is known. For this reason, estimates of the cross-correlation vector are used in place of the actual vector. These estimates are found from the following.

$$\hat{\mathbf{G}}_n = -\varepsilon_n \mathbf{u}_n^* \tag{4.6}$$

Replacing the actual gradient with the estimated gradient in Equation 4.4 produces the recursive equation used by the LMS algorithm.

$$\mathbf{c}_{n+1} = \mathbf{c}_n + \mu\varepsilon_n \mathbf{u}_n^* \tag{4.7}$$

The LMS algorithm is very simple and requires only $2N+1$ operations per iteration. Variations of the algorithm that use only the sign of the error value or the signs of the data symbols or both have also proven to be effective but converges more slowly than the original algorithm.

The LMS algorithm is a feedback system and exhibits the properties of such a system. One common problem of a feedback system is that if the step size is made too large, the algorithm can become unstable and will not converge to the optimal tap vector. For this reason, the step size, μ, should be chosen carefully. It has been shown that an equalizer that uses exact gradient calculations will remain stable if the following condition is met.

$$0 < \mu < \frac{2}{\lambda_{max}} \tag{4.8}$$

Here λ_{max} is the largest eigenvalue of the correlation matrix, $\mathbf{A}$, given by Equation 3.6. This maximum eigenvalue is upper bounded by

$$\lambda_{max} < \sum_{k=1}^{N} \lambda_k \tag{4.9}$$

where

$$\sum_{k=1}^{N} \lambda_k = N(x_0 + N_0). \tag{4.10}$$

In this set of equations, λ_k are the eigenvalues of the covariance matrix, $\mathbf{A}$, N is the number of taps in the equalizer, and $x_0 + N_0$ is the received power. Therefore, the step size should be chosen based on the number of taps in the equalizer and the combined power of the received signal and noise power. It should be noted that these conditions are based on the assumption that the gradient is known by the receiver at each point. When gradient estimates are used, this limit does not guarantee stability. However, it does provide a reasonable to base the step-size on. If the step-size is chosen much smaller than the limit, then stability should not be a problem.

The step size also affects the convergence speed of the LMS algorithm. When a large step size is used, the algorithm makes larger adjustments when updating the filter coefficients. This results in quicker convergence to the optimal filter taps. Smaller step sizes result in slower convergence. However, there is a drawback to using large step sizes besides the stability concern. When estimates of the gradient are used in place of the actual gradient vector, a certain amount of error occurs. The step size parameter determines the magnitude of this error. When analyzing the algorithm, this error is modeled as noise. The variance of this noise is referred to as *excess mean-square error*. The total MSE of the system is then measured by combining this excess MSE with the minimum MSE. There is a tradeoff to be made when selecting a step size. The larger it is, the faster the convergence, but the greater the excess MSE. The smaller the step-size, the longer the convergence time, but the smaller the excess MSE.

The approximate ratio of excess MSE to the minimum MSE is given by

$$\frac{J_\mu}{J_{min}} \approx 0.5\mu N(x_0 + N_0) \tag{4.11}$$

where J_μ is the excess mean-square error. The J_μ value should always be kept much less than the J_{min} value. Therefore, the ratio of the two should be kept at a value much smaller than one.

The convergence rate of the LMS equalizer also depends on the channel itself. The eigenvalue spread of the covariance matrix, $\mathbf{A}$, describes how severe the channel is and how quickly the algorithm can adapt to it. The smaller the ratio of the largest eigenvalue to the smallest eigenvalue, $\lambda_{max}/\lambda_{min}$, the more quickly the algorithm will converge. The convergence of

the algorithm will be much slower if the eigenvalue ratio is large, as is the case for channels with deep spectral nulls.

4.1.2 The Recursive Least Squares Algorithm

The LMS algorithm is a simple algorithm that is easy to implement and has a low computational complexity. However, the algorithm requires a very large convergence time, especially when the eigenvalue spread of the covariance matrix is large. In instances where fast convergence is needed, it is necessary to use adaptive algorithms that are more complex the LMS algorithm. The recursive least squares (RLS) algorithm [29] [30] is another adaptive algorithm that uses the mean-square error as its cost function. It is much more complex than the LMS algorithm, but converges much more quickly.

The derivation of the RLS algorithm is too long and cumbersome to be included in this book. For a detailed description of the algorithm see either [2] or [4]. The basic idea behind the RLS algorithm is that it uses time averages of the received signal in place of the statistical averages used by the LMS algorithm in its computations.

The RLS algorithm begins by initializing the vector of tap weights to zero. It also initializes a matrix used in the algorithm.

$$\mathbf{c}_{-1} = \bar{0} \tag{4.12}$$

$$\mathbf{P}_{-1} = \delta\mathbf{I} \tag{4.13}$$

The initial vector of tap weights, $\mathbf{c}_{-1}$, will have a length equal to the number of taps in the equalizer, N, and $\mathbf{P}_{-1}$ will be an N x N matrix. The constant, δ is a large positive integer.

Once the different values are initialized, the algorithm begins at time $n = 0$ and continues for the entire received data stream. The algorithm works using the following procedure.

First the data is filtered by the current filter taps.

$$z_n = \mathbf{c}_{n-1}^H \mathbf{u}_n \tag{4.14}$$

Next, the error between the desired signal and the computed value is found.

$$\varepsilon_n = I_n - z_n \tag{4.15}$$

Then the $\mathbf{k}$ vector and $\mathbf{P}$ matrix are found that are used to adjust the coefficients.

$$\mathbf{k}_n = \frac{\mathbf{P}_{n-1}\mathbf{u}_n}{\lambda + \mathbf{u}_n^H \mathbf{P}_{n-1}\mathbf{u}_n} \tag{4.16}$$

$$\mathbf{P}_n = \lambda^{-1}\left[\mathbf{P}_{n-1} - \mathbf{k}_n \mathbf{u}_n^H \mathbf{P}_{n-1}\right] \tag{4.17}$$

Finally, the tap weights are updated.

$$\mathbf{c}_n = \mathbf{c}_{n-1} + \mathbf{k}_n \varepsilon_n^* \tag{4.18}$$

Equations 4.16 and 4.17 both use the constant, λ, in their calculations. This constant is known as the *forgetting factor* of the algorithm. It determines how much past values affect the current value. It is usually set to 1 for time-invariant channels and a value less than but close to 1 for time-variant channels, usually in the range $0.8 < \lambda < 1$. This value does not affect the convergence speed of the algorithm, but does control how well the algorithm tracks time-varying properties. The smaller the forgetting factor, the better the tracking. However, if λ is made too small, it can make the algorithm unstable. For this reason it is usually kept larger than 0.8.

While the convergence speed of the algorithm is much faster than the LMS algorithm, the complexity is much larger. Each iteration of the RLS algorithm requires $2.5N^2 + 4.5N$ operations, much more than the number required by LMS.

There are several other adaptive algorithms based on the Recursive Least Squares method that are available. The RLS algorithm discussed above, also known as the Kalman RLS algorithm, provides fast convergence speed at a cost of high complexity. Other simplified RLS algorithms have less computational complexity than the Kalman algorithm, but either provide slower convergence or require more complex programming structures. For more information on these algorithms see [4].

4.2 Use of Adaptive Algorithms with Different Equalizer Structures

Incorporating the adaptive algorithms with the different equalizer structures is a fairly easy and straight forward task for both linear equalizers and the decision-feedback equalizer. The MLSE is a special case because it does not filter the received signal like the others. However the MLSE equalizer requires knowledge of the channel characteristics. To determine these characteristics, an adaptive algorithm is often used. This algorithm and how it works with the equalizer will be discussed in detail later on.

4.2.1 Adaptive Algorithms for Linear Equalizers

Integrating either the LMS or RLS into a linear equalizer is very simple. Both algorithms were designed with this type of equalizer structure in mind. For the LMS algorithm, the following steps are used to equalize the data and adapt the filter taps.

First, the received data is filtered by the current set of coefficients.

$$z_n = \mathbf{c}_n^H \mathbf{u}_n \tag{4.19}$$

Next, the error is found between the equalized data value and the desired signal. The desired signal is known by the receiver if a training sequence is used. If the decision directed method is being used, the desired signal is generated from estimating the current symbol from z_n.

$$\varepsilon_n = I_n - z_n \tag{4.20}$$

Finally, the adaptive algorithm is used to compute a new vector of equalizer coefficients.

$$\mathbf{c}_{n+1} = \mathbf{c}_n + \mu \varepsilon_n \mathbf{u}_n^* \tag{4.21}$$

In these equations the vector, $\mathbf{u}_n$, represents the vector of received data points. For the symbol spaced equalizer this vector is made up of the sampled output of the matched filter, $r(t)$, where $r_n = r(nT)$.

$$\mathbf{u}_n = [r_{n+N1}, \ldots, r_{n+1}, r_n, r_{n-1}, \ldots, r_{n-N2+1}]^T \tag{4.22}$$

For the FSE the vector consists of samples of the received signal, $y(t)$, where $y_n = y(nT/M)$.

$$\mathbf{u}_n = [y_{n+N1}, \ldots, y_{n+1}, y_n, y_{n-1}, \ldots, y_{n-N2+1}]^T. \tag{4.23}$$

When the RLS algorithm is used with the linear equalizer, the procedure defined in Equations 4.14 through 4.18 is used. The $\mathbf{u}_n$ vector used by the LMS algorithm is the same for the RLS algorithm.

4.2.2 An Adaptive Decision-Feedback Equalizer

Though the decision-feedback equalizer is a nonlinear equalizer, both the LMS and RLS algorithms can be easily incorporated with the DFE structure. In this case the vector of coefficients will be made up of both the feedforward and feedback taps. The vector u_n will consist of both received data points and previously estimated symbols.

For the LMS algorithm the coefficient update step is given by

$$\begin{bmatrix} \mathbf{c}_{n+1} \\ \mathbf{b}_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_n \\ \mathbf{b}_n \end{bmatrix} + \mu\varepsilon_n\mathbf{u}_n^*. \tag{4.24}$$

For the RLS algorithm, the coefficient update step is given by

$$\begin{bmatrix} \mathbf{c}_{n+1} \\ \mathbf{b}_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_n \\ \mathbf{b}_n \end{bmatrix} + \mathbf{k}_n\varepsilon_n^*. \tag{4.25}$$

In both cases the vector, $\mathbf{u}_n$, is represented by

$$\mathbf{u}_n = [r_{n+N1}, \ldots, r_{n+1}, r_n, -I_{n-1}, \ldots, -I_{n-N3}]^T. \tag{4.26}$$

when operating with a training sequence. If the DFE is being operated in decision directed mode, the symbols, I_n, will be replaced by the symbol estimates, $\tilde{I}_n$, in the above equation.

The performance and complexity of both algorithms remains the same when used by the DFE as it does for the linear equalizers. The only difference is the representation of the adjustable filter taps and the organization of the vector of input data.

4.2.3 An Adaptive MLSE Equalizer

An MLSE equalizer differs from the other equalizers in that it does not filter the received signal to compensate for ISI. Instead it hypothesizes the different possible received symbol values and forms metrics based on the difference of these hypotheses to the actual received symbol. To form these hypotheses, the equalizer requires knowledge of the channel. In most cases this knowledge is not directly known. To determine these channel characteristics, a device known as a channel estimator is used by the equalizer. This device uses adaptive algorithms based on the minimum mean-square error criterion to compute the impulse response of the channel. A training sequence is often used by the system to adapt the estimator. Once the channel estimator has converged, its final estimate of the channel response is used by the MLSE equalizer to form the symbol hypotheses. The different algorithms used by the channel estimator are discussed below.

Channel Estimator for an MLSE Equalizer

The channel estimator used by the MLSE equalizer can use an adaptive algorithm based on either the LMS or RLS algorithm. However, unlike the algorithms used for the other equalizers, the channel estimator algorithms adaptively change the estimates of the channel impulse response, $\hat{\mathbf{q}}$, instead of the filter taps of the equalizer. Also, the cost function used by the channel estimator is generated differently than the one used by the other equalizers. In this case the cost function is the error between the sampled received signal value and its estimated value.

$$\varepsilon_n = r_n - \hat{r}_n \tag{4.27}$$

When a training sequence is used, the estimated received sample is created by multiplying the estimated channel response with the symbols of the sequence.

$$\hat{r}_n = \sum_{j=0}^{L-1} \hat{q}_j I_{n-j} \tag{4.28}$$

Here, $\hat{q}_j$ is the estimated channel impulse response and L is the number of elements it contains. The length of the response, L, affects the number of states in the MLSE equalizer

trellis. When the channel estimator is used in a decision-directed mode, the estimated received signal is formed using the output decisions, $\hat{I}_n$, of the equalizer. Because the output of the sequence estimator is delayed by several symbol periods, the estimate of the current symbol is not readily available when the equalizer operates in decision-directed mode. This mode and how it accounts for this delay will be discussed in more detail in the next subsection.

When the channel estimator based on the LMS algorithm is used, the estimator updates its channel estimates using the steepest decent method. In this case the correlation between the error and transmitted symbols is used as the adjustment factor.

$$\hat{\mathbf{q}}_{n+1} = \hat{\mathbf{q}}_n + \mu\varepsilon_n\mathbf{u}_n^* \tag{4.29}$$

In this equation, μ is the step size value and affects the adaption speed and algorithm stability much like the step size of the LMS algorithm. The vector of transmitted data symbols, $\mathbf{u}_n$, needed in this computation is given by

$$\mathbf{u}_n = [I_n, I_{n-1}, \ldots, I_{n-L+1}]^T. \tag{4.30}$$

The convergence properties of the channel estimator are very similar to that of the LMS algorithm. If the step size, μ, is chosen too large, the algorithm can become unstable. The smaller the step size, the slower the convergence, but the more accurate the channel estimate. As long as the step size is chosen appropriately, the final vector of channel estimates, $\hat{\mathbf{q}}_n$, should match the actual channel impulse response vector, $\mathbf{q}_n$. This is true if the length, L, of the channel estimate vector is equal to or greater than the length of the actual channel response. Setting L to be less than the length of the channel impulse response will cause the algorithm to produce the first L values in the response only if the remaining portion is small compared to the portion that is being accounted for. Otherwise, the estimate will not be accurate at all and the algorithm may even become unstable.

The channel estimator based on the RLS algorithm uses many of the same computations as those used by the other equalizers. Both the $\mathbf{k}_n$ vector and $\mathbf{P}_n$ matrix are computed as given by Equations 4.16 and 4.17. The only difference is that in this case the vector $\mathbf{u_n}$ is made up of the symbol values as shown in Equation 4.30. The error computation in this

case is the same as that of the LMS based estimator given above. The channel estimates are updated after each iteration using the following equation.

$$\mathbf{f}_n = \mathbf{f}_{n-1} + \mathbf{k}_n \varepsilon_n^* \tag{4.31}$$

This channel estimator has convergence properties and computational complexity much like that of equalizers using the RLS algorithm. Convergence will be much quicker than the LMS-based estimator. The forgetting factor, λ, will have little effect on the convergence speed. Again, the drawback to using this approach is the computational complexity. Because of all the extra calculations, this algorithm will require many more operations per second than the LMS algorithm.

Adaptive MLSE Equalizers for a Wireless System

An MLSE equalizer using an adaptive channel estimator to determine the channel characteristics can work in two modes. The first mode uses a training sequence known by both the receiver and transmitter to adapt the channel estimator. When the channel characteristics are time-invariant but unknown, the training sequence is sent before any data is transmitted. The channel estimator uses the sequence to converge to an estimate of the channel impulse response. Once the estimator has converged, the equalizer uses this estimated impulse response to determine estimates for the remainder of the transmitted data.

When the channel is time-variant and unknown, the equalizer structure changes. If the channel characteristics change very slowly compared to the symbol rate, the training sequence method can still be used. However, instead of being sent only at the beginning of the transmission, it needs to be sent in over and over again between frames of data. How often it needs to be sent depends on how quickly the channel is changing. An example of a direct-form MLSE equalizer operating in this mode is shown below in Figure 4.1.

If the channel response changes at a rate close to the symbol rate, the equalizer must operate in a different mode. This mode still requires the system to use training sequences to adapt the channel estimator, but requires the equalizer to operate in a decision-directed mode when the actual data is being tranmitted. This decision-directed mode is used because the

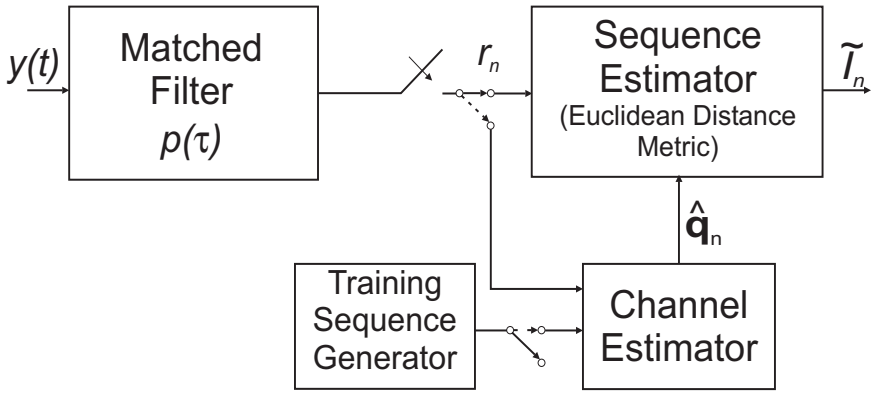

Figure 4.1: An Adaptive Direct-Form MLSE Equalizer Using a Training Sequence

channel characteristics will often change between the transmission of the training sequences. By using the decision-directed mode the channel estimator will always be adapting to the channel and the equalizer will be able to use a more current channel response instead of a past one that may no longer be accurate. Using the decision-directed mode will cut down on both the number of training sequences that are used and their length. This will allow for a higher data throughput.

The decision-directed mode, however, can prove difficult to implement. The main problem associated with this mode is that the equalizer does not reach decisions on the current symbol for several symbol periods. For this reason a received data sample must be buffered until the symbol estimate based on this data sample has been computed. The channel estimator then operates on this delayed sample. The problem is that channel estimates will be delayed by the latency of the sequence estimator which could be several symbol periods. If the channel is changing quickly, the channel estimate based on this delayed data may not be accurate for the current time.

A solution to the delay problem is to use a channel predictor along with the channel estimator. The predictor takes the delayed channel estimate along with past channel estimates to predict the current channel response. Channel prediction is a difficult and complex problem and will not be covered in depth within this book. For further information, please see [31].

An example of a direct-form MLSE equalizer operating in a decision-directed mode is shown

below in Figure 4.2.

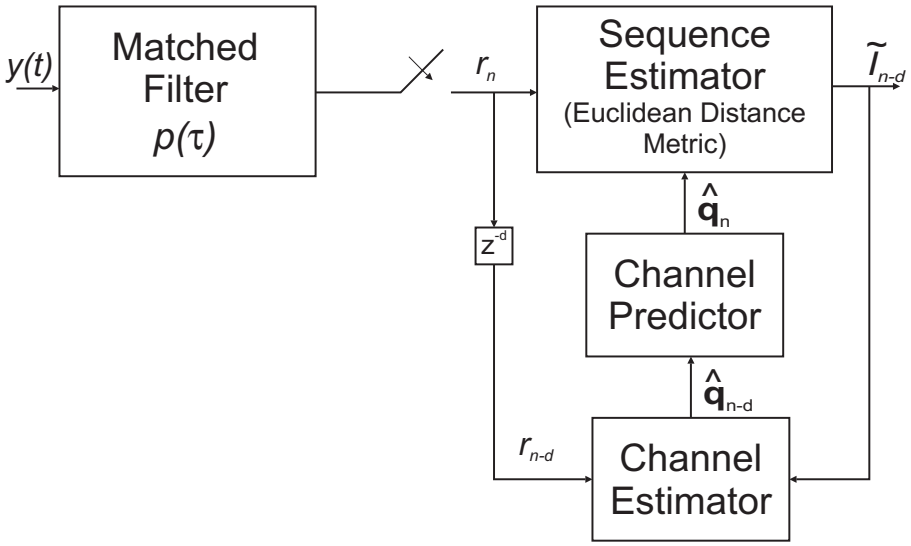

Figure 4.2: An Adaptive Direct-Form MLSE Equalizer Operating in Decision-Directed Mode

In summary, the complexity of an adaptive MLSE equalizer depends mainly on the severity of the channel. When a channel is time-invariant or varies much slower than the symbol rate, the equalizer can be kept relatively simple. Training sequences will be needed, but only occasionally. The equalizer will only require a channel estimator to operate. However, if the channel changes more rapidly, more training sequences will be needed, which reduces the throughput of the system. If the channel changes very rapidly, the equalizer must operate in a decision-directed mode. This mode requires the addition of a channel predictor which will be needed to predict the current channel response from past response estimates. The addition of this predictor will require more computations and will drastically increase the complexity of the equalizer.

4.3 Simulation of Adaptive Equalizers

As an aid to the reader, several MATLAB simulations of different equalizers employing adaptive algorithms have been included with this chapter. All of the four different equalizer

structures discussed in the previous chapter are simulated. For the linear equalizers and the decision-feedback equalizer, both the LMS and RLS algorithms are available to adapt the filter taps. The simulation of the MLSE equalizer includes a channel estimator that operates using the either the LMS or RLS based algorithm.

The processor programs use the same system model as the previous simulations. In these simulation an adaptive equalizer is used by the system. Along with the adaptive equalizer, the processor programs also include an equalizer that uses information about the channel to determine its taps. These are the same equalizers that were used in the previous chapter. By including both equalizer types, the reader is able to compare the output and performance of the adaptive equalizer with the optimal case.

Each simulation includes a preprocesor, processor, and postprocessor program. Each program functions the same as those included in Chapter 4. The preprocessor programs prompt the user for information about the simulation and system. They also request the user to set parameters for the adaptive algorithms. The postprocessor programs display many of the same output displays that were included with the previous simulations. They also display new graphs that are exclusive to these adaptive equalizer simulations. These include plots of the equalizer tap values versus time and plots of the mean-square error as the algorithm converges.

There are several different channel models available in these simulations. All of them are time-invariant and unknown by the adaptive equalizer. The optimal equalizer has knowledge of the channel response. The channels are kept time-invariant to keep the simulations simple and understandable. With time-invariant channel models, the convergence properties of the different algorithm becomes clear. A user is able to view the effects of the step-size parameter on stability, convergence, and excess mean-square error without worrying about variations in the channel. More complex, time-variant channel models are discussed in the next chapter. Simulations using these more complicated models are described then.

All the simulations use a sequence of data known by both the transmitter and equalizer to train the algorithms used by either the equalizer or channel estimator. This training sequence is sent to the receiver before any actual data is sent. This allows the equalizer

or channel estimator time to converge before the other data is transmitted. The length of the training sequence is selected by the user in the preprocessor program used by the simulation.

The next few sections describe each of the different simulations included with this chapter. These include simulations of adaptive LTE, FSE, DFE, and MLSE equalizers. The simulation structure of the programs and how they incorporate each adaptive algorithm are discussed in detail. Many of the plots available from the different simulations are included and explained.

4.3.1 Simulation of an Adaptive Linear Transversal Equalizer

The first simulation included with this chapter simulates an adaptive linear transversal equalizer. The programs **altepre.m**, **alteproc.m**, and **altepost.m** make up the preprocessor, processor, and postprocessor, respectively, of this simulation.

The preprocessor program prompts the user for the same data as the **ltepre.m** program did in the previous chapter. However, this program also asks for information about the adaptive algorithm such as the length of the training sequence and the type of algorithm to use. If the LMS algorithm is chosen, the size of the step size, μ, is asked for. If the RLS algorithm is selected, the value of the forgetting factor, λ, is requested. An example of this program is shown below.

```
>> altepre
Enter number of data symbols [1000]:
Enter Eb/N0 value [8]:
Enter number of samples (5-20) per symbol [10]:
Enter number of causal equalizer taps [4]:  10
Enter number of noncausal equalizer taps [2]:  4
Enter number of symbols in training sequence [500]:  100
Choose algorithm:  1-LMS 2-RLS [1]:  2
Enter forgetting factor for RLS algorithm [0.99]:
```

```
Enter the Number (1-10) of the Channel to Simulate [1]:  2
>>
```

Again, default values are given with each selection. These are just suggested values and are not absolute. Special attention should be paid to the step size of the LMS algorithm. The default value here is only a suggestion. For some equalizer setups, this value may cause the algorithm to not converge by the end of training sequence or, in some extreme cases, may cause the algorithm to become unstable.

The processor program is almost identical to the one used to simulate the optimal equalizer. The same code is used to model the system from the data source to the receiver's matched filter. Once the received data vector has been created it is passed to the equalizer section. The equalizer block then runs one of the two available adaptive algorithms.

If the user requested the LMS algorithm then it is used to adapt the filter taps. This begins by setting the equalizer to be an all-pass filter with an impulse response of a delta function. Then the algorithm begins. Each iteration requires three steps. The first is to filter the data. This is done by vector multiplying the filter coefficients with the vector representing the register holding the current received data. Next, the error is determined by subtracting the current filtered data value from the desired symbol value. Finally, the filter coefficients are updated using the following calculation based on Equation 4.7.

$$\mathbf{c}(:, \mathbf{i}+\mathbf{1}) = \mathbf{c}(:, \mathbf{i}) + \mathbf{mu} * \mathbf{r}(\mathbf{i}+\mathbf{N1} : -\mathbf{1} : \mathbf{i}-\mathbf{N2}+\mathbf{1})' * \mathbf{conj}(\mathbf{err}(\mathbf{i})); \tag{4.32}$$

The algorithm continues for the remainder of the training sequence.

If the RLS algorithm is requested, then the set of calculations used by this algorithm are used instead. Several of these calculations are similar to used by the LMS algorithm. Both the filtering and error calculations are identical. The RLS algorithm then calculates the $\mathbf{k}$ vector and $\mathbf{P}$ matrix in order to update the coefficients. These calculations along with the coefficient update are performed using the following commands.

$$\mathbf{k} = \mathbf{P} * \mathbf{u}/(\mathbf{lambda} + \mathbf{u}' * \mathbf{P} * \mathbf{u}); \tag{4.33}$$

$$\mathbf{P} = (\mathbf{P} - \mathbf{k} * \mathbf{u}' * \mathbf{P})/\mathbf{lambda}; \tag{4.34}$$

$$\mathbf{c}(:, \mathbf{i}+\mathbf{1}) = \mathbf{c}(:, \mathbf{i}) + \mathbf{k} * \mathbf{err}(\mathbf{i}); \tag{4.35}$$

Here, **lambda** is the forgetting factor set by the preprocessor. The filtering, error, and these calculations continue for the entire training sequence.

Once the training sequence has completed, the final tap values computed by either algorithm are used to filter the actual data. Symbol decisions are made on the output of the equalizer. These decision are used to compute a bit-error rate for the equalizer. This error rate is displayed on the workspace as **BERadapt**. The received data is also filtered by the optimal LTE coefficients. Symbol decisions are also made on the output from this equalizer. The BER computed from this data is called **BEReq** and is also displayed. Symbol decisions are also made on the unequalized data. The BER corresponding to these decision is displayed as **BERueq**. These BER computations can be used to compare the performance of the adaptive equalizer with both the optimal equalizer and the unequalized system.

Performance comparisons can also be made using the graphical plots of the postprocessor. The postprocessor produces many of the same plots that the optimal equalizer simulation did. For example, plots of the channel's impulse and frequency response are included. Plots of the equalizer taps can also be displayed. However, in this case, both the optimal equalizer taps and the adapted equalizer taps are displayed. This is also true for the scatter plot of the equalized data. Two plots are made, one for the output of the optimal equalizer and another for the adaptive equalizer's output. How well the adaptive equalizer performs can be measured by how closely its filter coefficients match the optimal coefficients or how well it filters the received data compared to the optimal equalizer.

Also included with the postprocessor are plots that describe the operation of the adaptive equalizer. One such plot shows the convergence of the filter taps. The value of each filter tap is plotted versus time for the duration of the training sequence. Similar to this is a plot of the mean-squared error versus time. The mean-square error is determined by performing a sliding average on the squared error data. When the algorithm is working correctly, the mean-square error should gradually decrease until it reaches a steady value. This indicates that the algorithm has converged. A user can compare the convergence properties of the LMS and RLS using this plot or can compare the convergence properties of the different

step sizes that are available with the LMS algorithm.

The final selection available with the LTE postprocessor is an animation of the adaptive equalizer's filter taps as they converge. This program, called **tapplot.m**, creates a bar graph of the equalizer taps at each point in time. An example output of this program is shown in Figure 4.3.

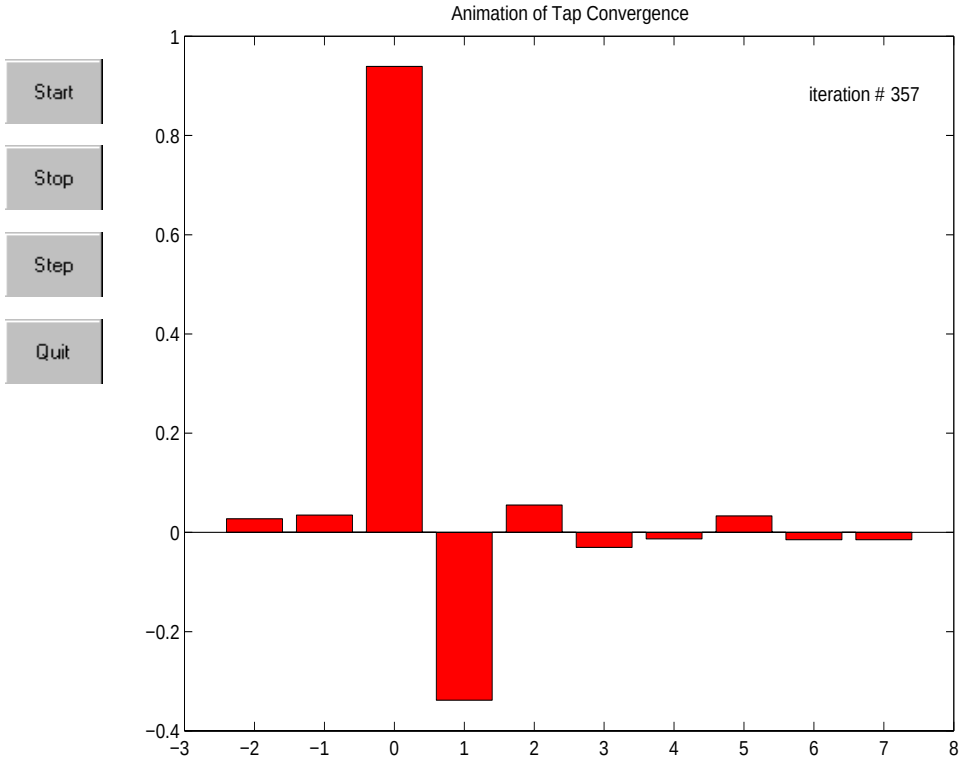

Figure 4.3: One Frame of LTE Filter Tap Animation

The plot is continuously updated with new tap values. This creates a type of animation where a user can observe the taps "grow" toward the converged values. There are four different control buttons included with the plot. The first, "Start", starts the animation. "Stop", stops the animation. To view one set of taps at a time, press the "step" button after the animation is stopped. To quit the program before all the tap vectors have been plotted, click on the "Quit" button. This returns the program to postprocessor menu. To reset the tap vector to its original state, quit the animation program and restart it from the postprocessor menu.

This simulation of the adaptive LTE can be used in many different ways. Users are encouraged to experiment with the different adjustable parameters. One example is to examine how long it takes the LMS algorithm to converge using different step-sizes. The sequence should be kept as short as possible. However, the equalizer needs sufficient time to converge to the minimum mean-square error. This tradeoff of training sequence length to minimum mean-square error is one that often arises when designing a communication system that employs an adaptive equalizer.

4.3.2 Simulation of an Adaptive Fractionally Spaced Equalizer

The second simulation included with this chapter is one of an adaptive fractionally spaced equalizer. Much like the adaptive LTE simulation, this simulation is very similar to the FSE simulation included with the previous chapter. It models the system exactly the same as the optimal FSE simulation from the data source to the input of the equalizer. At this point an adaptive equalizer is used to filter the received data.

The preprocessor program for this simulation, called **afsepre.m**, requests much of the same data as the preprocessor for the optimal FSE. The user is prompted for parameters such as the oversampling rate of the equalizer, the amount of symbols to be transmitted, and the number of samples per symbol. Along with these parameters, information about the adaptive portion of the simulation is also requested. This includes the length of the training sequence, the type of algorithm to be used, and the step size or forgetting factor for this algorithm.

The processor program, called **afseproc.m** works much like the simulation of the adaptive LTE system. A training sequence is added to the beginning of the actual data stream so that the equalizer is able to converge to a set of tap values before it filters the received data. The processor is also able to determine the filter tap values by running either the LMS or RLS algorithms. Along with the adaptive equalizer, the processor also includes an equalizer that calculates the optimal tap values based on the overall channel response. How well the adaptive equalizer performs can be measured by comparing its output with the output of the optimal equalizer. Once the processor finishes executing it computes and display the

bit-error rates for the system using the adaptive equalizer, the system using the optimal equalizer, and the system using no equalizer.

The postprocessor for this program, **afsepost** displays many of the same plots available with the other FSE simulation. These plots include graphs of the channel impulse and frequency responses, plots of the equalizer taps for both the optimal and adaptive cases, and scatter plots of the equalized data. Other plots that describing the performance of the adaptive algorithm used by the equalizer are also included. These include plots of the tap vector versus time, a plot of the mean-square error, and an animation of the tap values as they converge.

One plot, exclusive to the fractionally spaced equalizer simulation, is one that shows the combined response of the channel and equalizer. This provides a good idea of how well the equalizer is able to remove the ISI from the received signal. For this simulation, two plots are generated that describes this response. The first displays the combined response of the channel and the final tap values generated by the adaptive equalizer. The other plot shows the combined response of the channel and optimal equalizer taps. The comparison of these two plots provides insight into how well the adaptive equalizer is able to match the optimal equalizer taps.

An example of these plots is shown in Figures 4.4 and 4.5. In this example, the first plot shows the combined response of the channel with the optimal equalizer taps. The second displays the combined response of the channel with the taps of adaptive equalizer. The adaptive equalizer in this case, used the LMS algorithm with a training sequence of 100 symbols and a step size of 0.05. The E_b/N_0 was set to 8 dB.

As you can see, the combined response using the adaptive equalizer is very similar to the combined response using the optimal coefficients. There are some small variations but they are miniscule. Once the responses are sampled at the correct time, the overall response is very close to a delta function, the desired response. This indicates that the adaptive equalizer is able to come very close to matching the optimal equalizer. Further indication of the adaptive equalizer's performance can be seen by comparing the scatter plots of the output of the optimal and adaptive equalizers.

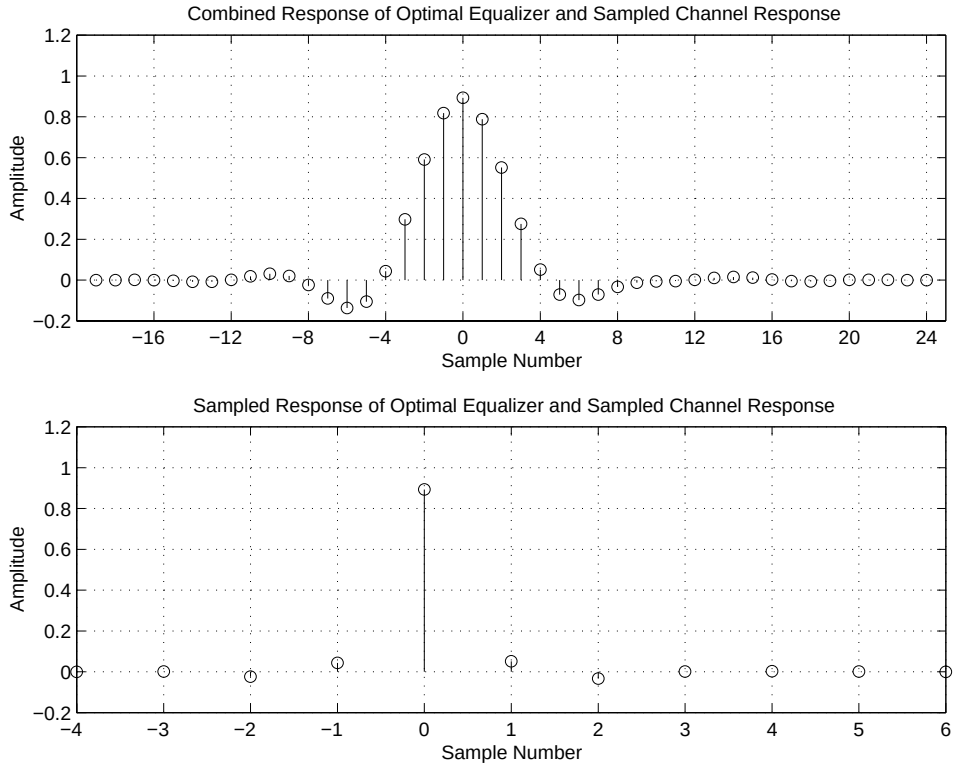

Figure 4.4: Combined Response of Channel and Optimal Equalizer

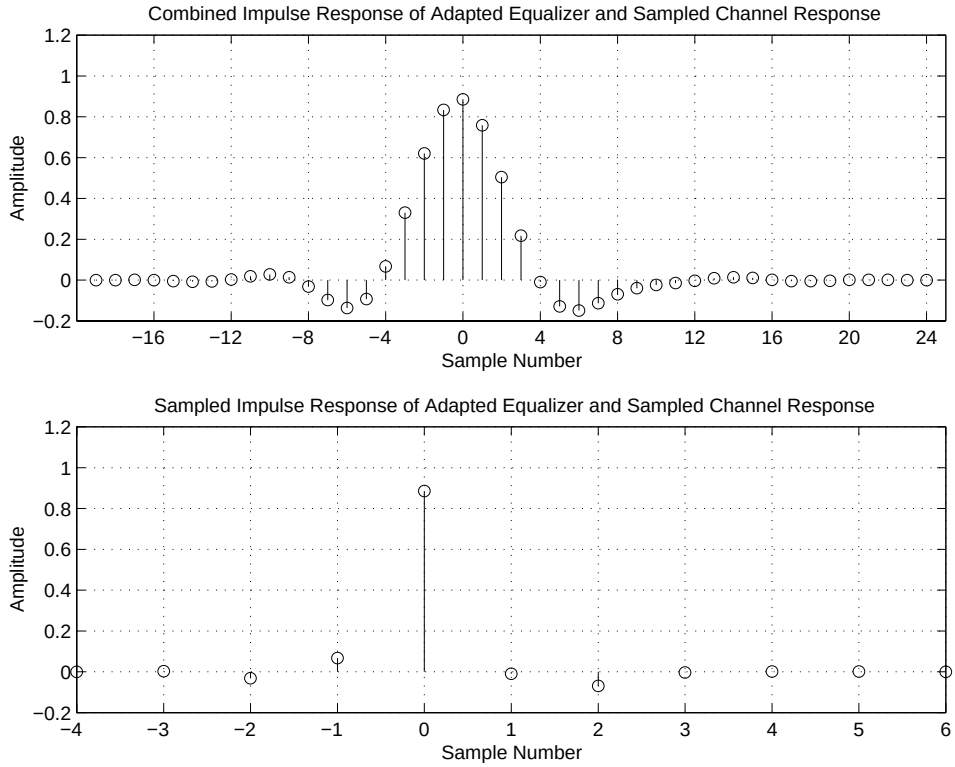

Figure 4.5: Combined Response of Channel and Adaptive Equalizer

4.3.3 Simulation of an Adaptive Decision-Feedback Equalizer

The third simulation included with this chapter models a wireless BPSK system employing an adaptive decision-feedback equalizer to compensate for ISI. This simulation is very similar to the simulation that models an optimal DFE given in the last chapter. The preprocessor for this program, **adfepre.m**, requests much of the same information as the preprocessor for the optimal DFE program. This includes information such as the number of taps in the feedback and feedforward filters, the number of samples to transmit, and the number of samples per symbol. This is joined by requests for information about the adaptive equalizer such as the type of algorithm to use and the length of the training sequence.

The processor program, **adfeproc.m**, used in this simulation is closely related to the processor program used by the simulation of the optimal DFE equalizer **dfeproc.m**. In this case an adaptive equalizer is included along with an equalizer that uses the optimal taps. The processor program generates a training sequence along with a random data sequence that is used to train the adaptive algorithm. Both the LMS or RLS algorithms are included with this program. These algorithms are used to compute filter taps based on the received data and the original transmitted data. The taps found from the adaptive algorithm are then used to filter the received data. The BER found from using this adaptive algorithm is calculated and displayed on the workspace. An equalizer using the optimal taps also filters the same received data. The BER found from decisions on its output is also computed and displayed. Finally, the BER that results from decision based on the received data itself, without using any equalization, is found and displayed.

The postprocessor program included with this simulation, called **adfepost**, includes many of the standard output plots associated with the equalizer simulations. This includes plots of the channel response, plots of both the optimal and adapted filter taps, and scatter plots of the equalizer and unequalized data. Also included is an animation of the tap convergence. This is the same type of program that was used to animate the coefficient convergence of the other two adaptive equalizers, the only difference is that the feedback taps in this case are represented by blue bars while the taps of the feedforward filter are colored red.

Two other plots that have been mentioned with the other two adaptive equalizer simulations, but not discussed in detail are plot of the mean-square error during the training sequence and the plot of the tap weights during the training sequence. These two plots provide much information about the convergence of the adaptive algorithm. An example of these plots is shown in Figures 4.6 and 4.7. The first graph is a plot of the mean-square error and the bottom graph is a plot of the tap weights. The mean-square error was calculated by taking the sliding average of 30 values of the square-error.

The system modeled in this simulation used a training sequence of 500 symbols and an E_b/N_0 of 20 dB. The equalizer has two feedforward taps and four feedback taps. The LMS algorithm with a step size of 0.03 was used to compute the tap weights.

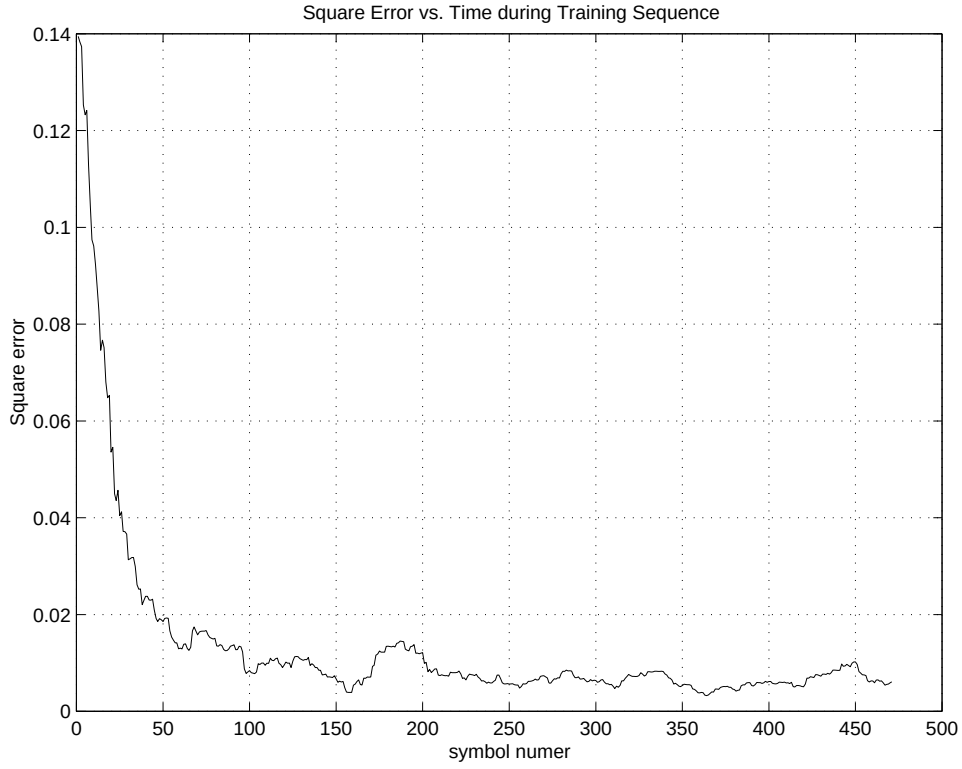

Figure 4.6: Mean-Square Error of the Adaptive Decision-Feedback Equalizer ($E_b/N_0 = 20$ dB, $\mu = 0.03$, Channel #1)

These graphs indicate that the algorithm reaches a steady state after approximately 200 symbols. This means that the full 500 symbol training sequence was not needed. It is also

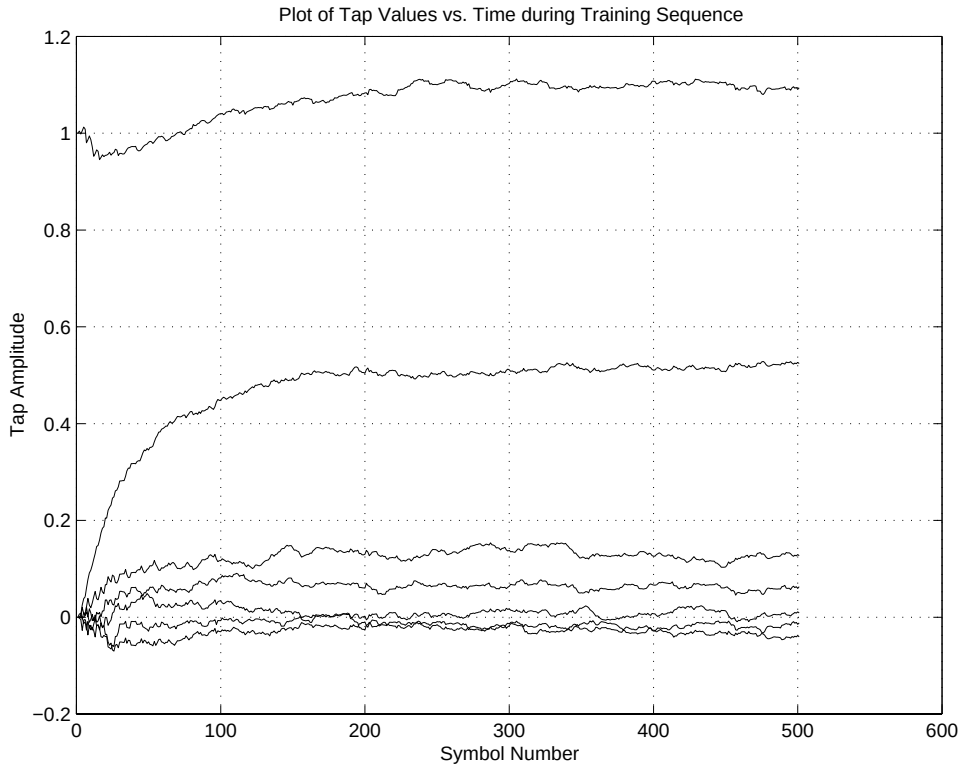

Figure 4.7: Convergence of the Tap Weights of the Adaptive Decision-Feedback Equalizer ($E_b/N_0 = 20$ dB, $\mu = 0.03$, Channel #1)

interesting to see that the tap values vary very little from their final values. This is because the step-size is rather small. Also of interest is the value of the mean-square error after the algorithm has converged. It is very close to the error that would be caused by the AWGN. This indicates that almost all of the ISI has been removed from the received signal and that the excess MSE is very small. Again this is because the step size is relatively small.

Users are encouraged to experiment with the simulation to observe the effects of the step size on the mean-square error and the convergence of the taps. Large step-sizes should result in fast convergence but large excess mean-square error. Smaller step-sizes, like the one used in this example, should result in slower convergence but low excess MSE.

4.3.4 Simulation of an Adaptive MLSE Equalizer

The final simulation included with this chapter models an MLSE equalizer that uses an adaptive channel estimator. A time-invariant channel is used just like in the previous simulations. For this reason, a training sequence is sent before any data to train the channel estimates. Once the estimator has converged, the estimated channel response is used by the equalizer to make decisions on the remaining data. Because the decision-directed mode of operation is not required, no channel prediction is needed. This keeps the simulation simple and straightforward.

The preprocessor for the simulation is called **amlsepre.m**. It requests system parameters such as the number of symbols, the size of the data frames, and the number of ISI terms to account for. It also requests information about the channel estimator such as the algorithm to use and the step size or forgetting factor associated with that algorithm. This information is then saved to be used by the processor program, **amlseproc.m**

The processor program for this simulation is based closely on the processor of the MLSE simulation given in the previous chapter. However, some changes have been made to include the adaptive channel estimator. For example, a training sequence is transmitted before any data is sent. This training sequence is then passed to the channel estimator that uses either an LMS or RLS based algorithm to estimate the channel response. This adaptive channel

estimator is based on the theory described in the channel estimation section of this chapter.

The output of the channel estimator is given by the vector **Fest**. This is the final channel estimate produced. Once this vector is found it is then passed to the equalizer block. In this block, the received data is broken down into frames of length set by the user. The channel estimate and each frame is then passed to the **mlse.m** program that uses the Viterbi algorithm to determine the most likely sequence of data that was transmitted.

The processor also runs the **mlse.m** program with the actual channel response in order to measure the optimal performance of the equalizer. This program also breaks the received data into frames before it passes it the equalizer. A BER is computed from the output of this optimal equalizer as well as for the output of the equalizer using estimates of the channel response. A BER is also found for the case where no equalization is used.

Besides the BER measure, there are several other performance measures that can be found. The postprocessor for this simulation, **amlsepost.m** includes many different graphical outputs that can be used to measure the performance of the adaptive equalizer. These include a plot of the mean-square error during the training sequence, a plot of the channel estimates during the training sequence, and plots of the trellises found from using either the estimated or actual channel response.

Probably one of the most informative plots included with the postprocessor shows the actual truncated channel response and the output of the channel estimator. How well the channel estimate matches up with the actual response gives a good indication of how well the equalizer using this estimate performs. Two examples of this plot are shown below. The first shows a case where the estimates match up very closely to the actual channel response. For this case the BER for the equalizer using the actual response is 0.021 while the BER for the equalizer using the channel estimate is 0.022. A very small difference. The second plot indicates that the channel estimates did not converge to the correct channel response. This could be because the step size of the algorithm was too small, or that the training sequence was not long enough to have the estimate converge. In this case the BER of the equalizer using the actual channel response was 0.016 and the BER of the adaptive equalizer was 0.041, almost three times as much. This shows how important good channel estimates are

when using the MLSE equalizer. The better the estimates, the better the performance of the equalizer.

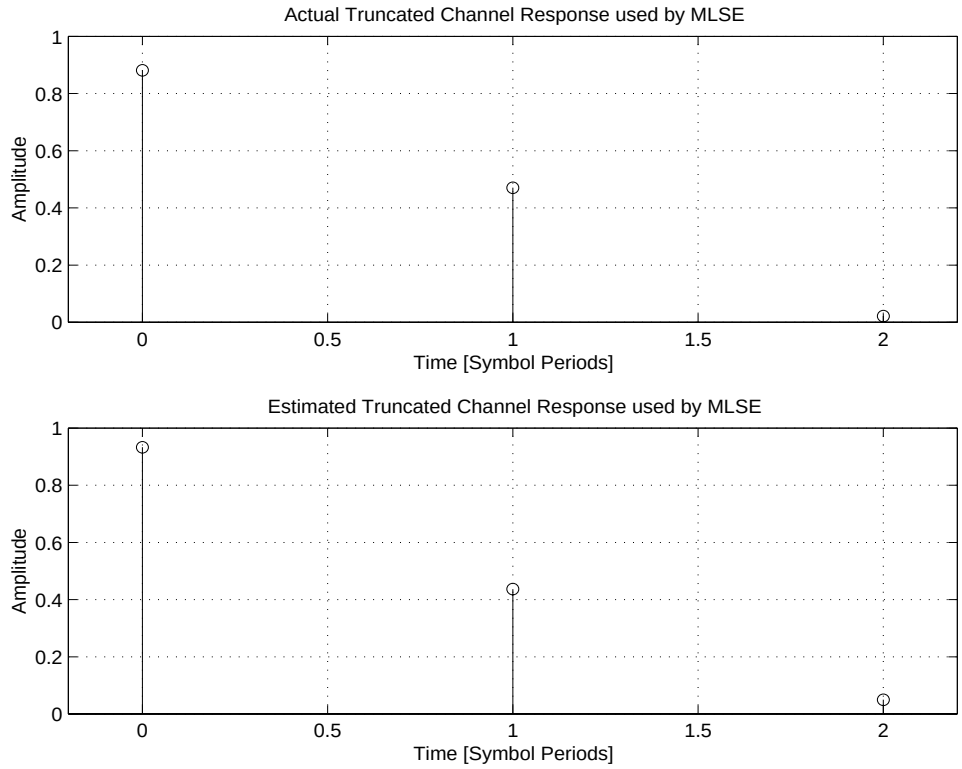

Figure 4.8: Actual Channel Response and Estimated Channel Response

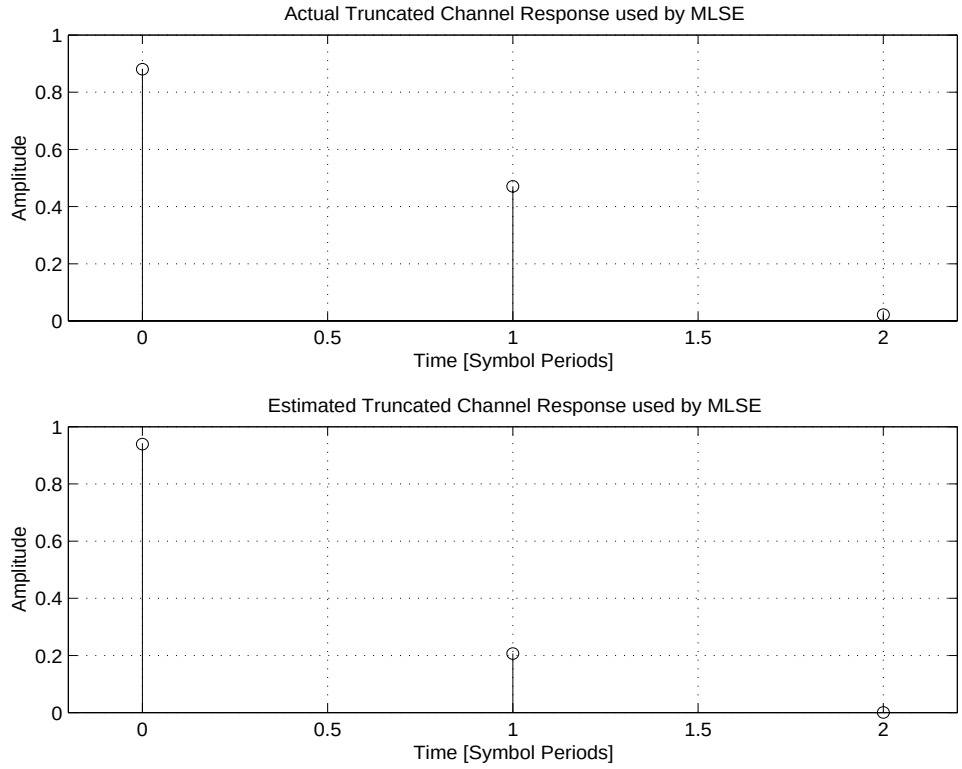

Figure 4.9: Actual Channel Response and Estimated Channel Response

Chapter 5

Equalization for Mobile Radio Systems

All the channel models used thus far have been time-invariant models either known or unknown by the receiver. These models are useful when simulating an indoor communication system because most of these applications have receivers and transmitters that are stationary or that move at very slow speeds. In these types of environments a large frame of data can be transmitted with almost no change of the channel response. The channel can therefore be modeled as time-invariant during the duration of the frame.

These time-invariant models work well for testing indoor applications, but do not provide a good representation of an outdoor environment. This environment is much different than the indoor. The delay spreads are much larger because the arrival times of the reflected waves are much greater than those of an indoor channel. Delay spreads of indoor channels can vary between 10 - 300 ns at a carrier frequency of 900 MHz depending on the type of building or room. Delay spreads can be anywhere between 20 to 100 times longer for the outdoor, depending on the exact environment, operating at the same carrier frequency. These larger delay spreads force the data rates of most outdoor systems to be slower than those used by indoor systems.

Another factor to consider when modeling outdoor channels is that many wireless applica-

tions operating in an outdoor environment are mobile. Wireless systems used in vehicles can have the transmitter or receiver moving at speeds up to 120 km/hr. This movement causes the channel seen by the system to change fairly quickly compared to a system's frame rate. In this situation, the time-invariant models used previously are not accurate representations of the channel. For this reason, time-variant models need to be developed to measure the performance of different equalizers operating in these types of environments.

This chapter examines how motion of the receiver or transmitter causes the channel to vary with time. It will also examine some of the statistical models used to represent outdoor and mobile channels. The chapter concludes with a detailed discussion of the simulations of both decision-feedback and an MLSE equalizers operating in a mobile system that are included with this chapter.

5.1 Small-Scale Fading

One of the largest effects of transmitting a signal using a mobile communication system is small-scale fading [32]. Small-scale fading describes the rapid change in received signal amplitude over a small time or distance. It is caused by two or more versions of the same signal adding constructively and destructively at different points in space. As the mobile moves through this multipath field, the signal envelope will drop out or fade from time to time. The faster the mobile travels, the more signal fades the receiver will see.

The movement inherent to the mobile communication system causes a Doppler shift in the received signal. How quickly the signal envelope fades is directly dependent on this shift. The maximum Doppler shift is the Doppler shift associated with the mobile moving directly away or toward an antenna. It is found from the velocity of the mobile, v, and the wavelength of the carrier frequency, λ.

$$f_m = v/\lambda \tag{5.1}$$

Closely related to the Doppler shift is the coherence time of the channel. The coherence time gives a time duration during which the signal envelope might experience a sharp fade.

This is often related to the Doppler shift using the following equation.

$$T_c = \sqrt{\frac{9}{16\pi f_m^2}} = \frac{0.423}{f_m} \tag{5.2}$$

5.1.1 Rayleigh Fading

One of the most common statistical distributions to describe the varying signal envelope is the Rayleigh distribution [3]. Short-term fading is sometimes referred to as Rayleigh fading. This distribution can describe the fading of the signal envelope or the fading of individual multipath elements. The pdf of the Rayleigh distribution is given by

$$p(r) = \begin{cases} \frac{r}{\sigma^2} \exp\left(\frac{-r^2}{2\sigma^2}\right) & 0 \leq r < \infty \\ 0 & r \leq 0 \end{cases} \tag{5.3}$$

where σ is the rms value of the received signal and r^2 is the instantaneous power.

5.1.2 Models for Rayleigh Fading

Several researchers have come up with ways to model the Rayleigh fading that occurs over a mobile channel. Clarke [33] came up with a model to describe why the fading envelope follows a Rayleigh distribution. Gans [34] improved on Clarke's to provide a frequency analysis of the envelope. His model explains the time-correlation included with the fading envelope.

Clarke's Model

The model that Clarke developed assumed that there was no line-of-site (LOS) signal at the receive antenna. This means that the electromagnetic field of the received signal is made up of components that were scattered off of buildings and other objects in the environment. Clarke determined that the E-field of the received signal could be expressed as

$$E_z = A_c(t)\cos(2\pi f_c t) + A_s(t)\sin(2\pi f_c t) \tag{5.4}$$

The amplitude of the in-phase and quadrature components take the form

$$A_c(t) = E_0 \sum_{n=1}^{N} C_n \cos(2\pi f_n t + \theta_n) \tag{5.5}$$

and

$$A_s(t) = E_0 \sum_{n=1}^{N} C_n \sin(2\pi f_n t + \theta_n). \tag{5.6}$$

Here, f_n represents the Doppler shift, given by $f_n = v/\lambda \cos \alpha_n$ where α_n is the angle of arrival, of the n-th reflected wave. The variable C_n is the amplitude of this wave and θ_n is the phase of this wave. Both these variables are random processes. When N is large enough, the central-limit theorem holds and the two amplitudes, A_c and A_s, can be modeled as Gaussian random processes. The envelope of the received signal is computed from these two amplitudes.

$$r(t) = |E_z| = \sqrt{A_c^2(t) + A_s^2(t)} \tag{5.7}$$

Since the amplitudes of the two components are Gaussian, the envelope will have a Rayleigh distribution.

Gans' Model

The statistical properties of the fading envelope do exhibit a Rayleigh distribution over many different ensembles, however, it is important to note that the fading envelope is correlated in time. Gans improved on Clarke's model and determined these correlations based on the Doppler spread of the signal.

The frequency of the signal component arriving from angle α is given by

$$f(\alpha) = \frac{v}{\lambda} \cos \alpha + f_c = f_m \cos(\alpha) + f_c. \tag{5.8}$$

Using this information, Gans determined that the power spectrum of the received spectrum will take the form

$$S_{E_z}(f) = \frac{P_0(p(\alpha)G(\alpha) + p(-\alpha)G(-\alpha))}{f_m \sqrt{1 - \left(\frac{f - f_c}{f_m}\right)^2}} \tag{5.9}$$

for $|f - f_c| \leq f_m$. Here, P_0 is the average received power, $p(\alpha)$ is the probability of a signal arriving at angle α, and $G(\alpha)$ is the power gain of the antenna at that angle.

Assuming the a quarter-wave dipole is used with $G = 1.5$ for all angles and that $p(\alpha)$ is uniform over all angles then the power spectrum can be simplified to

$$S_{E_z} = \frac{1.5}{\pi f_m \sqrt{1 - \left(\frac{f - f_c}{f_m}\right)^2}} \tag{5.10}$$

This describes the spectrum of the fading envelope. It can be used to determine the time-correlation of the fading.

Clarke's and Gans' models are often used to simulate a Rayleigh Fading Channel. How this is done is described below.

5.1.3 Simulation of Rayleigh Fading Channel

The simulation of a Rayleigh fading channel is often used to measure the performance of mobile wireless systems. There are few different ways to simulate the fading channel. One of the most common is a method developed by Smith [35]. A block diagram of his method is shown below.

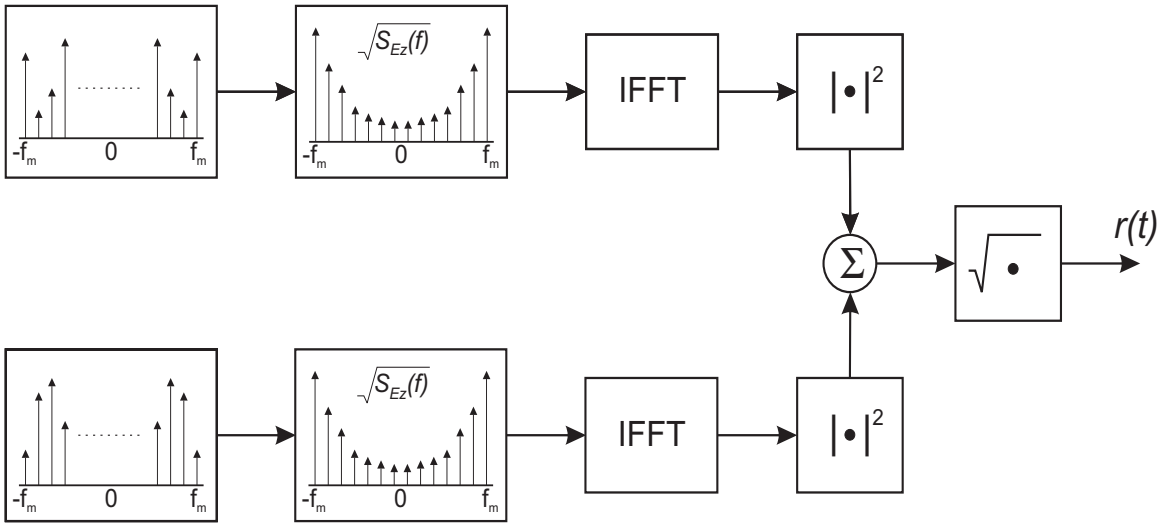

Figure 5.1: Block Diagram of Rayleigh Fading Envelope Simulator

Before the simulation begins, two parameters need to be found: the maximum Doppler frequency, f_m, and the number of discrete frequency components, N, to use. The value of

N is usually set to a power of two for convenience. Once these parameter values are found, the simulation can begin.

The first step is to determine the spacing, Δf, of the frequency components. This is found from

$$\Delta f = \frac{2f_m}{N-1}. \tag{5.11}$$

The next step is to generate random amplitudes for each of the discrete frequency components. The amplitude of each component will be a complex Gaussian random variables. Two sets of $N/2$ complex random numbers should be generated for the positive frequencies, one set for the in-phase component and one set for the quadrature component. The amplitudes of the negative frequency components are then set to the conjugate of the amplitude of the corresponding positive frequency component.

Once the frequency components amplitudes are determine, the next step is to find the spectrum of the waveform, $S_{E_z}(f)$. This is done by solving Equation 5.10 at each of the discrete frequency values. Because the frequency components are discrete, the values of the spectrum at $f = \pm f_m$ must be found by extrapolating their values from the points closest to them. After this spectrum is found, the random frequency component amplitudes are then multiplied with the square-root of the spectrum, $\sqrt{S_{E_z}(f)}$. Both sets of weighted frequency components are then transformed to the time domain using the IFFT. The square of the magnitude of each IFFTs' output is then computed. These two vectors of data are then added together. The output of the simulator is the square root of this sum.

A MATLAB simulation that uses this method is included with this chapter. It is called **fademaker.m**. It follows this method almost exactly. The only difference is that it computes the phase change of the signal along with its envelope. To find the value of $S_{E_z}(f)$ at $f = \pm f_m$, the function **polyfit** is used to extrapolate this particular value from the points lead up to it.

An example of the output of this program is shown below. The maximum Doppler spread, f_m, was 100 Hz which is the spread of a mobile moving at 120 km/hr receiving a signal with a 900 MHz carrier.

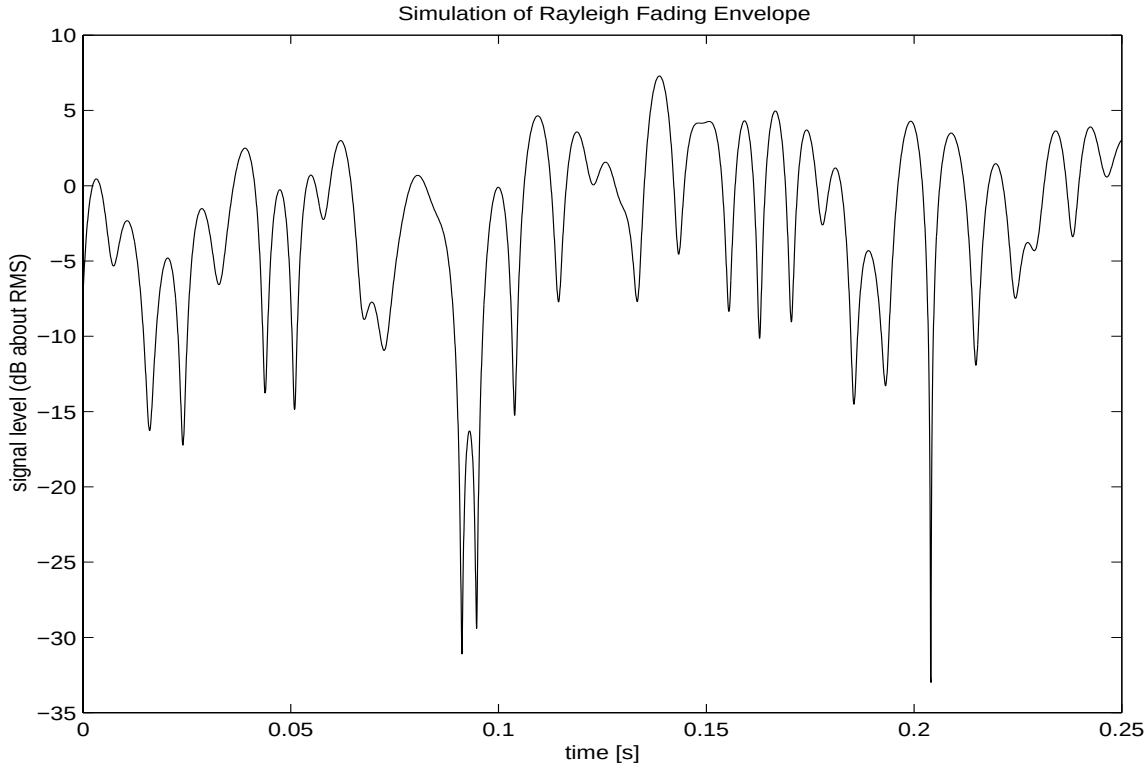

Figure 5.2: Example Output of the Rayleigh Fading Envelope Simulator

5.2 Time-variant Multipath Channel Models

There are several different types of channel models that can be used to represent a mobile radio channel that exhibits short-term fading. Some are simple mathematical models while others are very complex statistical models based on thousands of measurements of signals traveling in real environments.

5.2.1 Jakes Two-Ray Model

The two-ray model developed by Jakes [36] is one of the simplest of all the time varying frequency-selective channel models available. It is also one of the most widely used models, especially for testing the performance of different equalizer structures and algorithms. The form of the two-ray model is just what its name implies, one main signal and one delayed signal added together. A block diagram of the model is shown below.

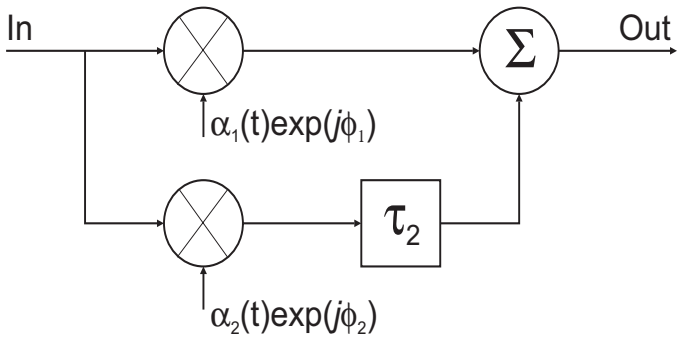

Figure 5.3: Block Diagram of a Two-Ray Channel Model

The impulse response of this channel is given by

$$g(t,\tau) = \alpha_1(t)e^{j\phi_1}\delta(\tau) + \alpha_2(t)e^{j\phi_2}\delta(\tau - \tau_2). \tag{5.12}$$

The $\alpha_n(t)$ terms refer to the amplitude of each ray. These amplitudes are independent and Rayleigh distributed. The ϕ_n terms describe the phase of each ray and are uniformly distributed between 0 and 2π. The delay of the second ray is given by τ_2. The delay spread of the channel depends on the position of this delay. The greater the delay, the larger the delay spread. The time correlation between α_1 and α_2 can be ensured by generating

two independent Rayleigh fading waveforms. These waveforms can be generated using the method proposed in Section 5.1.3.

This method is useful for testing equalizers, but does provides only a simplified model of an actual mobile radio channel. It tends to produce optimistic BERs. There are other channel models that provide a better representation of the mobile outdoor channel that are based on this model. For example, when representing mobile radio channels with a high normalized delay spread, it is often useful to use models that uses more than two rays. These additional rays will exhibit Rayleigh fading that is independent of the other rays. The delays of each ray will often be spaced evenly with the other rays, usually at the symbol duration. Another example is a two-ray model where the averaged received power of the second ray is less than that of the main ray. This attenuated ray model can also be expanded to the multi-ray case.

5.2.2 Time-varying Channel Models based on Measured Data

Along with Jake's two-ray channel model there are many other more complex channels models that are derived from measurements of real-world channels. Researchers will take thousands of measurements of actual channel responses and then determine statistical models that best fit the measured data. These statistics are used to develop a model that generates theoretical channels that are very similar to the real channels that were measured.

As stated in Chapter 2, the multipath channel can be modeled as a summation of weighted, delayed delta functions. When the channel is time-variant, each path amplitude and phase will become time dependent. This is shown in the equation below.

$$g(t,\tau) = \sum_k \alpha_k(t) e^{-j\theta_k(t)} \delta(t, \tau - \tau_k) \tag{5.13}$$

The time variation of the individual paths is caused be either movement of the receiver, transmitter, or objects in the environment. Most complex channel models ignore the movement of objects in the environment and are only concerned about the motion of the mobile. The models determine channel profiles at points in space, usually separated by a quarter-wavelength. To determine how the channel varies with time, the channel models update the

channel profile depending on the velocity of the mobile. The receiver will then see several channel profiles as it moves along. Each channel profile will be correlated with the next one so the channel model will change gradually with time and will not exhibit sharp changes between individual profiles.

The exact values of the amplitudes, phases, and delays of each channel profiles depend on many different factors. An example would be the type of environment the system is operating in. Urban channels tend to exhibit much longer delay spreads than suburban or rural channels. Most urban channels are also obstructed channels where there is no direct line-of-site component to the received signal. Most suburban and rural channels do provide a line-of site component to the receiver. Other factors that determine the values of the channel elements include the distance between the transmit and receive antenna and the height of each antenna.

Several researchers have created models based on measurements from different locations. In the late 1970s, Hashemi [37] came up with an outdoor channel model that is widely used today. It is based on thousands of measurements made by Turin *et al.* [38] of wireless multipath channels in and around the San Francisco area. Hashemi sound statistical distribution for the amplitude, phase, and time delay of each path in the channel profile. He also computed the correlation between the delays and the amplitudes of each path depending on the type of environment and other factors affecting the channel.

Using the work of Hashemi and others, researchers have developed many commercial software packages that can be used to create channel profiles and time-variant channel models. These programs compute these profiles based on several parameters that are selected by the user such as the type of environment, and the speed of the mobile. The programs are very useful in measuring the performance of a wireless communication system. Because they provide a better representation of the channel, simulations using these models can provide more accurate performance measures.

5.3 Simulation of Equalizers for Mobile Radio Systems

Included with this chapter are simulations of equalizers used in mobile radio systems. Many of these simulations are based on information found in [39] and [40]. These simulations model two types of decision-feedback equalizers and an MLSE equalizers operating in a mobile radio. These two equalizer structures are the most common types found in mobile radios. Both structures are able to provide good performance even when operating in nonminimum phase channels where the strongest energy arrives after the first arriving signal component, channels often found in outdoor mobile environments.

The channel model used in these simulation will be the two-ray frequency selective model developed by Jakes which was discussed previously. This model is often used to measure the performance of different equalization structures and algorithms. It has been shown that this model provides optimistic BER results, but it is still useful when comparing different equalization schemes.

There are three different simulations included with this chapter. The first models a decision-feedback equalizer used in a mobile BPSK system. This is a generic model that is used to study the affect of certain parameters on the performance of the system. The second simulation models an MLSE equalizer. This equalizer is also used in a mobile BPSK system. Aside from the type of equalization used, this simulation is very similar to that of the mobile DFE. It is a generic model that can be used to study the affect of changing certain system parameters. Since the MLSE equalizer and the DFE equalizer simulation are very similar, it is easy to compare the performance achieved by the two different structures.

The final simulation included with this chapter models a decision-feedback equalizer, but differs from the other DFE simulation. In this case the simulation models the use of a DFE in a system similar to that described by the United States Digital Cellular System (USDC) mobile radio standard. Many aspects of this simulation will differ from those of the generic DFE simulation. For one QPSK modulation is used in place of BPSK. Though the USDC standard actually uses a form of differential QPSK known as $\pi/4$-DQPSK, the use of QPSK provides a better approximation to this modulation scheme than BPSK can. Another

large difference is that the DFE in this simulation operates in a direct-decision mode. This means that the tap weights will be updated constantly, not just during the training sequence. Another change will be that the default values included with the preprocessor of the simulation will be set to the values actually used by USDC systems. Though it would be useful to have a fourth simulation modeling an MLSE equalizer used in a USDC type system, it was found that such a simulation would be too complex to be included with this book. Such a simulation would require the Viterbi algorithm used by the equalizer to compute 4^{L+1} states instead of the 2^{L+1} required by such an equalizer operating in a BPSK system. Also, operation in a decision-directed mode requires the equalizer to continously output symbol decisions after each symbol is received. This type of setup is much more difficult to implement compared to the frame-by-frame method used before. In order to continuously update the channel estimates, some sort of prediction algorithm would be needed to compensate for the delay incurred by the equalizer when determining symbol decisions. For these reasons, it was decided that such a simulation would be complicated and too advanced for this book.

5.3.1 Simulation of Decision-Feedback Equalizer Operating in a Mobile Radio System

The first simulation included with this chapter models a mobile BPSK system that uses a decision-feedback equalizer to compensate for the ISI introduced by the frequency-selective channel. This channel is created using Jake's two-ray channel model. The equalizer in this system is modeled much like the one used in the adaptive decision-feedback simulation included in the previous chapter. The main difference is that the equalizer is trained periodically to adapt to the continuously changing channel.

The data in this simulation is broken down into frames of a specified length. The frame is made up of two parts: the training sequence and the data symbols. The training sequence is used to adapt the equalizer to the channel. Once the entire training sequence has been received, the final tap weights found from the equalizer's adaption algorithm are saved. These tap weights are then used to filter the received data until the end of the frame is

reached. Once the frame is completed, the process begins again with a new frame of data.

This simulation is made up of a preprocessor, processor, and postprocessor programs like all the other simulations included with this book. These programs are described below.

Preprocessor

The preprocessor for this simulation is called **mbldfepre.m**. It is similar to the preprocessor used for the adaptive DFE simulation but requires additional information that describes the channel and other attributes. An example of the preprocessor is shown below.

```
>> mbldfepre
Enter symbol rate [20000]:
Enter Doppler frequency [50]:
Enter number of data symbols per frame [100]:  200
Enter number of frames [50]:
Choose algorithm:  1-LMS 2-RLS [2]:
Enter forgetting factor for RLS algorithm [0.9]:
Enter length of training sequence [20]:
Enter Eb/N0 value [30]:  10
Enter delay of second ray in samples [12]:
Enter number of feedforward equalizer taps [2]:  1
Enter number of feedback equalizer taps [2]:
>>
```

This preprocessor requires a great deal of information, all of which is very important to the simulation. Some of the parameters that have not been seen in previous simulation include information about the channel such as the Doppler frequency, f_m, and the delay of the second ray. Also requested is information about the system such as the symbol rate, the number of frames, and the number of data symbols.

The Doppler frequency, as mentioned in Section 5.1, describes the fading rate of the channel and is computed from the velocity of the mobile and the wavelength of the carrier frequency. The symbol rate is needed to determine the sample rate of the system. For mobile systems this number is usually on the order of 10^4 to 10^5. There are again twelve samples per symbol so the sample rate will be twelve times the symbol rate. These rates determine how quickly the channel changes from one sample to the next.

The number of frames determines how many total symbols are sent in the simulation. It should be kept relatively large so that several fades occur during the transmission. This will allow for a more accurate prediction of the BER. The number of data symbols per frame is another important variables in this simulation. Because the equalizer does not work in a decision-directed mode, the equalizer taps predicted by the training sequence will become less reliable as the frame progresses. For this reason, the frames should be kept short, to a few hundred symbols, especially when the channel is changing relatively quickly.

The delay of the second ray determines the delay spread of the channel. The longer the delay, the more severe the ISl will be. The delay should be entered in samples. Since there are twelve samples per symbol, a delay of one sample would cause the delay of the second ray to be 1/12 of the symbol period. A delay of twelve samples would be a delay of one full symbol period. The number of feedback taps in the DFE should be adjusted based on this delay.

Processor

The processor program, called **mbldfeproc.m** takes the parameters set in the preprocessor and runs the simulation of the wireless mobile system using the DFE. The simulation begins by computing several values from the input parameters. These values include the length of each frame, the total number of symbols and samples, and the delay caused by the matched filters. Once these values are found, the simulation computes the fading envelope of the two rays. This is done by passing the sample rate, the Doppler frequency, and the length of the envelope in samples to the function **fademaker.m**. This function uses the method talked about in Section 5.1.3 to determine the fading envelope.

After the envelope of each ray is found, the simulation creates the frames of data that will pass through the system. It begins by finding a random bit sequence that will be used as the training sequence. It then determines a matrix of random bit values. Each row of the matrix will be the data portion of each frame. The complete frames are then found by appending the training sequence to the beginning of each data sequence. All the frames are combined together to form the vector of symbols to be transmitted, called **frames** in the program.

In the next step, the transmitted data sequence is pulse shaped with the square-root raised cosine filter to form the vector, **v**. This vector is then passed to the channel. The output of the channel is created in a few steps. The first is to take the transmitted signal and multiply it with the first ray's channel envelope. The same transmitted signal is delayed by the requested delay and multiplied by the second channel envelope to form the second ray. The two rays are then added together. to form the signal portion of the output of the channel. The entire process is performed with the following line of code.

$$\mathbf{y} = \mathbf{ch1}.*\mathbf{v} + \mathbf{ch2}.*[\mathbf{zeros}(\mathbf{1},\mathbf{delay}),\mathbf{v}(\mathbf{1}:\mathbf{numsym}*\mathbf{sps}+\mathbf{2}*\mathbf{delayMF}-\mathbf{delay})]; \tag{5.14}$$

This forms the signal portion of the output of the channel. The noise portion will be added later on.

It should be noted that only the envelope of each ray was considered when passing the signal through the channel, no phase information was included. It is difficult to model how an actual system would track a two-ray signal like this. The receiver would most likely track the strongest ray. Since this would be difficult to model, it is assumed for the purposes of this simulation that the receiver is perfectly synchronized with the first ray and that the phase of the second ray does not come into play. Though this is a loose assumption to make, it is adequate for the purposes of this simulation.

Once the signal leaves the channel it is passed through the matched filter and then sampled to form the vector **rsig2**. The vector is then normalized so that the energy per bit is set to unity. The noise component of the signal is created by passing white Gaussian noise with the correct variance through the same matched filter, sampling the output, and then adding

to samples to the signal portion of the received signal vector to form the completed received signal vector **r2**. The noise portion of the signal is filtered separately so that the signal portion of the received signal can be normalized before the noise is added. This ensures that the actual E_b/N_0 ratio will be the same as what was requested.

The received signal vector is then passed to the equalizer. The equalizer operates on one frame of data at a time. At the beginning of each frame, the equalizer adapts itself based on knowledge of the training sequence using whichever algorithm was specified. Once the training sequence has run its course, the final tap values are saved. These taps are then used to filter the remaining data included in the frame. For this simulation, perfect decision-feedback is assumed. This means that the feedback portion of the equalizer operates on the actual symbols and not the symbol estimates. Though this is not how a real system works, this method is useful in simulation because it avoids the disastrous effects of error propagation. The BER estimates will be lower than if symbol estimates were used, but it provides more consistent results.

This procedure is repeated for each frame until all the frames have passed through the equalizer. The final step taken by the processor is to calculate the number of errors for both the equalized data and the nonequalized data and compute the BERs for each. Both the number of errors and the BERs will be displayed in the workspace.

postprocessor

The postprocessor included with this simulation, **mbldfepost.m**, is very similar to the one included with the adaptive decision-feedback simulation. Many of the same plots are available, such as constellation plots, a plot of the square error, a plot of the tap convergence, and an animation of the tap weights varying with time. The plots of the taps and of the squared error only include values computed during the final training sequence. Another available plot, exclusive to the simulations of the mobile radio channel, is of the envelopes of the two rays. An example of this is shown below in Figure 5.4.

This plot indicates how strong the direct ray is compared to the interfering ray at any point

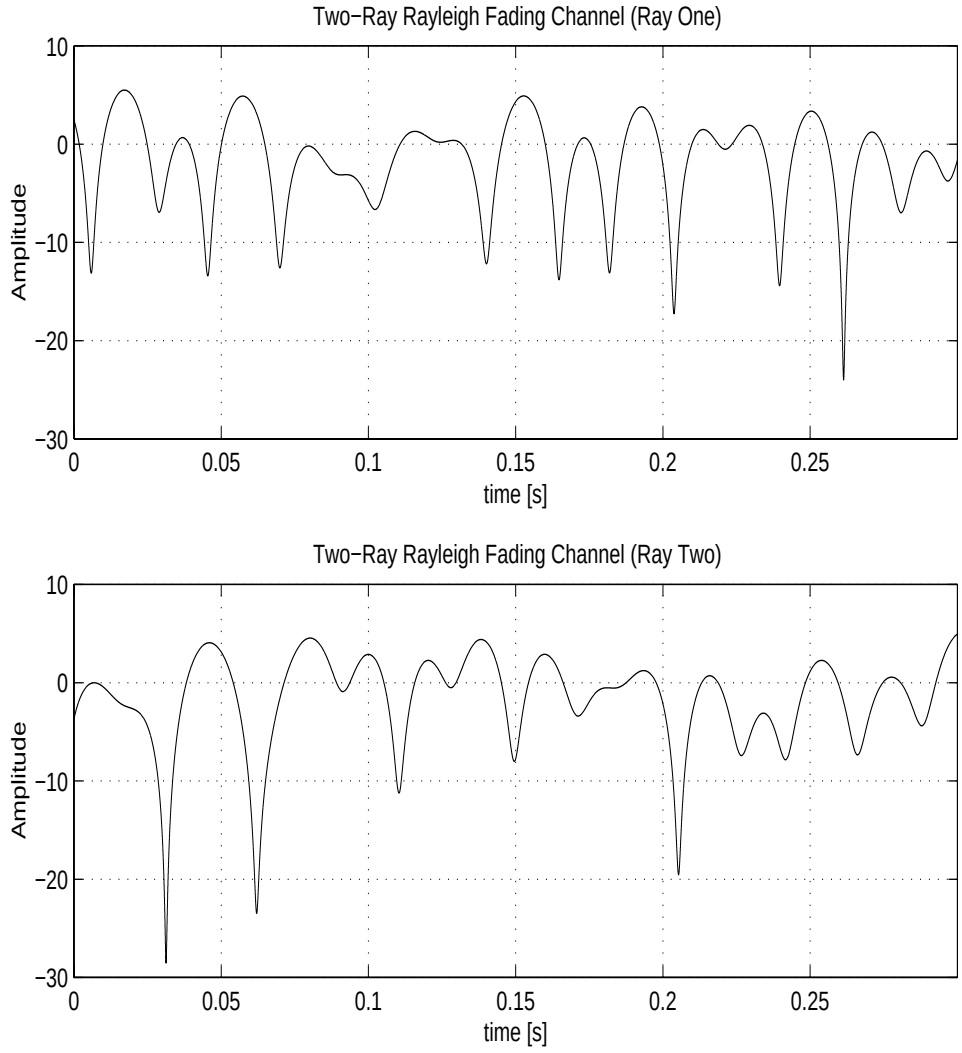

Figure 5.4: Plot of Fading Envelopes used in the Two-Ray Model

in time. The stronger the second ray is, the more severe the instantaneous ISI will be.

Users of the simulation are encouraged to experiment with the many different parameters available to them. Of particular interest is the length of the data sequence in relation to the fading rate. With shorter data lengths, the equalizer is able to train more often and provide more accurate taps for each frame of data. Users should run the simulation for one set fading rate and then vary the data length parameter and observe the change in the BER of the equalized data. The BER should increase significantly as the data length increases.

Another parameter that should be examined is the number of feedback taps. Sometimes when there are more taps than interfering symbols, the equalizer will introduce some of its own ISI because of imperfect tap values. Users should increase the number of feedback taps beyond the delay of the second ray and examine its effect on the performance of the system.

5.3.2 MLSE Equalization for a Mobile Radio Channel

The second simulation included with this chapter models a mobile radio system using an MLSE equalizer to compensate for ISI. It is organized much like the DFE equalizer system. A mobile BPSK system is modeled, but in this case an MLSE equalizer and channel estimator is used in place of the DFE. It uses the two-ray model to represent the channel and also breaks the data down into frames that consist of both the training sequence and data symbols. However, instead of determining equalizer taps during the training sequence, the channel response is estimated so that it can be used by the MLSE equalizer during the data portion of the frame. The direct-form receiver is used in this simulation. This means that the filter in front of the equalizer is fixed and the MLSE equalizer uses a Euclidean distance calculation to compute its metrics.

The preprocessor for this simulation, **mblmlsepre.m** is almost identical to the preprocessor used by the DFE simulation. The only difference is that instead of asking for the number of taps in the feedforward and feedback filters, it asks for the number of ISI terms to account for. All the other parameter requests are exactly the same. Much like in the DFE system, the length of the data sequence is an important parameter to consider. If it is made too

long, the channel estimates will become less and less valid as the frame progresses and the equalizer will not be able to perform as well. For this reason the number of data symbols per frame should be kept to a small number, especially when the fade rate of the channel is high.

The processor of this simulation, called **mblmlseproc.m**, is also very similar to the DFE processor. The differences begin after the received data has been normalized and noise is added with it to form the received data vector, **r2**. At this point each frame of data is cycled through. During the training sequence portion of the frame, channel estimates are made using one of the two adaptive algorithms. The algorithm that is used is set in the preprocessor along with the stepsize parameter that accompanies it.

During the data portion of the frame, the received data is passed to the **mlse.m** function that has been used in all the MLSE simulation included in this book. This function takes the received data and the channel estimates and uses the Viterbi algorithm to determine the most likely sequence of data for that frame. Once this sequence is determined, it is passed back to the processor program. Once all the frames have been equalized, the program compares the estimated data to the actual data and determines how many errors occured. It then computes BERs for both the estimated data using equalization and the symbol estimates that did not use equalization. Both these values are displayed on the workspace when the processor program finishes.

The postprocessor, **mblmlsepost.m** also includes many of the same plots as that of the DFE simulation. The are some differences, however. One difference is that this postprocessor has graphs and an animation of the channel estimates made during the final training sequence in place of plots of the equalizer taps. This program also provides a plot of the MLSE trellis computed during the final frame. This plot is useful because you can see exactly where the equalizer made incorrect decisions and how those errors affected the remaining decisions.

Because this MLSE simulation and the DFE program are so similar, users should use the simulations to compare the performance of the two equalization techniques. This can be done by setting the channel conditions and system parameters to similar values for each

simulation and then determining the equalized data's BER for both systems. Users can also measure and compare the performance of each equalizer after changing parameters such as the amount of noise added to the signal and the length of the data frames. This technique can be used to see if each of these factors has a greater affect on one equalizer structure over the other.

5.3.3 Simulation of a Decision-Feedback Equalizer used in a USDC System

The final simulation included with this chapter models a decision-feedback equalizer used in a system conforming to the United States Digital Cellular System standard. The simulation is similar to the previous DFE simulation, but includes many changes. The first is that QPSK modulation is used in place of BPSK modulation. This is not a drastic change. It just means that there will be two bits per symbol and the symbols will be complex valued. Another difference is that the DFE in this case will operate in a decision-directed mode. This means that the equalizer taps will be continuously updated. The advantage to this method is that the equalizer can use taps based on the current channel conditions instead of using taps computed several symbol periods back. The disadvantage is that this method is much more computationally intensive then the nondecision-directed method, especially when the complex RLS algorithm is used.

The other change is that the default values in the preprocessor, called **mbldfe2pre.m**, are set to the system parameters of the USDC system. This includes setting the symbol rate to 24.3 ksps, the length of the training sequence to 14 symbols, and the length of the data sequence to 148 symbols. The Doppler frequency is set to 60 Hz which is the approximate Doppler shift of a vehicle moving at 75 kph broadcasting on a carrier of 850 MHz. The delay spread of most channels will cause little ISI in a USDC system because of its relatively low symbol rate. However, certain urban channels can cause significant ISI in adjacent symbols. For this reason the delay of the second ray should be kept to around one symbol period (12 samples).

The processor program, **mbldfe2proc.m** of this simulation is identical to that of the pre-

vious DFE simulation until the equalizer section, but even here the differences are small. The normal adaption of the equalizer taps during the training sequence is the same. The change comes during the data sequence. This section is very similar to the training sequence section except that the symbol estimates are used in place of the training symbols when computing the error signal. Once the equalizer finishes, the number of errors are counted and the BER is computed and displayed for both the nonequalized and equalized data.

The plots available with the postprocessor program of this simulation, **mbldfe2post.m**, are the same as those available with the other DFE postprocessor. The only difference is that the plots of the square error, the plot of the taps, and the tap animation, display values for the entire stream of data and not just the training sequence. These are important plots because they display how well the adaption algorithm works with estimated symbols as well as the known symbols of the training sequence. This plot is also useful in understanding how the equalizer works when the direct channel ray is in a deep fade.

An example of the squared-error plot and the tap adjustment plot are shown below in Figures 5.5 and 5.6.

Another postprocessor item of interest is the animation of the taps over time. For the previous DFE simulation, if the training sequence was short, the tap animation only ran for a fraction of a second. Now that there are many more tap vectors, the animation runs for a long time and provides good insight into the adaption of the equalizer.

There are several aspects of this system that can be examined with this simulation. One aspect to look at is how the length of the training sequence affects the performance of the equalizer. The same holds true for the length of the data sequence. Finally, users can also experiment with the two adaptive algorithms to see how each one affects the equalizer's performance.

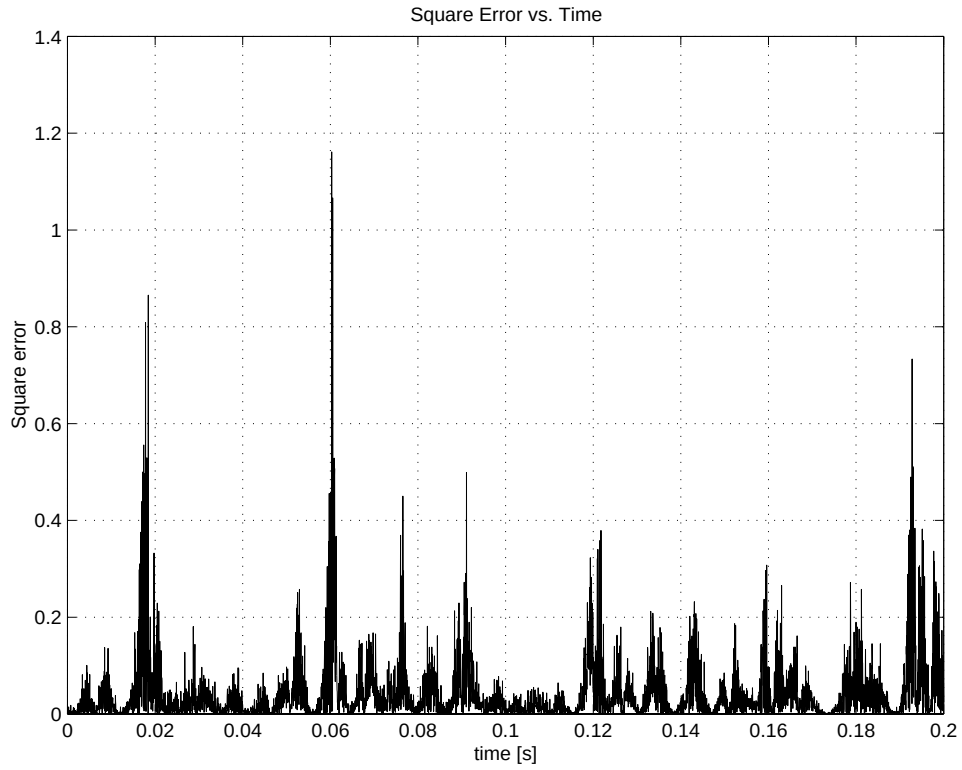

Figure 5.5: Plot of Squared Error for DFE using the RLS Algorithm in a USDC System ($f_m = 60$ Hz, $\tau_2 = T_s$, $\lambda = 0.99$)

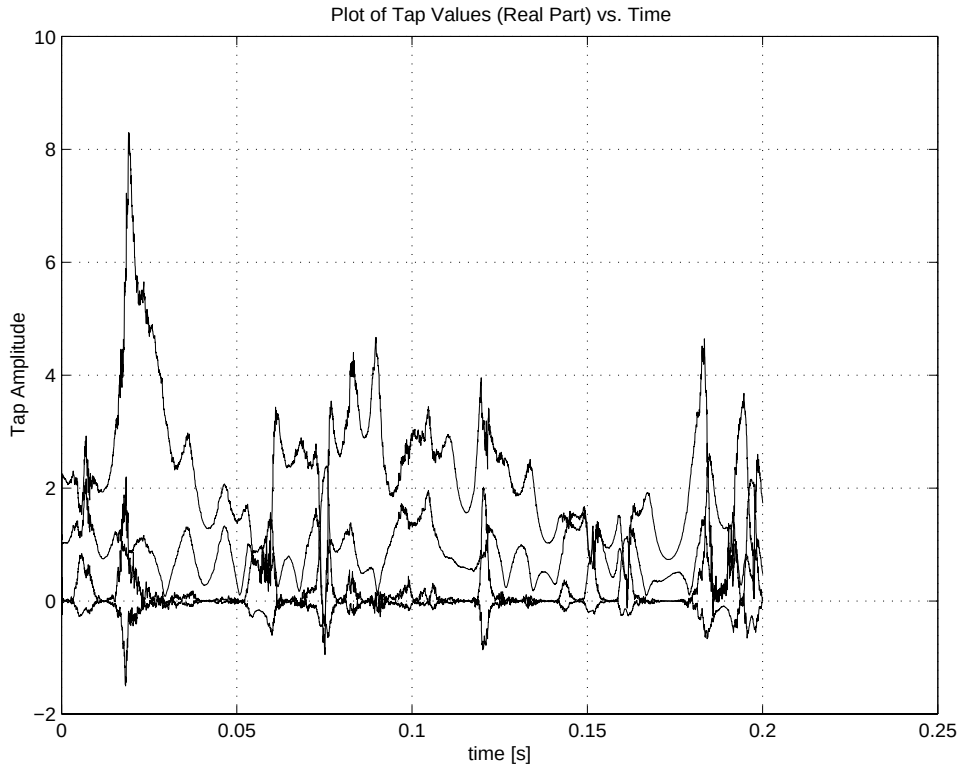

Figure 5.6: Plot of Tap Adjustment for DFE using the RLS Algorithm in a USDC System ($f_m = 60$ Hz, $\tau_2 = T_s$, $\lambda = 0.99$)

Chapter 6

Adaptive Arrays

So far this book has focused primarily on the use of different equalization techniques to compensate for intersymbol interference caused a multipath channel. ISI is not the only phenomenon that can degrade the performance of a wireless communication system. Another major impediment to such a system is co-channel interference. This type of interference occurs when two different transmitters broadcast on a similar carrier frequency in the same geographic vicinity.

Co-channel interference is very common in cellular phone systems where frequencies are reused from one cell to the next. Signals tend to "leak" from nearby cells and interfere with the signals within a cell.

Wireless engineers have developed several techniques to combat this type of interference. One such technique is the use of directional antennas. These antennas are used to divide cells into sectors so that they receive interference and other distortions from only certain directions. This greatly improves the performance of the system. Another technique that is related to sectoring, but is much more powerful is the use of adaptive antenna arrays. Adaptive arrays are better then directional antennas because they are not limited to receive signals in one direction only, but are able to electronically "steer" themselves to minimize interference caused by other mobiles. They can also adapt to combine or reject multipath components of a signal.

Adaptive arrays have other uses in mobile communications. One idea that has just recently evolved is the use of arrays in a multiple access technique called Spatial Division Multiple Access (SDMA). In SDMA mobiles operate on the same carrier frequency and time slot, and are differentiated from each other by their location in the cell. Adaptive arrays are used to track each user and maintain a strong link as the user moves through the cell.

Adaptive arrays are discussed in this book because of the many similarities they share with adaptive equalizers. An adaptive array has a structure very similar to that of a linear equalizer. The difference is that each tap of a linear equalizer operates on signal samples delayed by a certain amount of time while the taps of an adaptive array operate on signals from antennas that are separated by a certain amount of distance. Both techniques also use similar algorithms to adapt themselves. The LMS, RLS, and algorithms based on these two are often used with adaptive array systems.

This chapter will begin with an introduction to antenna arrays and antenna patterns. It will then discuss how these antenna arrays can be made adaptive and how the different adaptive algorithms work with this technique. The chapter will conclude with an explanation on how to simulate adaptive arrays and will examine the simple adaptive array simulations that accompany this chapter.

6.1 Antenna Arrays

An antenna array is formed when several antennas are arranged in a certain geometric pattern and wired together [41]. The received signal from each individual antenna in the array is weighted and then added together with the signals from the other antennas. A directional pattern can be created by controlling the weights and phase shifts applied to the signal of each antenna. This antenna pattern describes which directions the array is the most sensitive. The antennas in an array can be arranged in any pattern, but it is most common to arrange the antennas in a straight line, called a linear array. It is also common to space the antennas equally. An example of a linear array is shown in Figure 6.1.

The pattern which results from an array of antennas is called the array factor. The array

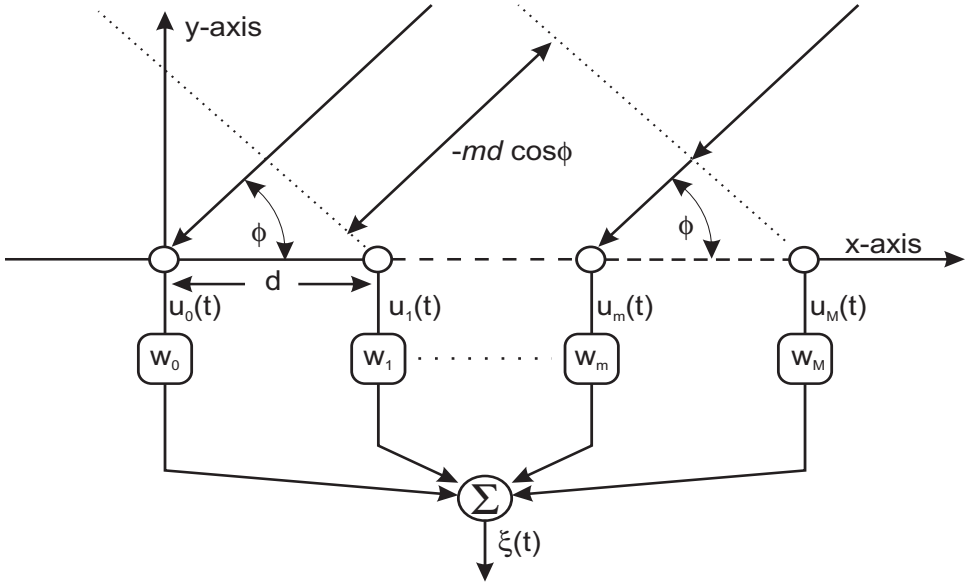

Figure 6.1: A Linear Equally Spaced Array

factor of a linear, equally spaced array is described by:

$$f(\phi) = \sum_{m=0}^{M-1} W_m e^{j\psi_m} \tag{6.1}$$

where

$$\psi_m = m\beta d\cos(\phi) + \alpha_m. \tag{6.2}$$

Here W_m is the weight magnitude placed on the m-th antenna in the array, M is the number of elements in the array, d is the distance between adjacent elements in the array, ϕ is the angle of the incoming wave relative to the axis of the array, and α_m is the added phase shift. The variable β is called the wavenumber and is found from the wavelength, λ, of the incoming signal.

$$\beta = \frac{2\pi}{\lambda} \tag{6.3}$$

The complete antenna pattern of the array is the multiplication of the array factor with the pattern resulting from each individual antenna. The antenna pattern of a dipole antenna is the same over all angles on the plane parallel to the ground. Therefore if dipole antennas are used in the array, the array factor determines the sensitivity of the array at each angle.

An example of an antenna pattern is shown below in Figure 6.2.

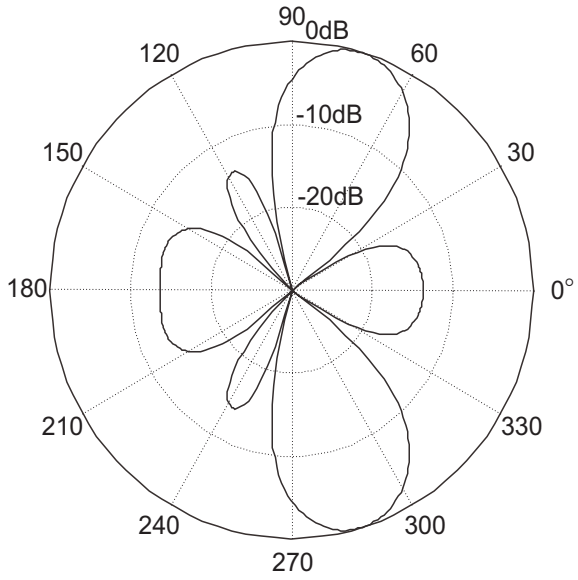

Figure 6.2: Antenna Pattern for a Four Element Linear Array

In this example the antenna pattern has many different beams or lobes. The two main beams extend in the 75° and 285° directions. The antenna is the most sensitive in these directions. In other directions, such as 0° in this example, there exist other beams, called sidelobes, where the antenna is somewhat sensitive, but not nearly as much as in the direction of the main beam. There are also angles for which the signal is significantly attenuated, angles where the received signal is over 30 dB below the power of the main beam. These angles are called the "nulls" of the antenna array. In this antenna pattern, 35° is an example of a null. Notice that the antenna pattern is symmetric about the axis of the antenna. If a null is located at 35° then there will be a null at -35° as well.

This type of filtering is spatial and should not be confused with frequency filtering. The antenna array attenuates signals arriving from selected angles incident upon the array. It is not meant to attenuate certain frequency components of the signals. In a normal antenna array, the weights of each antenna branch are constant values. This causes the antenna pattern to be fixed. With an adaptive antenna, the weights of the elements are adjustable so the antenna pattern can be changed over time. This way the array can be electronically

steered so that the desired signals are in the direction the array is most sensitive and the interfering signals come from directions that are nulled out by the array.

6.2 Adaptive Antenna Arrays

Antenna arrays are made adaptive much the same that equalizers are. Basically, an algorithm is used to adjust the tap values of the array such that a cost function is minimized. A diagram of an adaptive array is shown below in Figure 6.3.

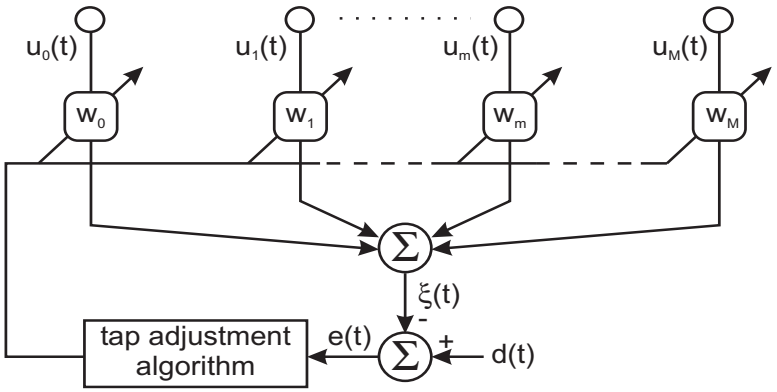

Figure 6.3: Structure of an Adaptive Array

There are several possible cost functions that can be used to adapt the array. For example, the Max SNR technique adapts the array until the maximum SNR is achieved. Linearly Constrained Minimum Variance (LCMV) is another method that can be used to adapt the array [41]. However, both the SNR and LCMV techniques require angle-of-arrival (AOA) information that is not always known beforehand. A technique that does not require this information and the technique most commonly used is the minimum mean squared error (MMSE) technique. This technique maximizes the signal quality by minimizing the difference between the desired output and the output of the array much like the MMSE technique used by an adaptive equalizer.

The MMSE technique attempts to minimize the error between the signal that is received

and the desired signal. The mean-square error is defined as

$$J(\mathbf{w}) = E\left[|\zeta(t) - d(t)|\right]^2. \tag{6.4}$$

In this equation $J(\mathbf{w})$ is the mean-squared error and is a function of the weight vector, $\zeta(t)$ is the output of the array, and $d(t)$ is the desired signal. Most often the desired signal and output signal will be sampled before the MSE is found. In this case the sampled signals $\zeta_n = \zeta(nT)$ and $d_n = d(nT)$, where T is the sample period, will be used in place of $\zeta(t)$ and $d(t)$. An advantage of the MMSE method is that it does not require knowledge of the location and power of the mobiles. A disadvantage of the MMSE technique is that the desired signal or at least a portion of the desired signal needs to be known by the receiver. For this reason, training sequences similar to those used by adaptive equalizers are often sent to adapt the array before any data is transmitted.

6.2.1 Determining Optimal Taps for Minimum Mean-Square Error

When the angle-of-arrival (AOA) and relative power of both the desired user, called the signal-of-interest (SOI), and the interferers, called the signals-not-of-interest (SNOI), are known, the optimal weights for the array can be found. This method is similar to computing the optimal taps for a linear equalizer from the cross-correlation vector and covariance matrix. The following set of linear equations are used to solve for the optimal array taps.

$$\mathbf{a}_k(\phi) = \begin{bmatrix} 1 \\ a_1(\phi_k) \\ \vdots \\ a_{M-1}(\phi_k) \end{bmatrix} = \begin{bmatrix} 1 \\ \exp(-j\beta d\cos(\phi_k)) \\ \vdots \\ \exp(-j\beta d(M-1)\cos(\phi_k)) \end{bmatrix} \tag{6.5}$$

$$\mathbf{R} = 2\begin{bmatrix} \mathbf{a}_0 & \dots & \mathbf{a}_{K-1} \end{bmatrix} \begin{bmatrix} P_0 & & 0 \\ & \ddots & \\ 0 & & P_{K-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_0^H \\ \vdots \\ \mathbf{a}_{K-1}^H \end{bmatrix} + \sigma_n^2\mathbf{I} \tag{6.6}$$

$$\mathbf{p}_k = 2P_k\mathbf{a}_k \tag{6.7}$$

$$\mathbf{w}_k = \mathbf{R}^{-1}\mathbf{p}_k \tag{6.8}$$

In these equations, $a_k(\phi)$ is called the steering vector which describes how the array receives the k-th user, P_k is the received power of the k-th user, σ_n^2 is the variance of the noise, $\mathbf{R}$ is the covariance matrix of the received data, $\mathbf{p}$ is the cross-correlation vector between the received data vector and the desired response, and $\mathbf{w}_k$ is the vector of the optimal complex array weights for user k.

6.2.2 The LMS and RLS Algorithms

It is rare that the AOA and power of the SOI and SNOI are known *a priori*. If they are not known, an adaptive algorithm can be used find the optimal weighting vector. There are several different adaptive algorithms that can be used to adapt the array taps. Many of these algorithms are based on the same LMS and RLS algorithms used by adaptive equalizers. These algorithms are repeated below in the notation associated with the adaptive array system.

Both algorithms begin by computing the received signal, ζ_n.

$$\zeta_n = \mathbf{w}_n^H \mathbf{u}_n \tag{6.9}$$

Here, $\mathbf{u}_n$ is the vector of sampled outputs of each array element at sample point n and $\mathbf{w}_n$ is the vector or array weights.

Next the error between the desired signal and the received signal is found.

$$\varepsilon_n = d_n - \zeta_n \tag{6.10}$$

The LMS algorithm, after this step, updates the tap values.

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \varepsilon_n^* \mathbf{u}_n \tag{6.11}$$

Once again, μ is the stepsize of the algorithm which determines the stability of the algorithm and controls how quickly the algorithm converges and the size of the excess MSE.

The RLS algorithm still has a few more computations before is updates its tap values.

$$\mathbf{k}_n = \frac{\mathbf{P}_{n-1}\mathbf{u}_n}{\lambda + \mathbf{u}_n^H \mathbf{P}_{n-1}\mathbf{u}_n} \tag{6.12}$$

$$\mathbf{P}_n = \lambda^{-1}\left[\mathbf{P}_{n-1} - \mathbf{k}_n \mathbf{u}_n^H \mathbf{P}_{n-1}\right] \tag{6.13}$$

Once these values are determined, the tap values are updated with the following.

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mathbf{k}_n \varepsilon_n^* \tag{6.14}$$

Both the $\mathbf{P}_n$ and $\mathbf{w}_n$ vectors are initialized before the algorithm begins. The forgetting factor, λ, affects the algorithm much the same way it does in an adaptive equalizer. It is usually kept close to 1.0 for stationary cases, but can be reduced in nonstationary cases to provide better tracking.

6.3 Simulation of Adaptive Arrays

Two MATLAB simulations are included with this chapter that model the techniques and algorithms of an adaptive array system. The first simulation takes the angle-of-arrival and power information about the desired user and interferers and finds the optimal array weights that minimizes the mean-square error between the desired signal and the received signal. In the second simulation, the system adapts the tap weights using either the LMS or RLS algorithm to find the minimum mean-square error. The system does not require AOA or signal strength information about the mobiles in the environment, but does require a training sequence to adapt the taps.

6.3.1 Finding the Optimal Weights

This first simulation is a relatively simple one that uses Equations 6.5-6.8 to find the optimal weights for the array. The simulation does not model any type of communication system where bits are being tranmitted but rather finds the antenna pattern that would minimize the MSE between a desired signal and the received signal. To solve for the optimal array weights, the simulation requires the AOA and power level of both the desired user and interferer. Also needed is information about the array such as the number of elements and the spacing between the elements. This information can be entered by the user in the pre-

processor program for this simulation called **arraypre.m**. An example of the preprocessor is shown below.

```
>> arraypre
Enter number of mobiles [4]:  3
Enter SNR of desired user in dB [20]:
Enter Angle of Arrival of desired user in degrees [90]:  75
Enter SNR of interferer #2 in dB [20]:
Enter angle of interferer #2 in degrees [55]:  45
Enter SNR of interferer #3 in dB [20]:
Enter angle of interferer #3 in degrees [110]:  145
Enter number of array elements [4]:
Enter spacing of antenna elements in fraction of wavelengths [0.5]:
>>
```

What is important when solving for the optimal array weights is the power ratio between the different users and the noise level. This is why the signal-to-noise ratio of each mobile is requested. The SNR that is requested is the value that would result if a single omnidirectional antenna was used with the system instead of an antenna array. The noise level is considered the same for each received signal so the power ratio between the interferers and the desired user can be found by comparing the different SNR values.

Besides the power levels of the mobiles and the noise, the preprocessor also requests information about the array such as the spacing between the elements of the array. This spacing should be entered in fractions of wavelengths. This means that if the spacing is half a wavelength then 0.5 should be entered. If the spacing is a quarter wavelength then 0.25 should be entered.

With all the parameters entered, the processor program, **arrayproc.m** is used to find the optimal weights for the array. The program begins by finding the variance of the noise. The power of the desired user is set to unity for convenience. This means that the variance of the noise changes depending on the SNR of the desired user. The powers of each interferer

is set based on the difference between the SNR of the desired user and the SNR of the individual interferer.

Once the noise variance and power of each user is found, the processor computes the **P** and **a** matrices based on these values. The **P** matrix is all zeros except on the diagonal which holds the power of each user. The **a** matrix holds the steering vectors for each user. It is found using Equation 6.5. After these two matrices are found, the program determines the covariance matrix of the received data, **R**, and the cross-correlation vector of the desired data with the received data, **p**. This is done with the following two commands.

$$\mathbf{R} = \mathbf{2} * \mathbf{a} * \mathbf{P} * \mathbf{a}' + \mathbf{nvar} * \mathbf{eye}(\mathbf{M}); \tag{6.15}$$

$$\mathbf{p} = \mathbf{2} * \mathbf{pwr}(\mathbf{1}). * \mathbf{a}(:, \mathbf{1}); \tag{6.16}$$

The **pwr(1)** variable holds the power of the desired user and the **a(:,1)** vector holds the steering vector of that user.

The optimal array weights are then determined by multiplying the inverse of **R** with **p** as given in Equation 6.8.

Now that the optimal taps have been found, it is useful to determine the antenna pattern of the array using these taps. To do this, steering vectors are found for angles of each degree around the array for a total of 360 steering vectors. These vectors are stored together in the matrix **aa**. To find the array factor, **f**, the vector of optimal weights is multiplied with this matrix. The resulting vector describes the sensitivity of the array in each direction.

This vector can be used to determine the difference in sensitivity in the direction of the desired user with the sensitivity in the direction of the interferers. This difference is displayed on the workspace once the processor has finished. An example is shown below.

```
>> arrayproc
CI =
-0.0896 -75.5471 -75.8495


CI2 =
```

```
0 -75.4576 -75.7599

>>
```

The first vector that is displayed, **CI**, is the attenuation of each user compared with the maximum gain of the array in dB. The first value is the attenuation of the desired user and the remaining values are the attenuation of the interferers. The second vector, **CI2**, displays the attenuation of each user compared to the desired user. Again, the first value is the desired user, which will always be zero, and the remaining values are the relative attenuation of each interferer.

The resulting antenna pattern of the array using the optimal weights can be viewed with the postprocessor of the simulation, **arraypost.m**. The values of the optimal array weights can be viewed as well. An example of the antenna pattern plot is shown in Figure 6.4. This is the antenna pattern that results from the parameters set in the preprocessor shown previously.

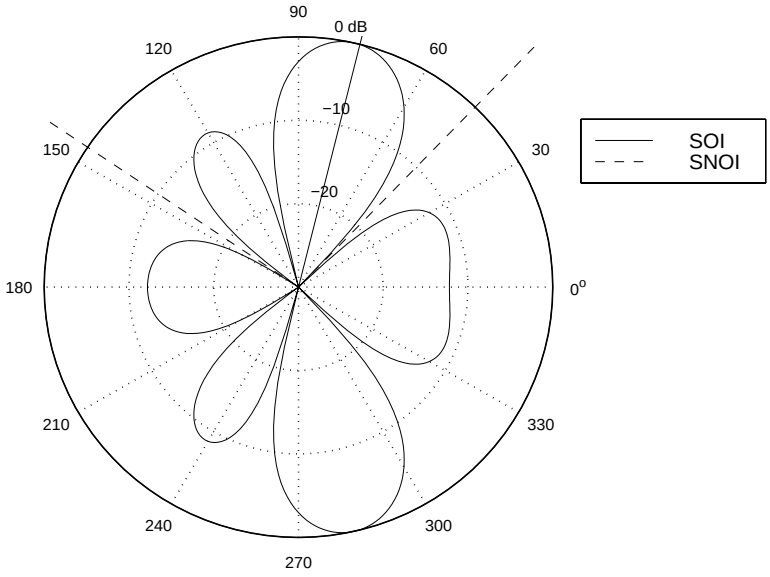

Figure 6.4: Optimal Antenna Pattern

In this plot, the angle of arrival of the desired user is shown as a solid line and the angles of arrival of the interferers is represented with a dashed line. In this example, the array is

able to place its main lobe in the direction of the desired signal and its nulls in the direction of the interferers.

There are several parameters readers can experiment with in this simulation. One good experiment is to vary the number of elements in the array while keeping the number of mobiles constant. When there are more mobiles then elements, the array is considered *overloaded* and can have difficulty nulling out all of the interferers. When there are more array elements than mobiles, the array is considered *underloaded* and can attenuate interferers much more.

6.3.2 Simulation of an Adaptive Array

Also included with this chapter is a simulation of an array that uses an adaptive algorithm to find the best possible antenna pattern. The system being modeled in this simulation uses a training sequence to adapt the array weights. This simulation differs from the previous one in that it models data being received by the antenna. The performance of the array system can therefore be measured through both the antenna pattern and BER calculations.

The preprocessor for this simulation, called **aarraypre.m**, requests all the same information as the previous preprocessor. However, this preprocessor also requests information about the adaptive algorithm. This information includes the type of algorithm used, either LMS or RLS, and the parameter for each algorithm. It also request the length of the training sequence and how many actual symbols should be run by the processor program in addition to the training sequence.

The processor program, **aarrayproc.m** begins by determining the power levels of the interferers to the desired user and the noise variance much the same way that the optimal array processor began. Once this is done, a matrix of random data is created to represent the transmitted data of each mobile. All the users are considered synchronized in time. For this reason only one sample per symbol is used and no pulse shaping is used.

The data is then scaled so that the received data from each user will have the SNR taht was requested in the preprocessor. Next the steering vector for each user is found and combined with the other vectors into the matrix **a**. This matrix is then multiplied with the matrix

of the received data and noise is added to determine the signal received by each element in the array. Then the adaptive algorithms are used to find the weights that provide the minimum MSE for the array.

The adaptive algorithms run much the same way they did in the adaptive equalizer simulations. The LMS algorithm in the simulation follows Equations 6.9 through 6.11. The RLS algorithm uses Equations 6.9, 6.10, and 6.12 - 6.14. The adaptive algorithms in the processor saves both the error vector and the matrix of each array weight over time.

When the adaptive algorithm has finished, the antenna pattern is found from the final set of array weights. This is done by finding the steering vectors for each degree around the array and multiplying it by the weights just like it was done in the optimal array simulation. The attenuation of the interferers is also found and displayed on the workspace.

The final steps of the simulation uses the final adapted array weights to spatially filters the data unknown by the receiver. Decisions are made from this filtered data and the BER of the desired user is calculated based on these symbol estimates. This value is also displayed on the workspace.

The postprocessor of this simulation includes the same plots as the optimal array postprocessor, but also includes plots that describe the data while the array was adapting. This includes plots of the squared error and array weights during the training sequence as well as a plots of different antenna patterns during the convergence of the adaptive algorithm. An example of this plot is shown in Figure 6.5. In this case the LMS algorithm with a step-size of 0.01 was used.

This plot is very useful in determining the effectiveness of the adaptive algorithm in converging to the best weights. Users can use this graph to see how quickly the algorithm converges using various step-sizes. The **CI** and **CI2** vectors displayed after the processor finishes can also be used to determine how well the algorithm performed. This performance measure is comparable to the excess MSE of an adaptive equalizer. Usually, the lower the step-size the more the interferers will be attenuated, given that enough time has passed for the weights to converge to their final values.

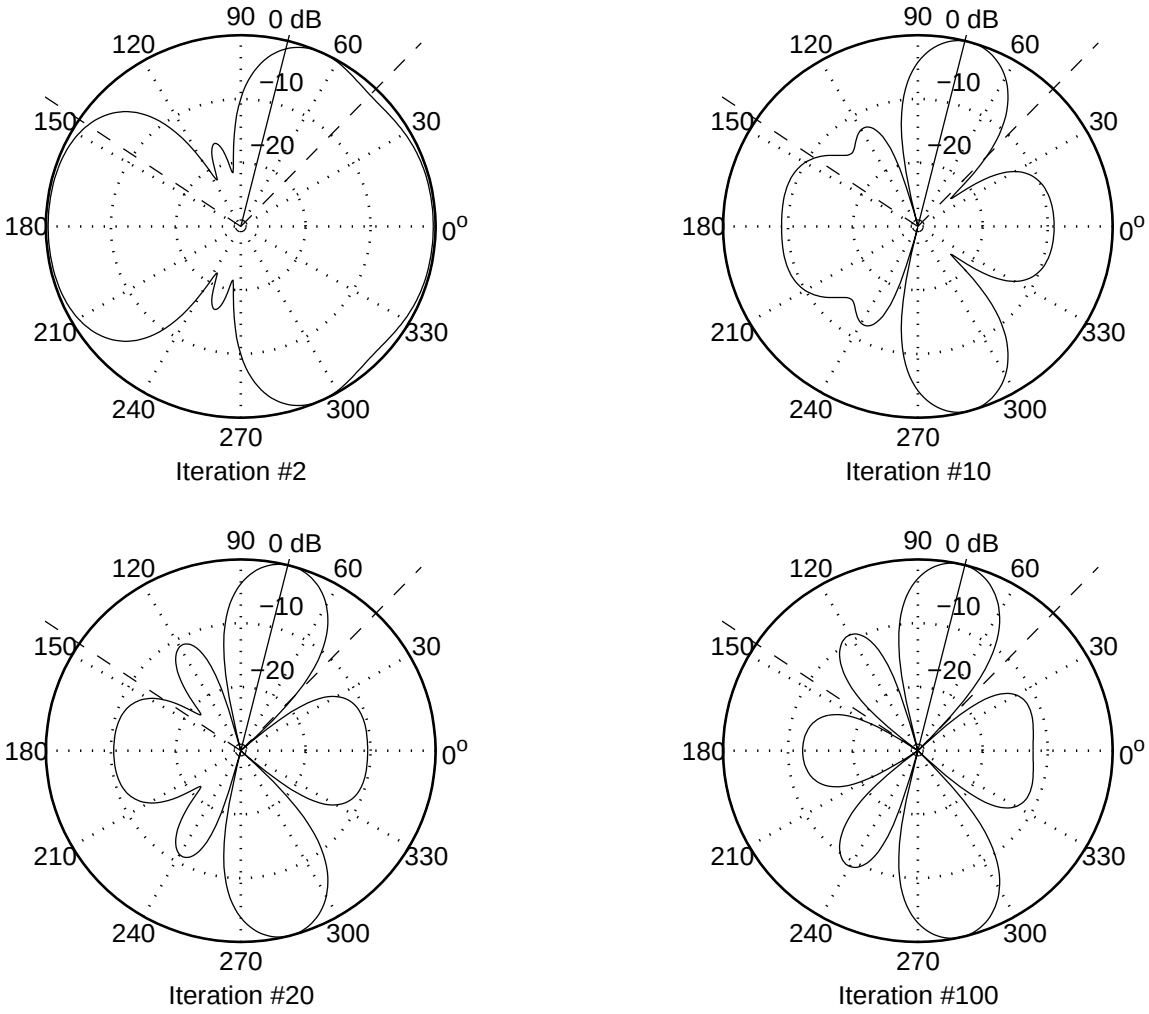

Figure 6.5: Convergence to the Final antenna pattern

Another simulation setting that can be experimented with is the number of array elements relative to the number of users. The performance of the array in underloaded and overloaded situations is very important. One final suggested experiment is to vary the angle of the desired user so that it is close to the angle of an interferer. When an array has only a few elements it is difficult to form a main lobe that is very narrow. In a situation where the desired user is at an angle similar to an interferer, the array will have a difficult time attenuating the interferer and not the desired user because it will try to place a null and the main lobe in the same direction.

Chapter 7

Equalization for Indoor Wireless Channels

The need for wireless transmission of digital data in an indoor environment has grown dramatically in the last few years. Wireless local area networks (WLANs) allow users to freely move about an office, factory, or other building while still connected to their network. However, to support the many multimedia applications needed by the user, these WLAN systems will need high speed data links to compete against their wired cousins. A standard has already been defined in Europe for such a high speed wireless network. The HIgh PErformance Radio LAN (HIPERLAN) standard operates at 5.2 GHz and provides data rates up to 24 Mbps using GMSK modulation [12] [42].

At such high data rates (10 - 30 Mbps), ISI caused by the indoor channel becomes a major limitation. The RMS delay spread of most indoor channels is usually on the order of tens of nanoseconds. In large open buildings such as factories, the delay spread can reach up to 150 ns [8]. In such a case the data rate can only reach as high as 667 ksps without ISI becoming a factor if the $\sigma_N < 0.1$ criterion is used. Therefore, to achieve data rates larger than this amount, an adaptive equalizer must be used to compensate for the ISI [43].

In this chapter several of the equalization techniques discussed previously will be applied to the problem of high speed data transmission in an indoor environment. Several simulations

that model wireless indoor high speed data systems using equalizers have been developed and executed. Both decision-feedback and MLSE equalization has been used in these simulations to compensate for the ISI introduced by the indoor channel. Models of several stationary channels, where the receiver and transmitter are fixed, and many mobile channels, where either the receiver or transmitter is moving, are used in the simulations. The performance of each equalization type and a measure of the complexity of each equalization technique is found for each of the several different channel models. This chapter presents the simulation procedure used to model these systems, the results of these simulations, and analysis of these results.

7.1 The Indoor Channel

In an indoor radio channel, reflections from the walls, ceilings, furniture, people, and other obstacles close to the receive and transmit antennas cause multipath. However, because buildings are enclosed, the delay of individual multipath components will be much smaller in an indoor channel compared to an outdoor channel. The typical RMS delay spread of a radio channel in a factory building is around 100 ns with 90% of the channels having delay spreads less than 150 ns [8]. RMS delay spreads of channels in office buildings will be smaller.

The indoor channel can be represented as the sum of several delta functions with different delays.

$$g(\tau) = \sum_{k=0}^{K} \alpha_k e^{j\phi_k} \delta(\tau - \tau_k) \tag{7.1}$$

The value of the multipath components' magnitude, α_k, the phase ϕ_k, and the delay, τ_k, are random variables. The exact distribution of these variables depends on factors such as T-R separation, the type of building the system operates in, and the frequency that the system transmits on.

Since most indoor wireless systems are stationary or move only at pedestrian speeds (<1 m/s), the indoor channel changes very slowly. In high speed systems (R_b >1 Mbps), the channel will appear stationary over several thousands of symbols. This is beneficial to

a system using an equalizer because the equalizer can be trained at the beginning of a frame and won't have to be retrained for several thousands of bits. This will increase the throughput of the system. This will also cause the tracking error of the equalizer to be very small. This allows the equalizer to train only at the beginning of the frame instead of continuously adapting over the entire frame of data.

7.2 Simulation of Equalization for Indoor Channels

Several simulations of high speed wireless systems operating in an indoor channel were developed to measure the effectiveness of different equalization techniques in reducing the ISI caused by such a channel. Simulations of systems employing both DFE and MLSE equalizers were performed using channel models representing both stationary and slowly varying channels were used. The following sections describe how these channel models were developed, the system and simulation characteristics that were used, and the results of the simulations. These results include both BER plots that measure the performance of the systems using the equalizers as well as the computational requirements of the different equalizers.

7.2.1 System Model

The wireless system model used in the simulations is implemented in baseband. The complex baseband envelope is used to represent the RF signal. The channel models are also implemented in baseband. Data is transmitted using BPSK modulation. A square-root raised cosine filter is used to pulse shape the signal in the transmitter and another is used as a matched filter in the receiver. Both filters have a rolloff factor, α, equal to 0.3.

A block diagram of the system model is shown below.

The overall channel model in this case is made up of the combined response of the pulse shape filter, $p_{ps}(\tau)$, the channel filter, $g(\tau)$, and the matched filter, $p_{mf}(\tau)$, as shown below.

$$q(\tau) = p_{ps}(\tau) * g(\tau) * p_{mf}(\tau) \tag{7.2}$$

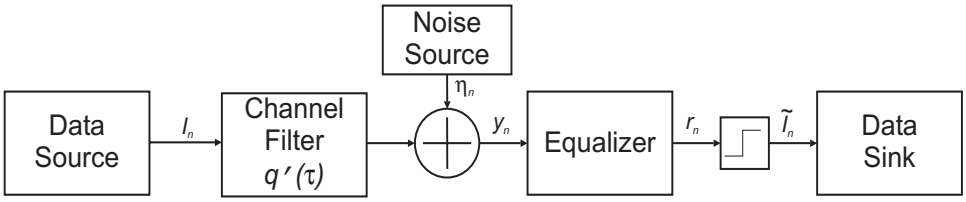

Figure 7.1: Block Diagram of a Indoor Wireless System

This response is then sampled at the symbol rate to find the impulse response seen by the receiver.

$$q'(\tau) = q(nT_s) \tag{7.3}$$

This is the filter that is used to model the overall channel in the simulations. It operates at the symbol rate which allows for smaller simulation runs since only one sample is needed per symbol. In the simulations this response is normalized so that the path loss is 0 dB.

Once the data passes through the overall channel filter, noise is added. The noise variance will be set based on the E_b/N_0 ratio that is desired.

$$\sigma_\eta^2 = \frac{1}{2\left(\frac{E_b}{N_0}\right)} \tag{7.4}$$

The signal is then passed to the equalizer. The output of the equalizer is then passed to a decision device if the DFE equalizer is used. Because the MLSE equalizer makes bit decisions internally, no decision device is needed.

7.2.2 Channel Models

Two different types of channel models were developed for these simulations. The first models a stationary channel where the transmitter and receiver do not move. This would be the case where a receiver is placed on a desk or other surface. The second type of channel that is modeled is a channel where the receiver or transmitter is moving at pedestrian speeds. This type of channel would result if a person was walking with the receiver or transmitter or if the transmitter or receiver was attached to some kind of mobile machine such as a robot or mobile factory machinery.

Stationary Channel Model

The stationary channel models used in the simulations are based on the channel models developed in Section 2.3. This is a twelve-ray model where there are twelve paths uniformly distributed per symbol period. The path magnitudes are Rayleigh random variables with an exponential distribution that decays as the excess delay increases. The phases have a uniform distribution between 0 and 2π.

Three sets of channel responses were generated. The first set had a normalized delay spread of 1.0. This means the symbol period is equal to the RMS delay spread of the channel. The two other sets had normalized delay spreads of 0.5 and 2.0 respectively. Channels with these delay spreads were found by changing the decay factor of the exponential magnitude distribution. Through trial and error, a decay factor was found that generated channels that had delay spreads close to the delay spread that was desired. Once this decay factor was found, fifty channels were generated. From these fifty channels, the corresponding overall channel, $q'(\tau)$ was found. Since BPSK modulation is used, only the real part of this channel is needed. The delay spread of this channel is then computed. From the set of fifty channels, the four channels whose delay spread was closest to the desired delay spread of 1.0, 0.5, or 2.0 was saved for use in the simulations. In almost all the cases the actual delay spread was within 2% of the desired delay spread.

Time Varying Channel Model

To generate channel models that represent a indoor mobile radio channel, the SIRCIM 2.0 software package was used. An educational version that works with MATLAB 4.2 is available over the web from the Mobile and Portable Radio Research Group (MPRG) of Virginia Tech. The SIRCIM program generates indoor channel models based on many different wideband channel measurements made in several different types of buildings. The program prompts the user for certain parameters and then creates a channel response with the same ensemble statistics as actual measured data [39].

The parameters requested by SIRCIM include the T-R separation, the speed of the mobile,

how obstructed the channel is, and the frequency of the carrier wave. From this data, a set of 19 channel profiles are created. These profiles represent the channel as a receiver moves along a track that is four-and-a-half wavelengths long. The profiles are spaced a quarter wavelength apart. Each profile is broken down into 64 bins with a resolution of 7.8 ns. Each bin contains both magnitude and phase information. This information is correlated with the same bin in adjacent profiles. Wideband fading is included with the set of channel profiles. Each profile has a different path loss.

Much like with the stationary channel models, the corresponding sampled overall response was found for each profile. The profiles were normalized so that the average path loss over all the profiles is equal to 0 dB. This will make calculating the noise variance for different E_b/N_0 much easier. The average delay spread of each set of overall channel profiles was found.

Two sets of channel profiles were selected from 20 different sets of profiles. The first set of indoor channel profiles had an average delay spread of 93.69 ns. These profiles resulted from a 70% obstructed channel, with an average T-R separation of 29.11 m, a carrier frequency of 1.3 GHz, and a velocity of 1 m/s. This set of profiles is shown in Figure 7.2.

The average delay spread reported in the plot is for the actual set of profiles. The delay spread of the sampled overall response is actually 98ns.

The second set of channel profiles saved have an average delay spread of 65.77 ns. These channel profiles were created by setting the T-R separation to 22 m, the carrier frequency to 1.3 GHz, the obstruction percentage to 30%, and the velocity to 1 m/s. This set of profiles is shown in Figure 7.3.

The average delay spread of the sampled overall response is very close to that of the actual channel at 66 ns.

7.2.3 Adaptive Equalizers

Both decision-feedback equalization and MLSE equalization were used in the simulations to compensate for ISI. The DFE uses a feedforward filter with symbol-spaces taps. The

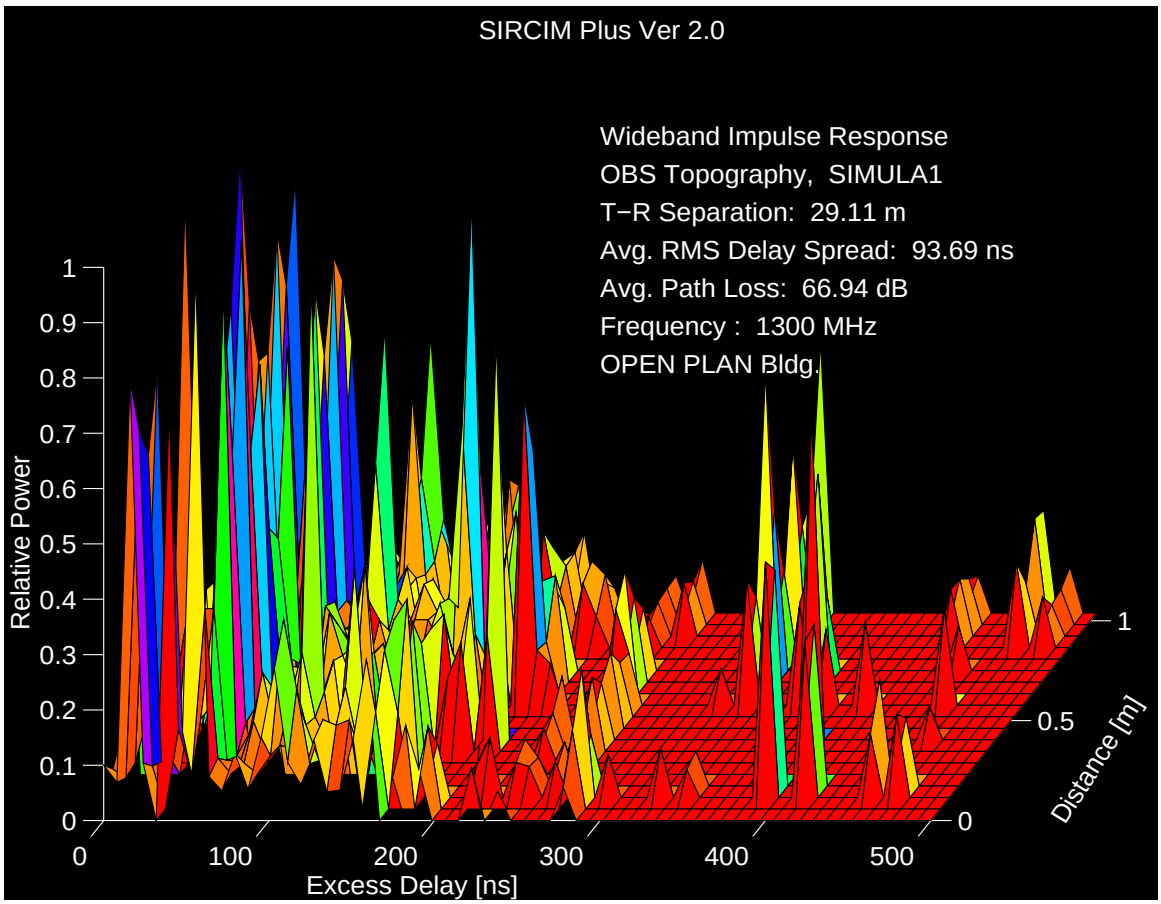

Figure 7.2: Channel Profiles for Indoor Channel Generated with SIRCIM with Average σ_τ = 98 ns

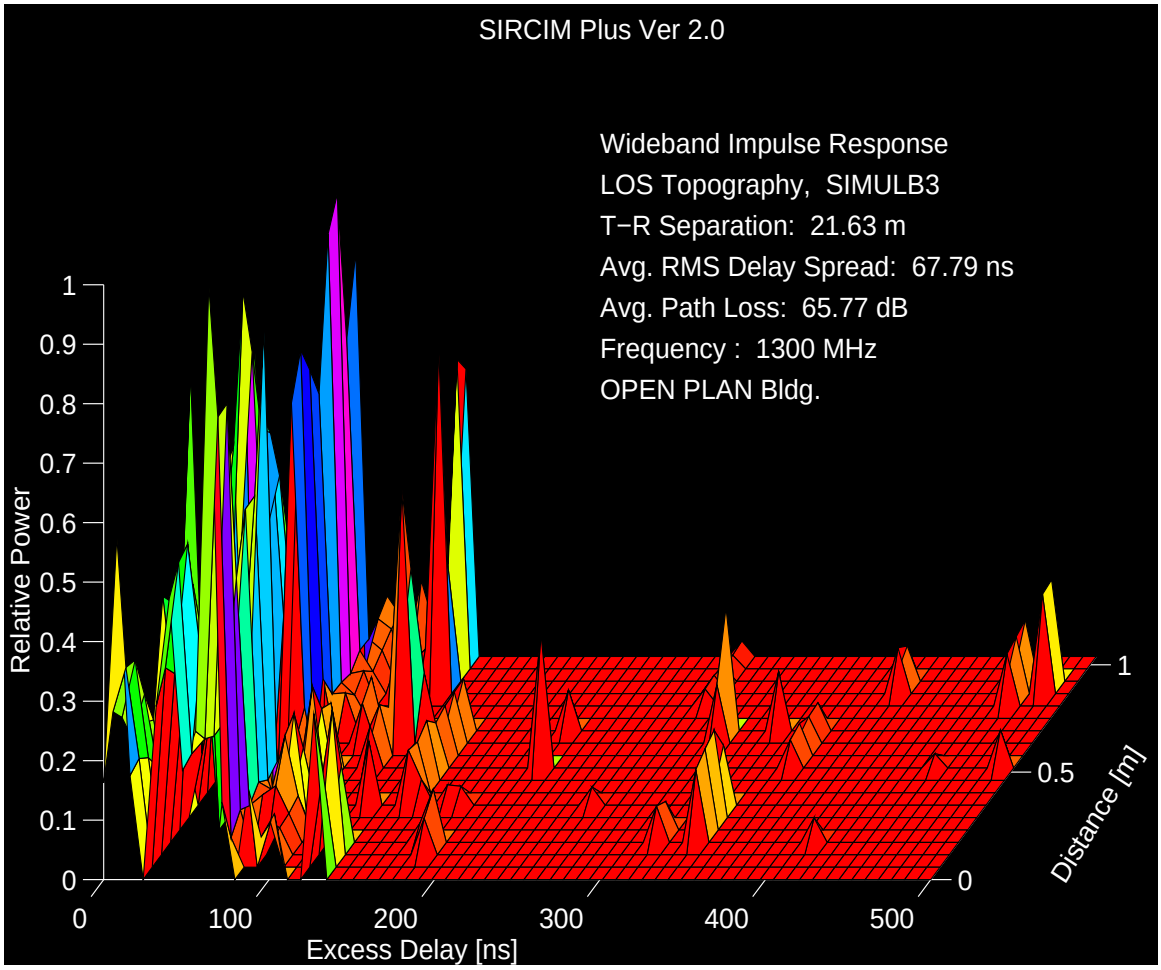

Figure 7.3: Channel Profiles for Indoor Channel Generated with SIRCIM with Average σ_τ = 66 ns

equalizer is synchronized to last tap in the feedforward filter. The number of taps in the feedback and feedforward filter is variable. How these numbers are chosen will be discussed in the next section.

The DFE equalizer uses the RLS algorithm to adapt itself. A training sequence of 50 symbols is included with the data stream before any data is sent. The forgetting factor, λ, is set to 0.99. When the training sequence is over, the final set of tap values found from the adaptive algorithm is saved. These values are then used to equalize the received data.

The MLSE equalizer operates on data frames that contain 1000 symbols. The direct-form receiver is used. This means that only the fixed matched-filter precedes the equalizer, no adaptive filter is included. The Euclidean-distance metric is used to compute the metrics for each state. The number of symbols included in each state, L, is variable. At the end of each data frame, L extra symbols are added to force the MLSE trellis into a known state. The path that ends in this state is chosen as the most likely sequence of bits.

The MLSE equalizer operates on channel estimates found from an adaptive channel estimator. The estimator uses the RLS algorithm with $\lambda = 0.99$. The training sequence is 50 symbols long. The final vector of channel estimates is saved and used to by the equalizer.

7.3 Simulation Procedure

The first set of simulations that were ran were those using the stationary channel models. For all these simulations, each channel profile in each set of channels with similar delay spreads was simulated three times. The BER for these twelve runs was then averaged. The performance of each equalizer structure was measured for E_b/N_0 values of 3, 6, 9, and 12 dB. For the 3 and 6 dB simulations, 10^3 symbols were simulated for each run of each channel profile. The 9 and 12 dB simulations ran 10^4 and 10^5 symbols, respectively, in most cases. For equalizers that provided extremely good performance at these noise levels, the number of symbols was increased to provide better accuracy.

The simulations of the mobile radio channels differed slightly from those of the stationary

channels. Even though the channel is changing with time in this case, it was determined that at pedestrian speeds of 1 m/s the channel changes so slowly compared to the frame rate, that during this frame the channel can be modeled as stationary. This is true for data rates on the order or 10 Msps and frame sizes on the order of 1000 symbols.

In order to determine all the different channel responses that each frame of data would see while the receiver moves, tens of thousands of profiles would have to be interpolated from the 19 channel profiles provided by SIRCIM. However, this is not necessary. A good performance measure of the equalizers as the receiver moves along the 4.5λ track can be found from the 19 profiles themselves. To measure this performance, each channel profile is simulated and the resulting BERs are averaged together. This is done for both sets of channel profiles, the set with an average delay spread of 98 ns and the set with an average delay spread of 66 ns.

To determine the optimal DFE setup for each set of channel profiles with similar delay spreads, several different feedforward and feedback filter lengths were tried. The first set of simulations kept the number of feedforward taps constant and varied the number of feedback taps. From the output of these simulations, the smallest number of feedback taps that provided the best performance was determined. The length of the feedback filter was then set to this number, and another simulation was run that varied the number of feedforward taps. The optimal DFE was then found from the results of this simulation.

The simulations of the MLSE equalizer also used this method of diminishing returns. In these simulations, the value of L was increased until the BER of the system no longer improved. The MLSE equalizer using the smallest value of L that provided the best performance was then determined to be the optimal equalizer for channels with this delay profile.

7.4 Simulation Results and Analysis

The first set of simulations that were conducted determined the performance of a DFE and MLSE equalizer for a stationary indoor channel with a normalized delay spread of 1.0.

The results of these simulations are shown in Figures 7.4 - 7.6. Figure 7.7 compares the performance of the optimal DFE setup for this channel compared with that of the optimal MLSE setup. The first graph indicates that three is the best number of feedback taps for the DFE equalizer. The second graph indicates that the DFE(5,3) structure is the optimal DFE structure for this set of channels. The third graph indicates that the MLSE(4) structure provides the best performance for this equalization technique. The fourth graph indicates that the MLSE(4) equalizer provides better performance than the DFE(5,3) by between 1 and 2 dB.

The second set of simulations measured the performance of the two equalization techniques used for stationary profiles with a normalized delay spread of 0.5. The results of these simulations are shown in Figures 7.8 - 7.11. In this case, the DFE(3,2) and the MLSE(3) structures provide the best results. The MLSE(3) algorithm provides only slightly better performance than the DFE(3,2) equalizer, both of which provide performance only about 1 dB off that of a system that has no ISI.

The third set of simulations is for stationary channel profiles with $\sigma_N = 2.0$. In this case the best DFE structure was the DFE(5,5) and the best MLSE structure was the very complex MLSE(6). In this case, the DFE has a very hard time equalizing the channel even with a large feedback filter. The MLSE provides much better performance, up to 6 dB better, but can only provide this performance with a very large equalizer.

The fourth round of simulations model a system operating at 10 Mbps over a mobile channel with an average RMS delay spread of 98 ns. This corresponds to an average normalized delay spread of 0.98. In this case the DFE(5,5) was needed to provide the best DFE performance and the MLSE(4) provided the best performance for that structure. In this case the optimal MLSE equalizer beats the optimal DFE equalizer by about 3 dB. These results match up very closely with the results of a simulation of a DFE using SIRCIM channel models given in **??**. The simulation included in that work used a channel with an average delay spread of 95.6 ns, very close to the average delay spread of the channels used here. The major difference is that his simulation used QPSK modulation and not BPSK modulation. However, a 20 Mbps QPSK system would be very similar to the 10 Mbps

system used here. The optimal DFE found in the QPSK case was a DFE(6,5). Very close to the DFE(5,5) found in this BPSK simulation.

The results of the final simulations are shown in Figures 7.20 - 7.23. These simulations used channel profiles from an indoor mobile radio channel with an average delay spread of 66 ns. Again, the data rate was 10 Mbps which means $\sigma_N = 0.66$. For this channel, the DFE(5,4) and MLSE(5) equalizer structures provided the best performance. The MLSE equalizer provides better performance compared to the DFE for most E_b/N_0 values except when the ratio is relatively high ($>$ 10 dB). At higher ratios, the BER of the MLSE equalizer in this case bottoms out while the BER of the DFE keeps improving for higher E_b/N_0 values. The reason why the MLSE performance bottoms out has to do with the condition of the channel modeled in this set of simulations. The SIRCIM program includes wideband fading when determining the channel profiles. In this case many of the profiles are in a deep fade. In deeps fades such as this, the adaptive algorithm will have a difficult time finding the optimal channel estimates. Since the MLSE equalizer is much more sensitive to poor channel estimates than the DFE is to poor tap estimates, the performance of the MLSE equalizer can end up being worse than that of the DFE as it is here.

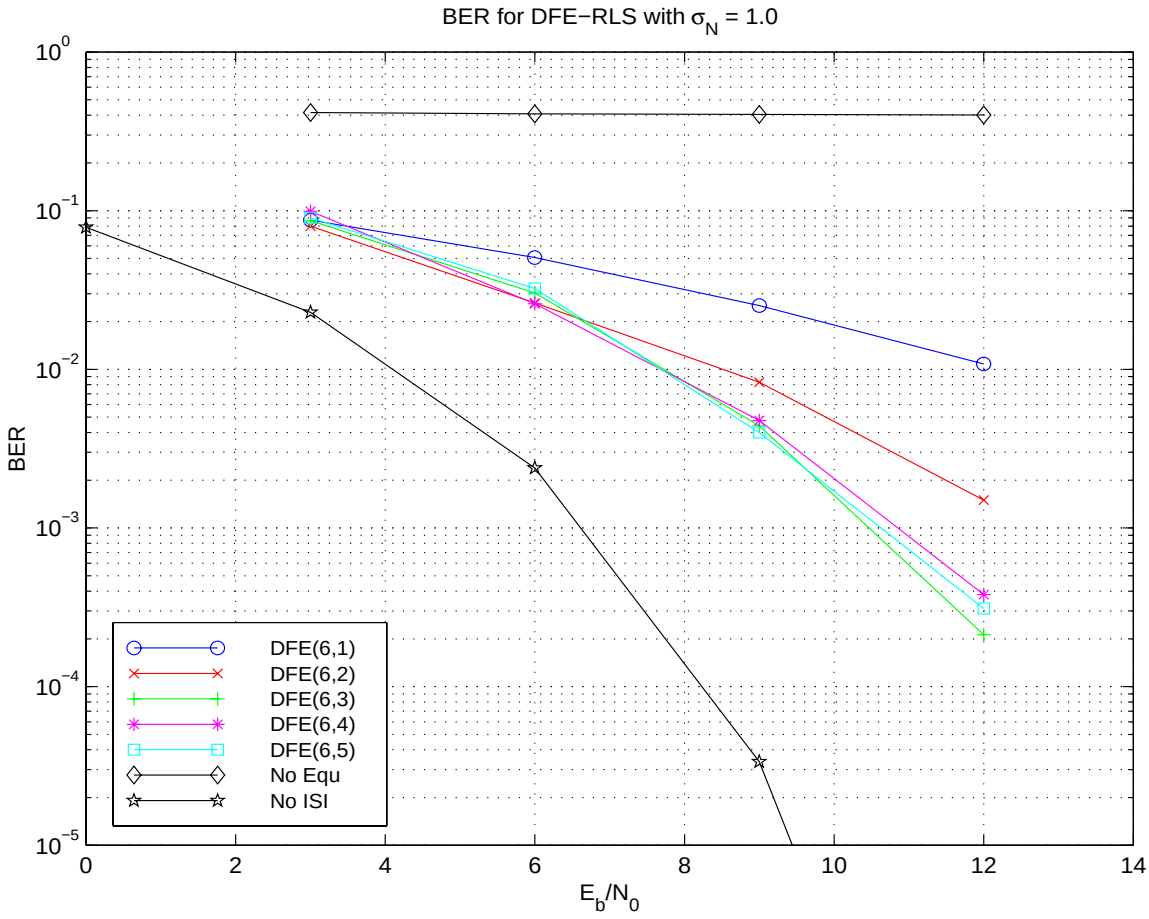

Figure 7.4: BER Performance vs. E_b/N_0 for a DFE with six feedforward taps and different numbers of feedback taps. The channel is stationary and has a normalized RMS delay spread of 1.0

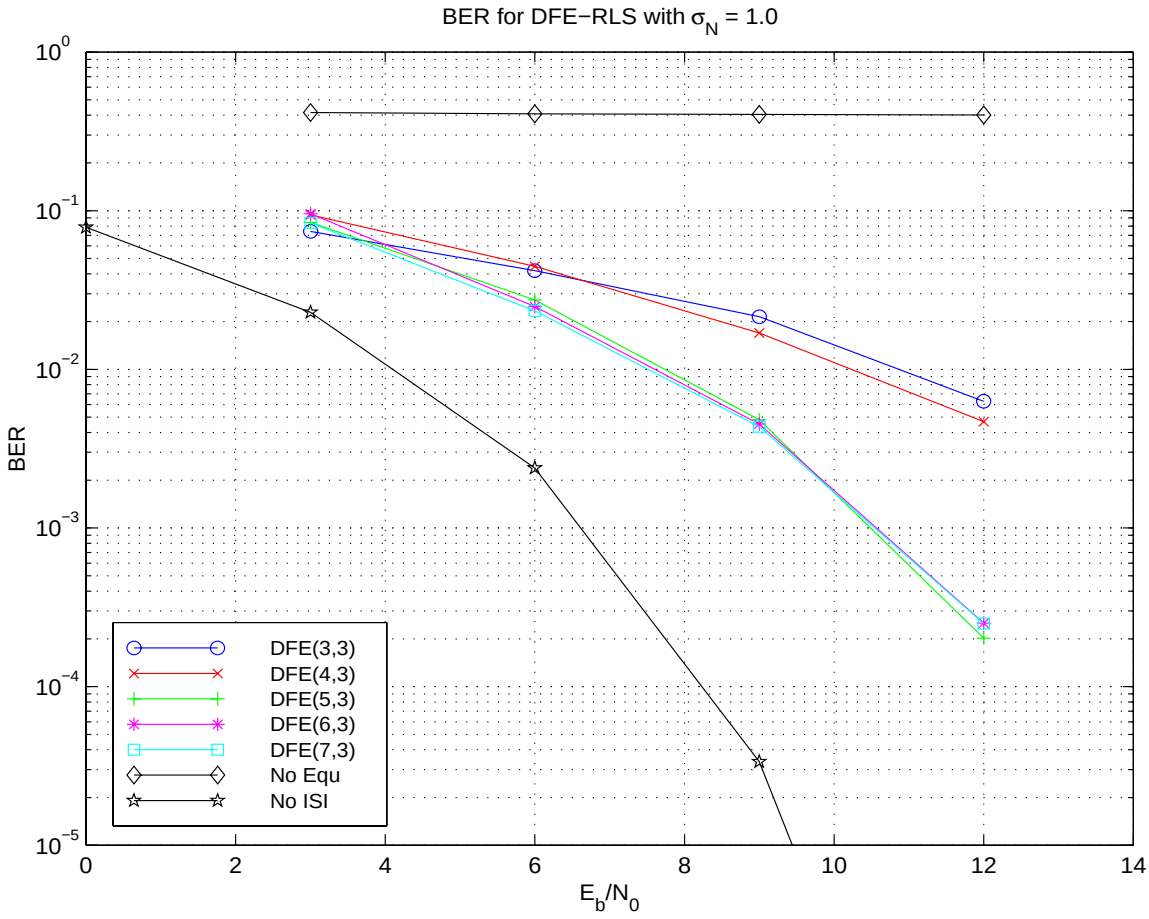

Figure 7.5: BER Performance vs. E_b/N_0 for a DFE with different numbers of feedforward taps and three feedback taps. The channel is stationary and has a normalized RMS delay spread of 1.0

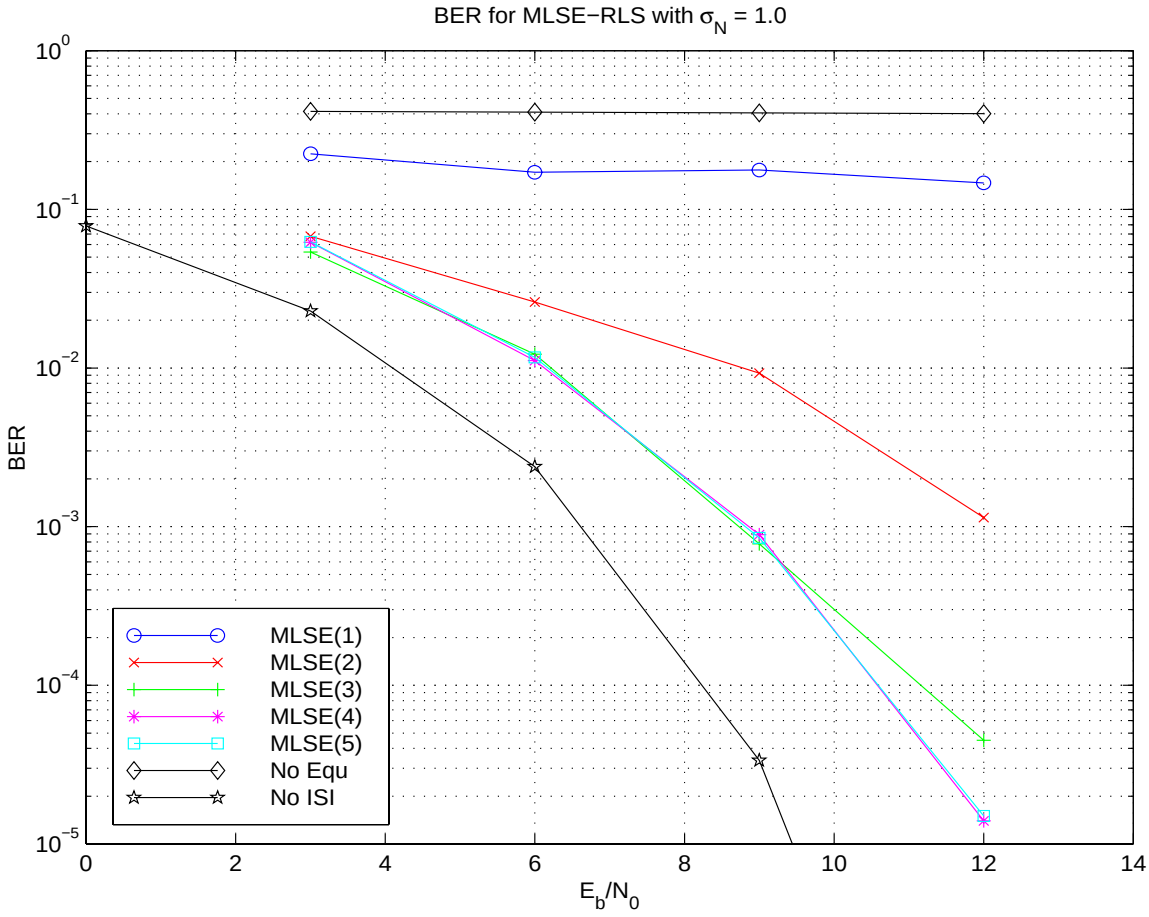

Figure 7.6: BER Performance vs. E_b/N_0 for a MLSE using different values for L. The channel is stationary and has a normalized RMS delay spread of 1.0

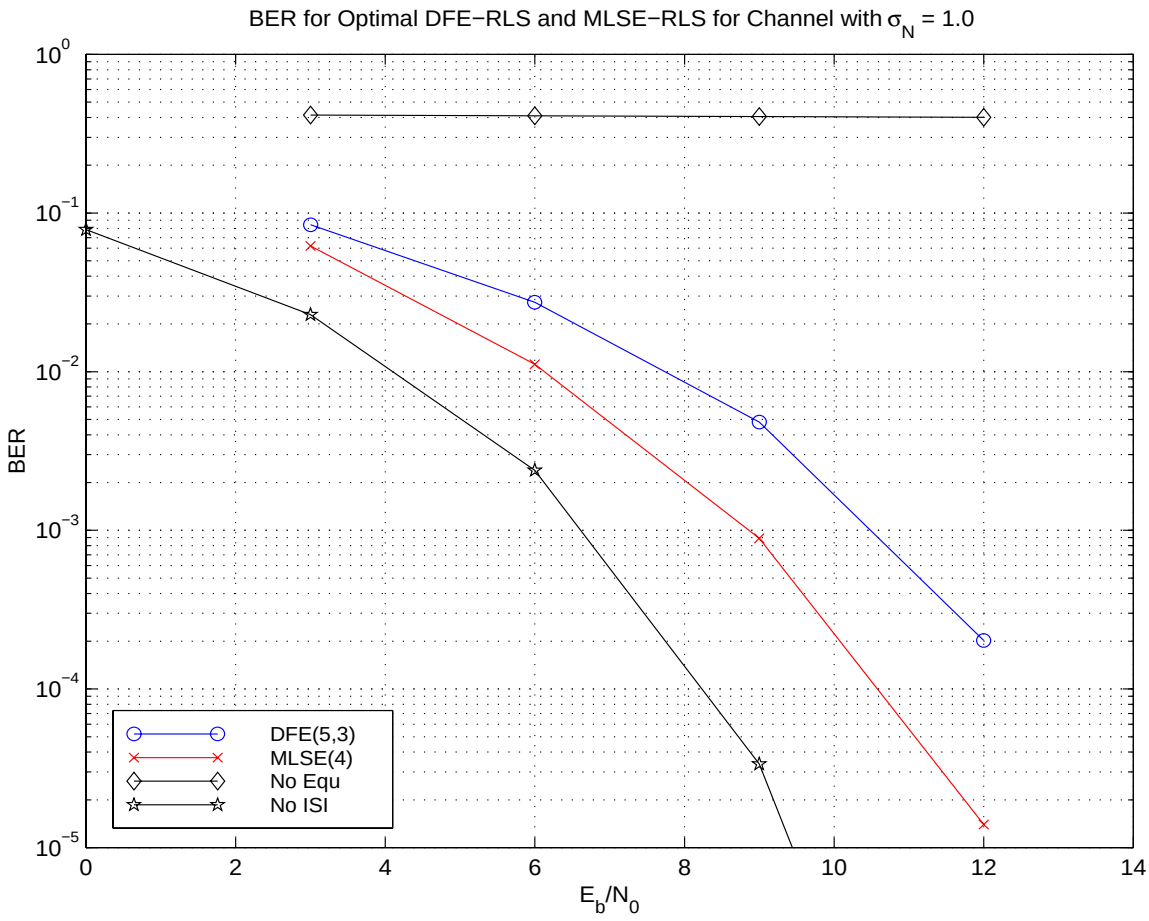

Figure 7.7: Comparison of DFE(5,3) and MLSE(4) for Channel with $\sigma_N = 1.0$

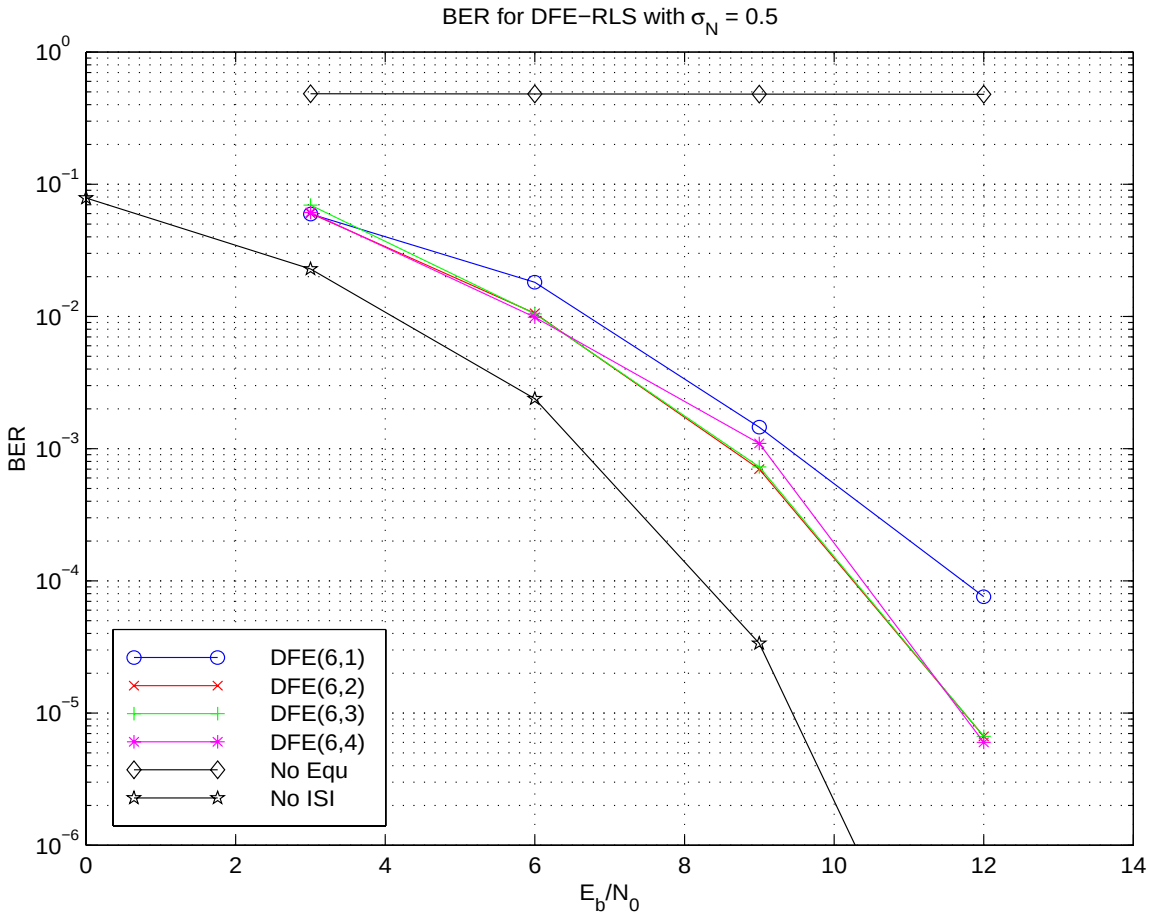

Figure 7.8: BER Performance vs. E_b/N_0 for a DFE with six feedforward taps and different numbers of feedback taps. The channel is stationary and has a normalized RMS delay spread of 0.5

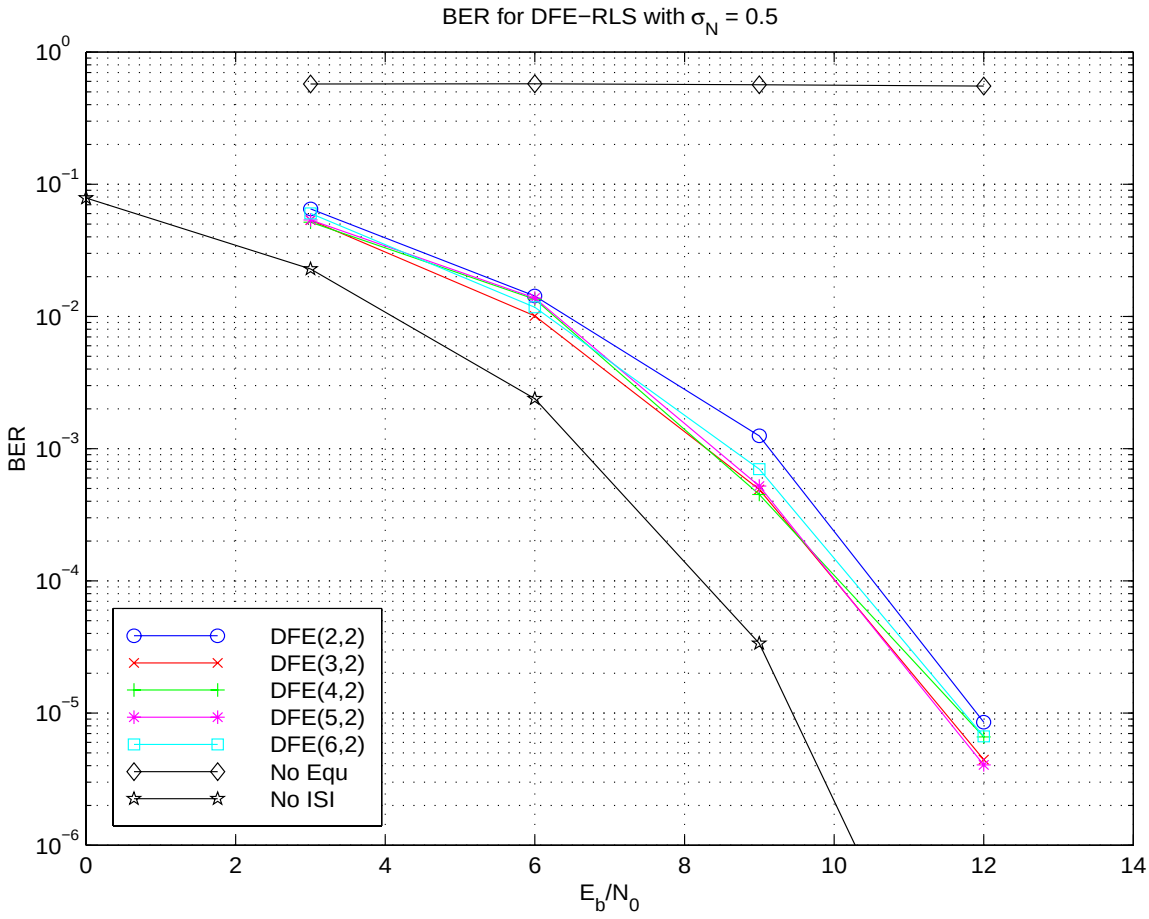

Figure 7.9: BER Performance vs. E_b/N_0 for a DFE with different numbers of feedforward taps and two feedback taps. The channel is stationary and has a normalized RMS delay spread of 0.5

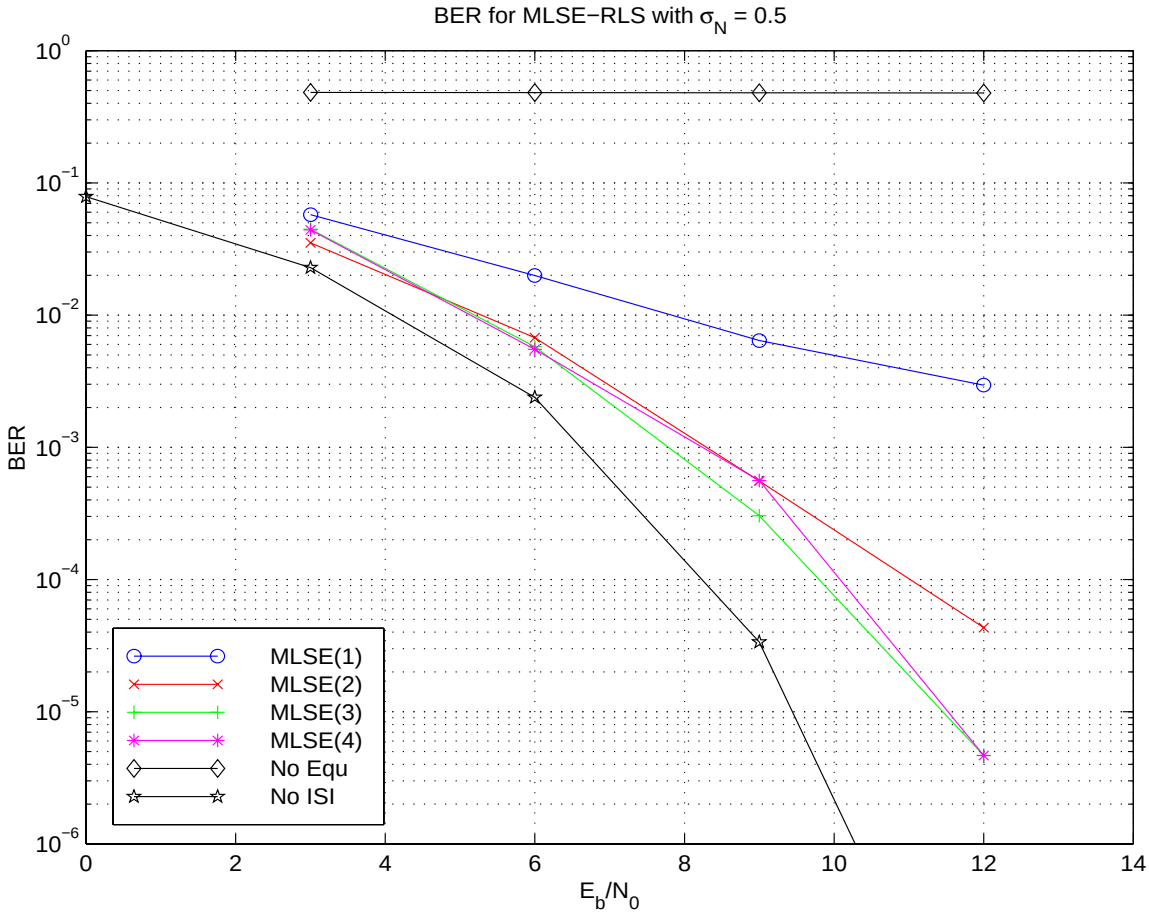

Figure 7.10: BER Performance vs. E_b/N_0 for a MLSE using different values for L. The channel is stationary and has a normalized RMS delay spread of 0.5

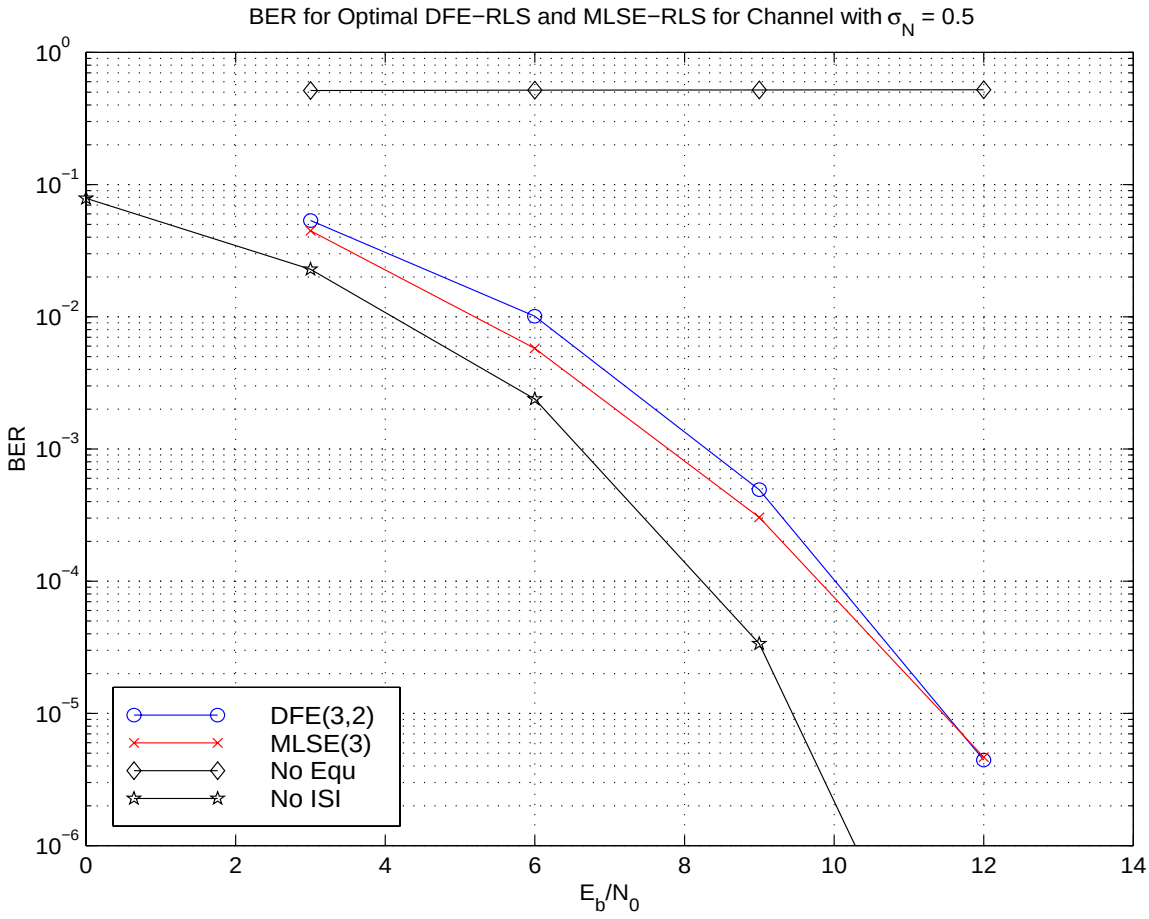

Figure 7.11: Comparison of DFE(3,2) and MLSE(3) for Channel with $\sigma_N = 0.5$

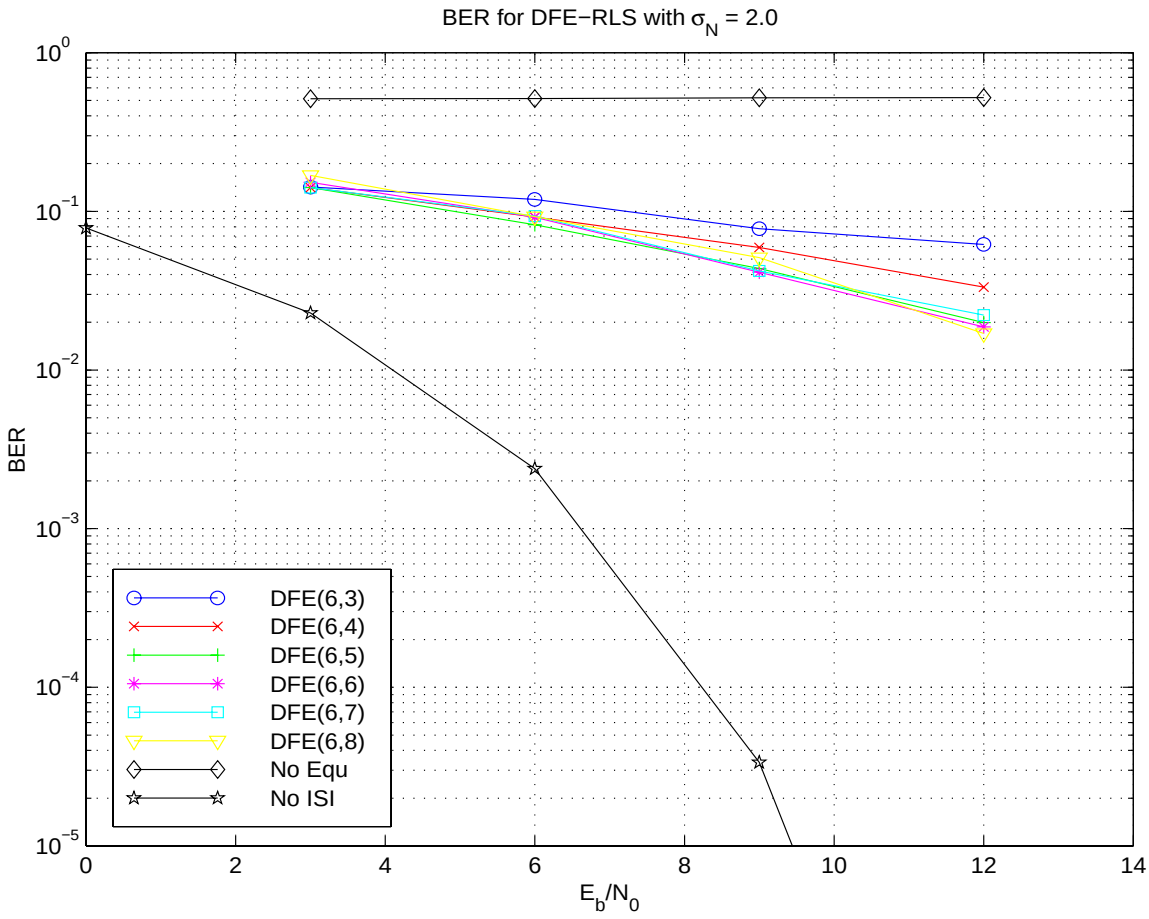

Figure 7.12: BER Performance vs. E_b/N_0 for a DFE with six feedforward taps and different numbers of feedback taps. The channel is stationary and has a normalized RMS delay spread of 2.0

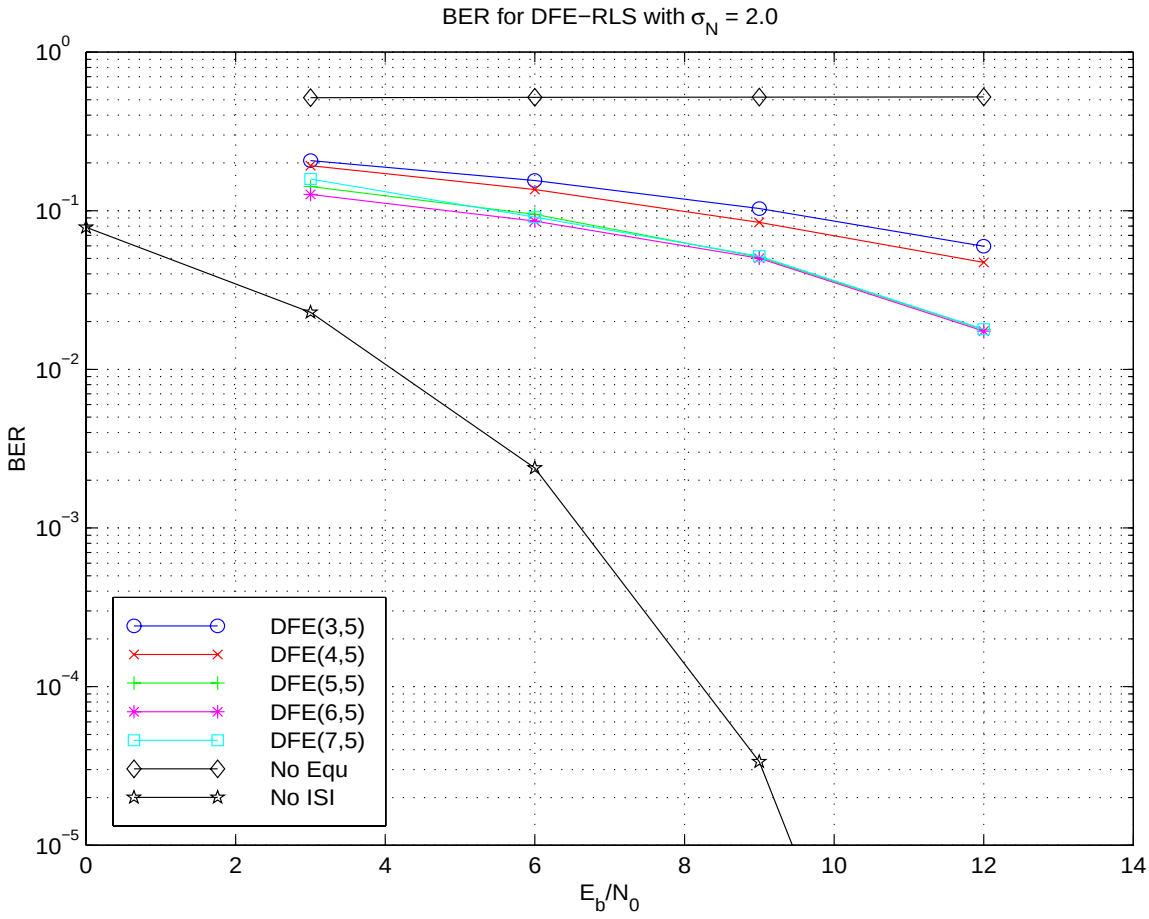

Figure 7.13: BER Performance vs. E_b/N_0 for a DFE with different numbers of feedforward taps and five feedback taps. The channel is stationary and has a normalized RMS delay spread of 2.0

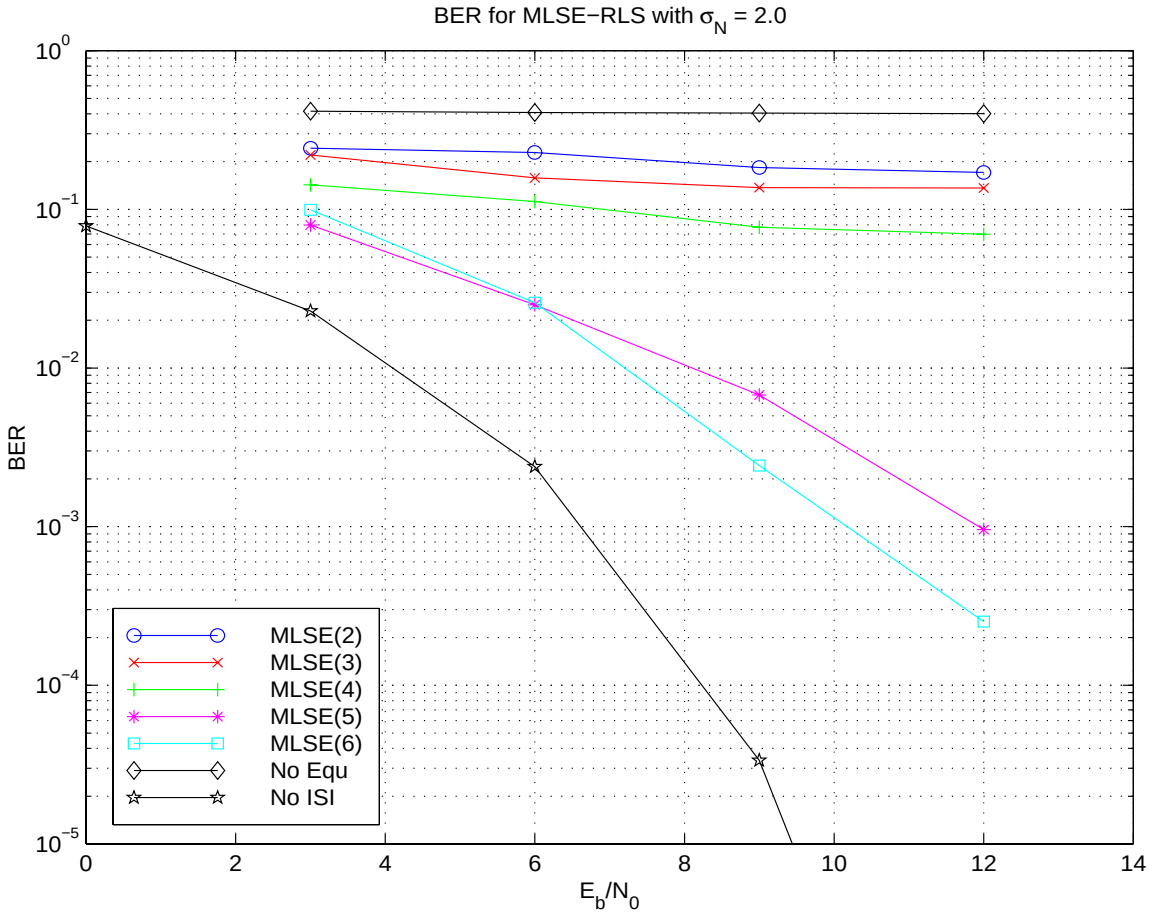

Figure 7.14: BER Performance vs. E_b/N_0 for a MLSE using different values for L. The channel is stationary and has a normalized RMS delay spread of 0.5

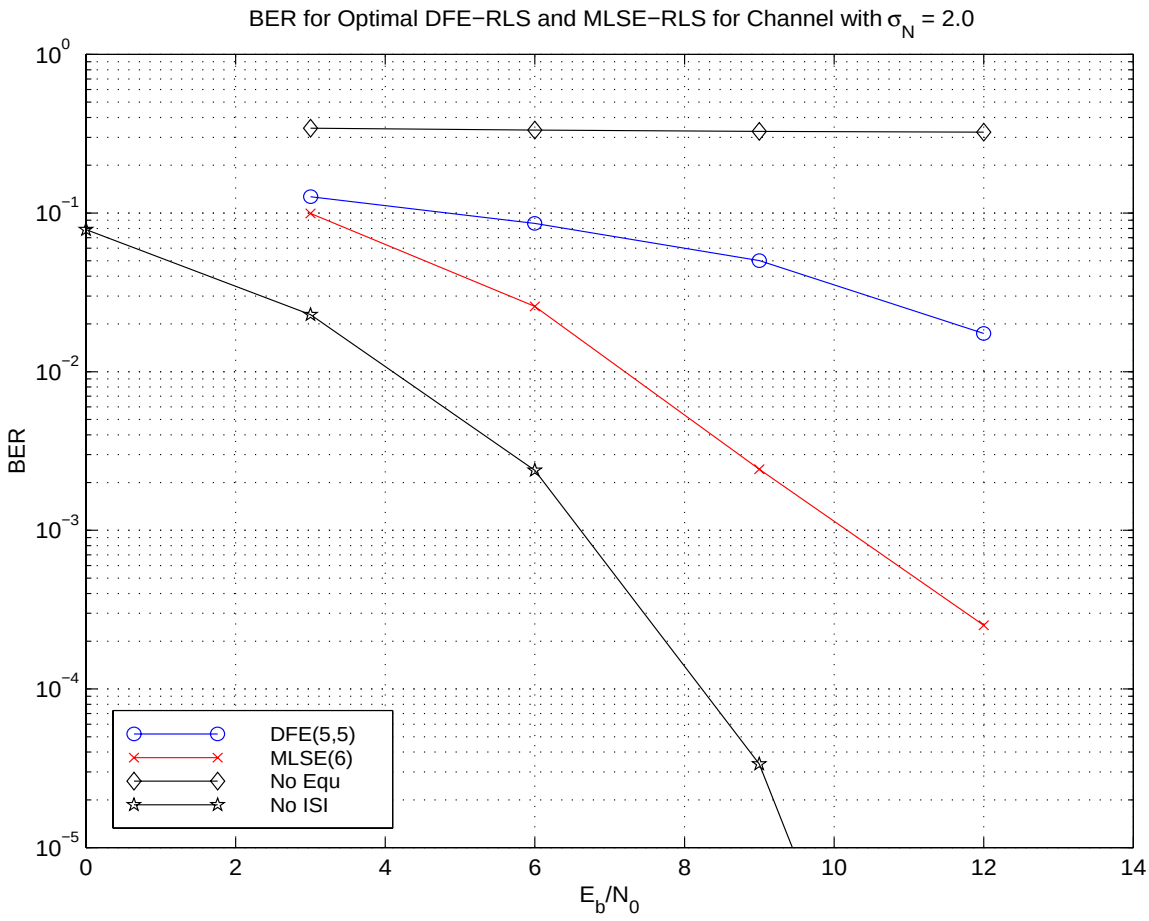

Figure 7.15: Comparison of DFE(5,5) and MLSE(6) for Channel with $\sigma_N = 2.0$

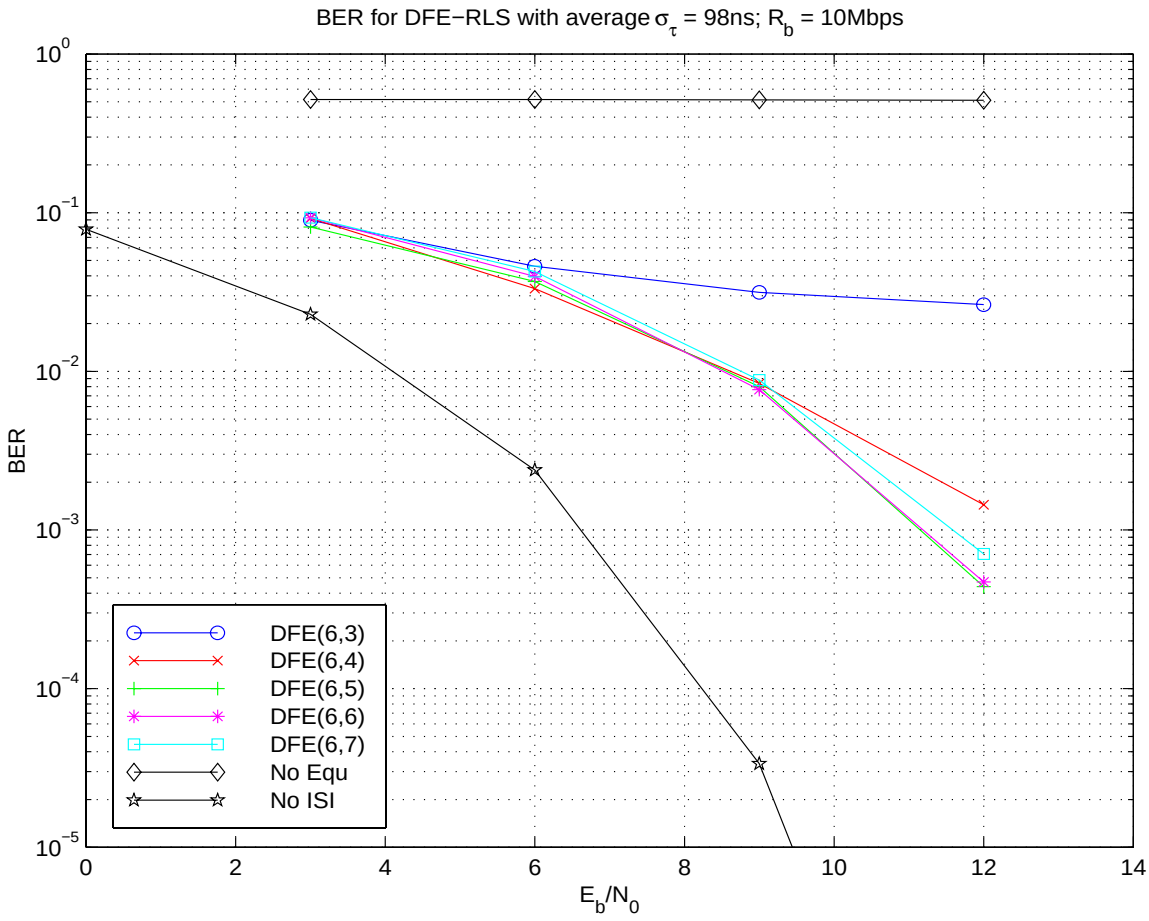

Figure 7.16: BER Performance vs. E_b/N_0 for a DFE with six feedforward taps and different numbers of feedback taps. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 98$ns were used. $R_b = 10$Mbps

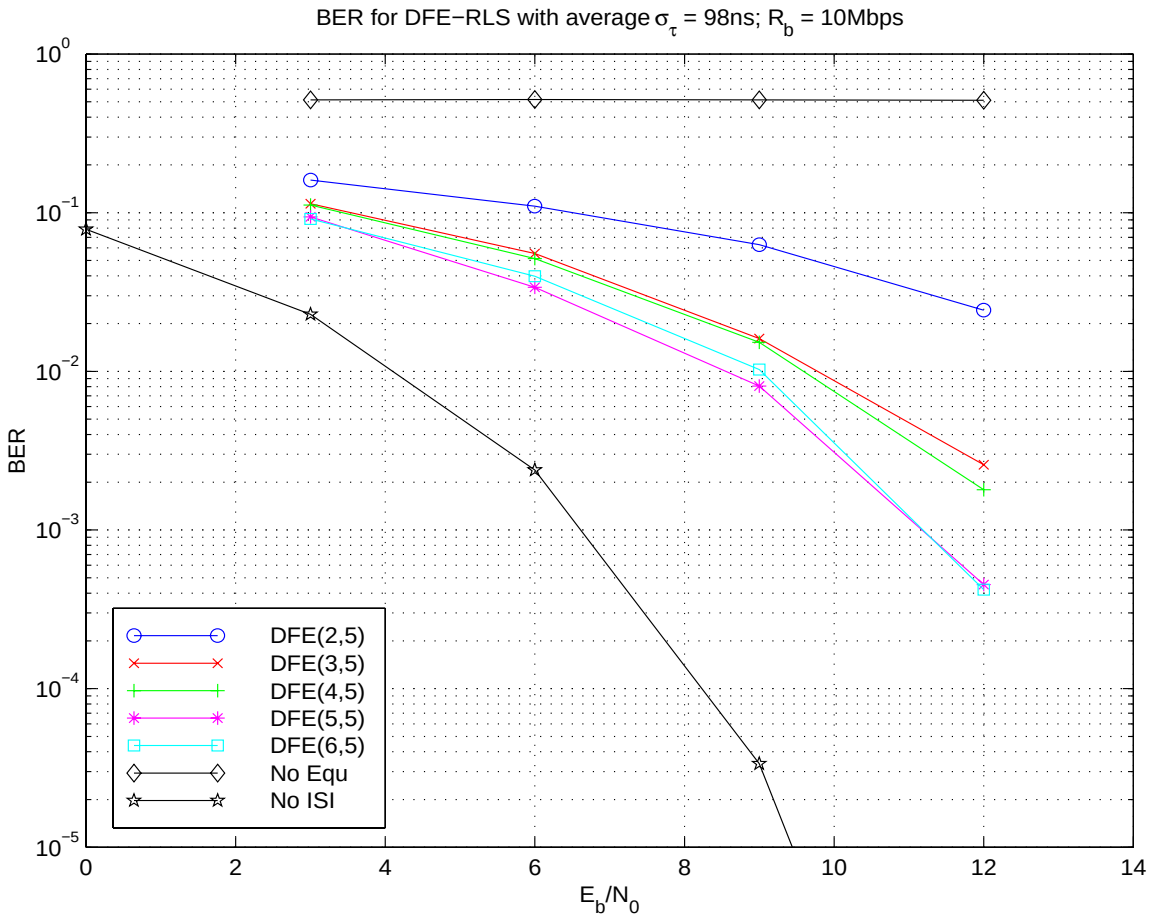

Figure 7.17: BER Performance vs. E_b/N_0 for a DFE with different numbers of feedforward taps and five feedback taps. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 98$ns were used. $R_b = 10$Mbps

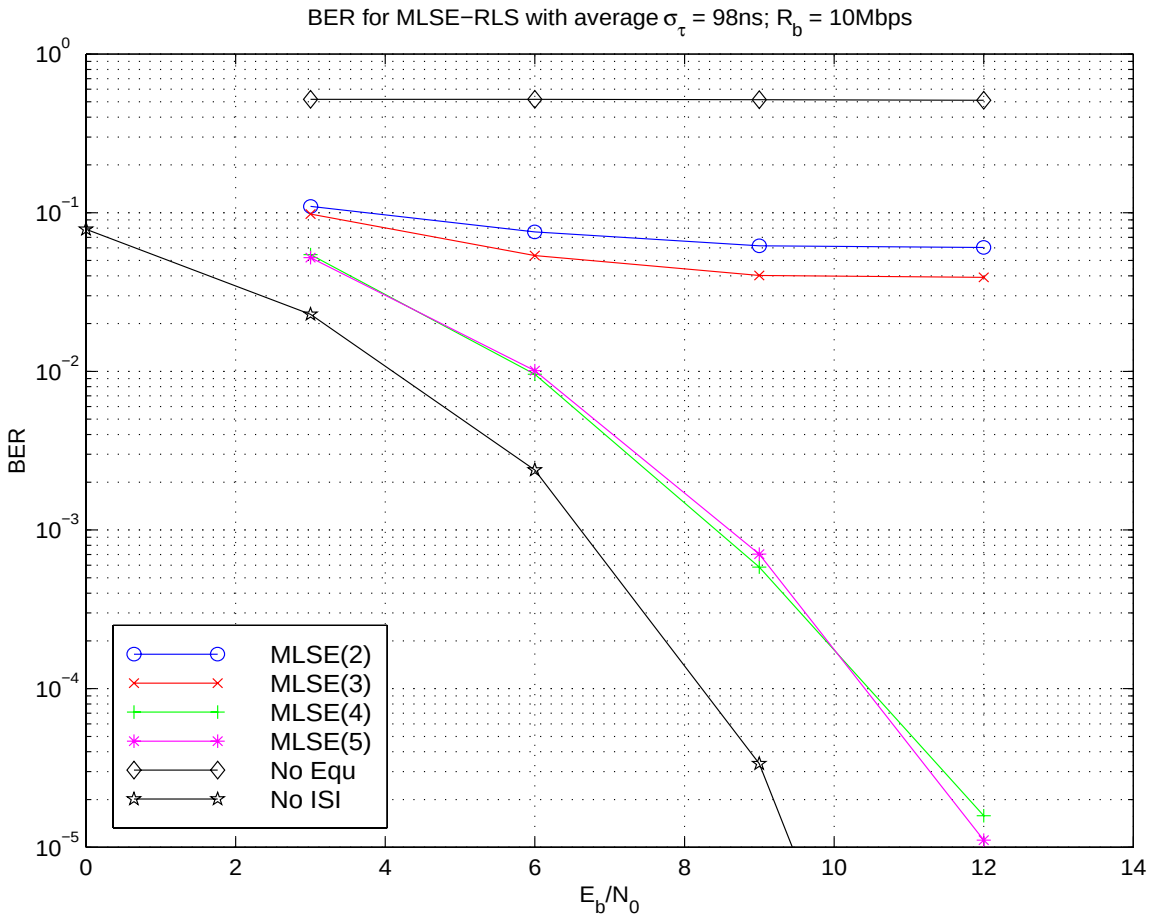

Figure 7.18: BER Performance vs. E_b/N_0 for an MLSE using different values of L. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 98$ns were used. $R_b = 10$Mbps

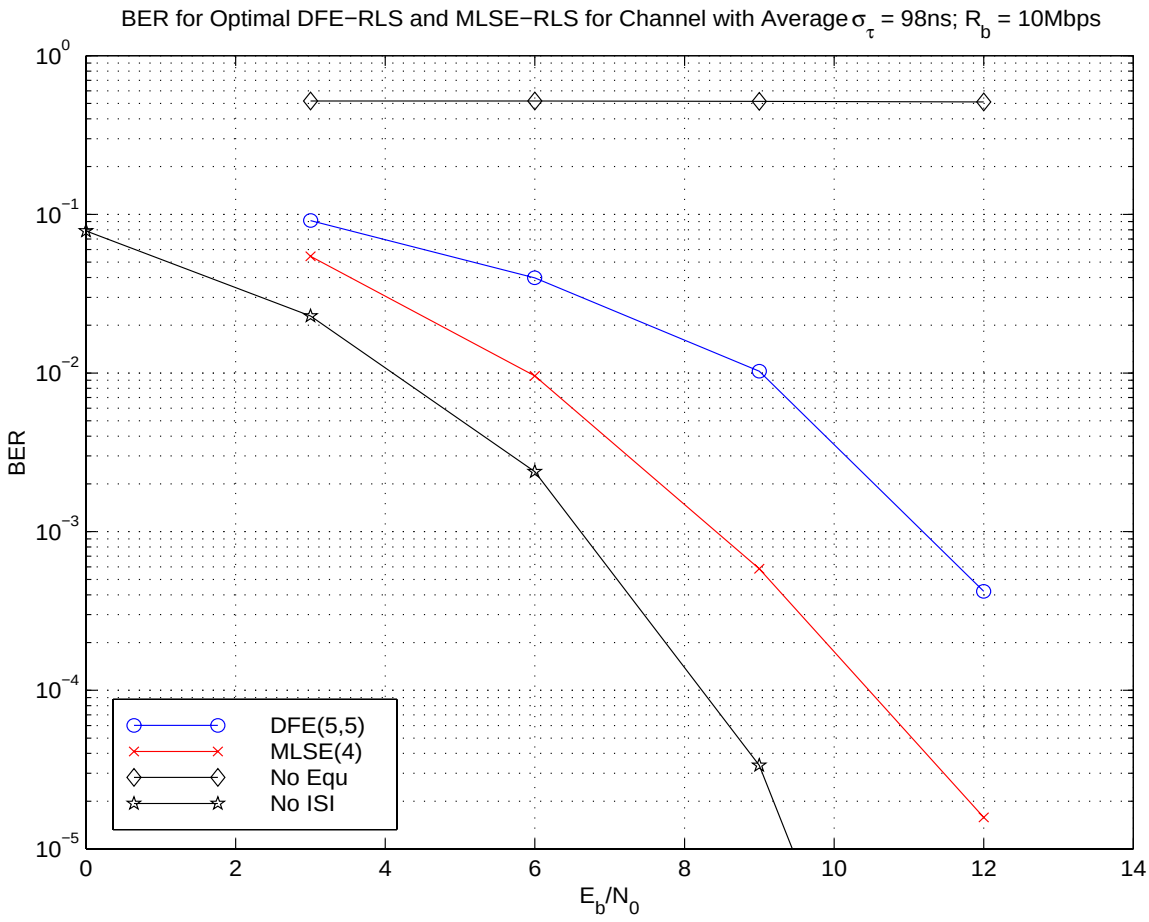

Figure 7.19: Comparison of DFE(5,5) and MLSE(4) for Mobile Radio Indoor Channel with $\sigma_\tau = 98$ ns; $R_b = 10$ Mbps

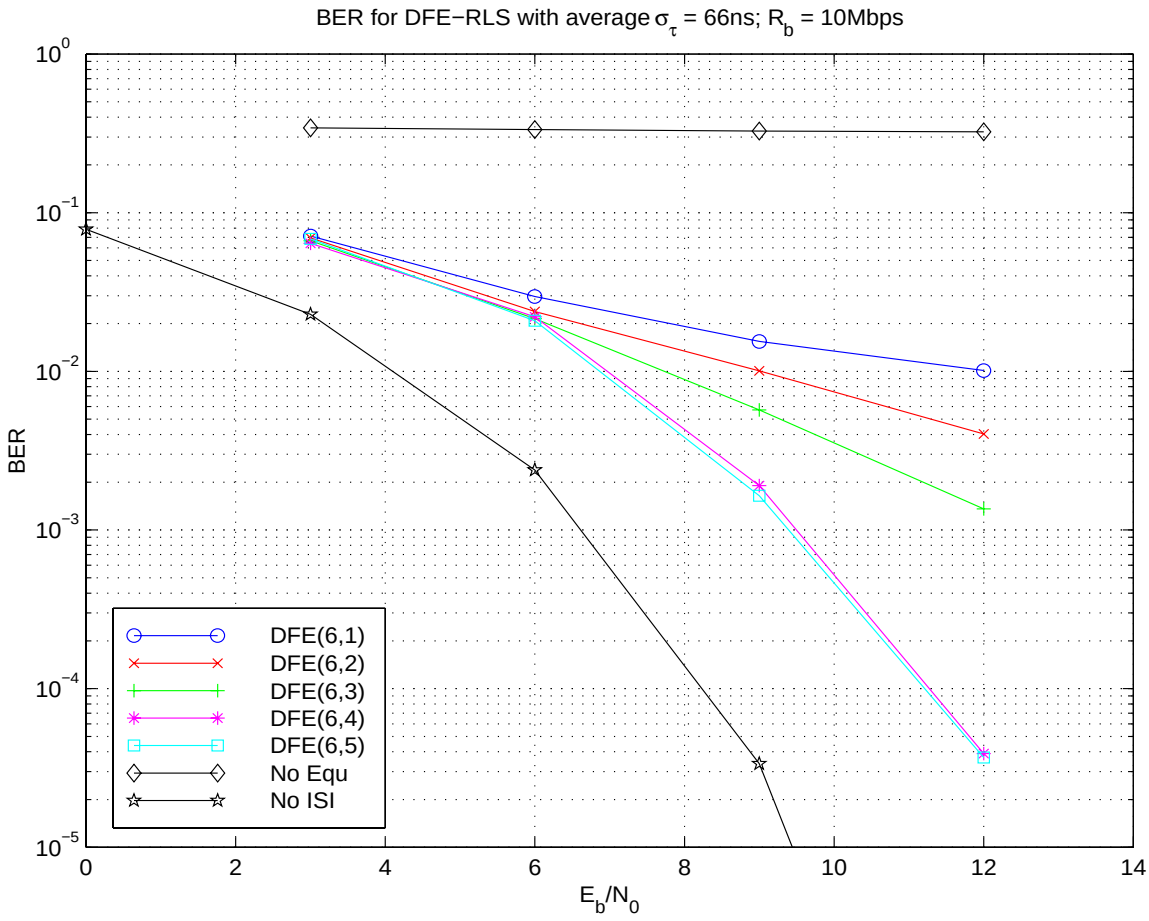

Figure 7.20: BER Performance vs. E_b/N_0 for a DFE with six feedforward taps and different numbers of feedback taps. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 66$ns were used. $R_b = 10$Mbps

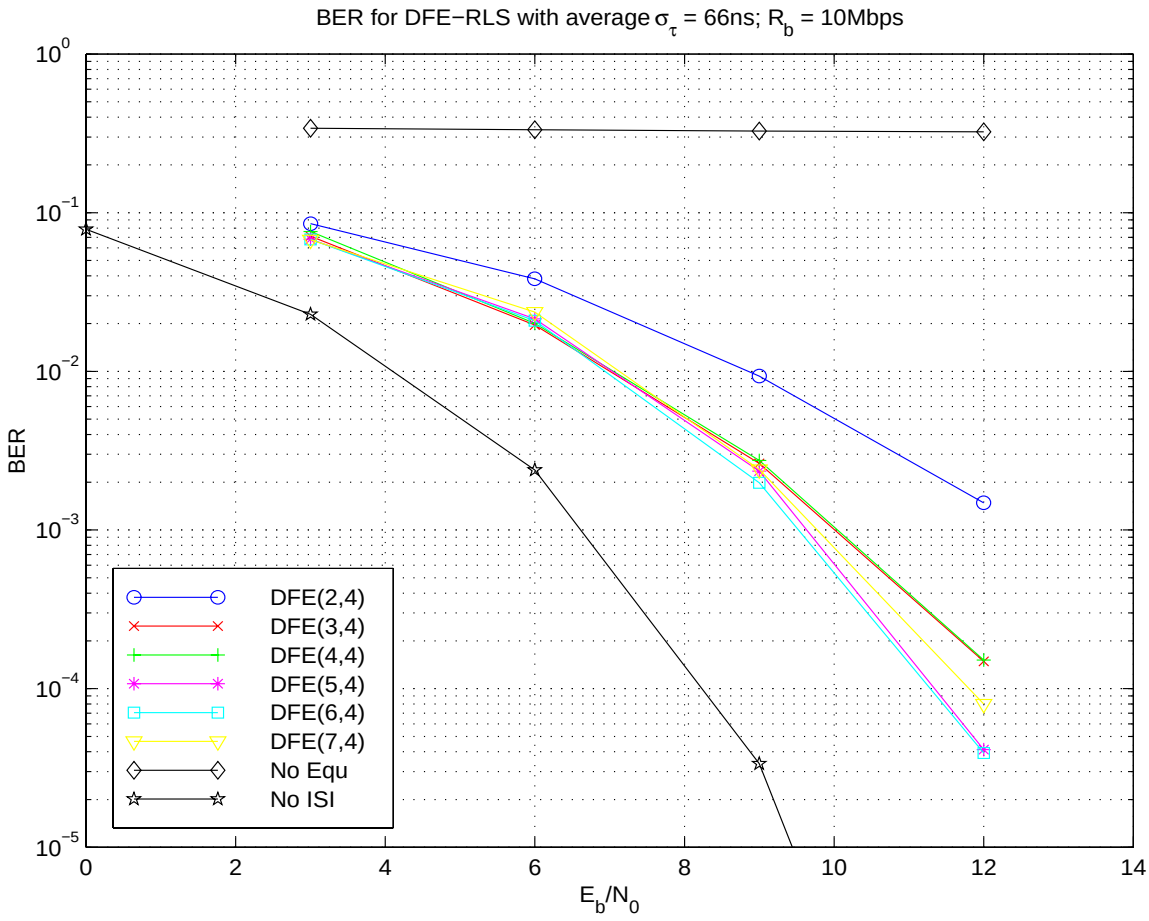

Figure 7.21: BER Performance vs. E_b/N_0 for a DFE with different numbers of feedforward taps and five feedback taps. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 66$ns were used. $R_b = 10$Mbps

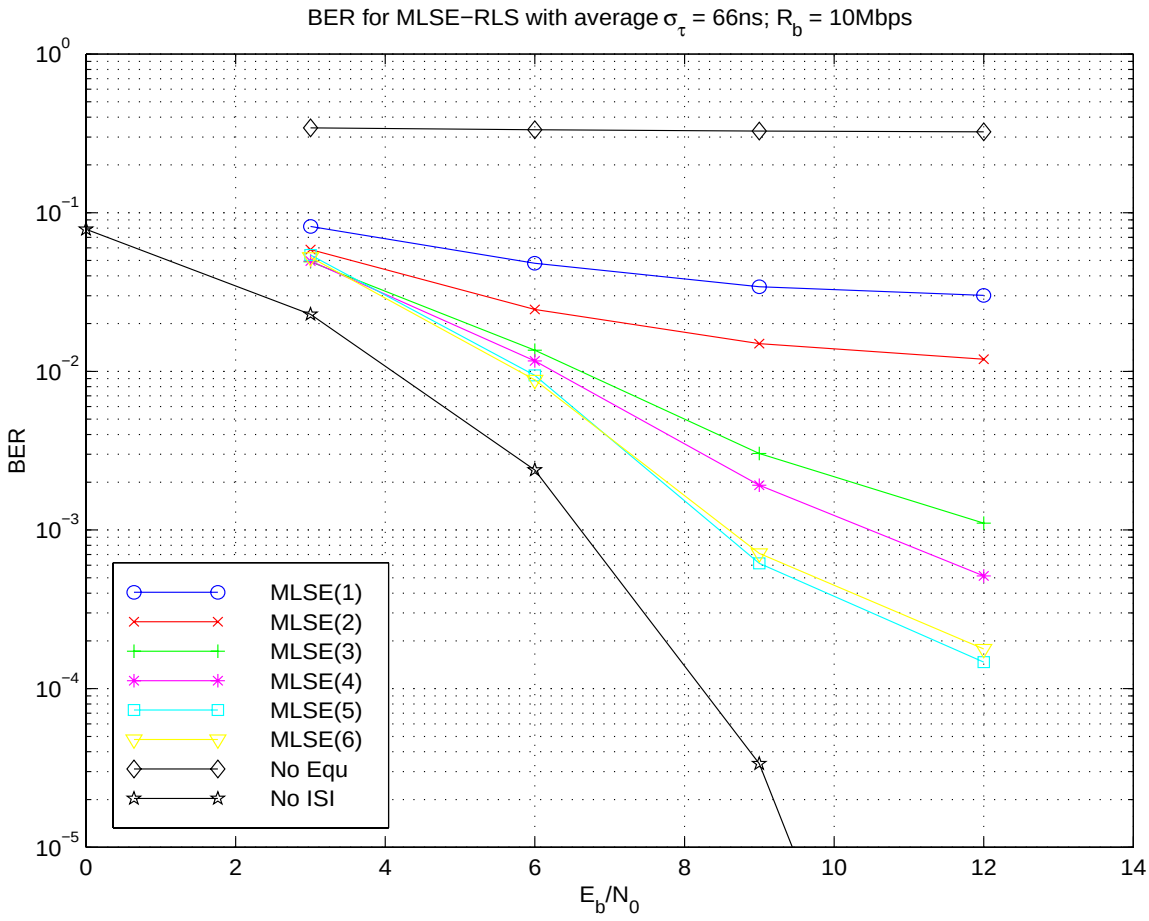

Figure 7.22: BER Performance vs. E_b/N_0 for an MLSE using different values of L. Nineteen uniformly spaced channel profiles along a 4.5λ track with an average $\sigma_\tau = 66$ns were used. $R_b = 10$Mbps

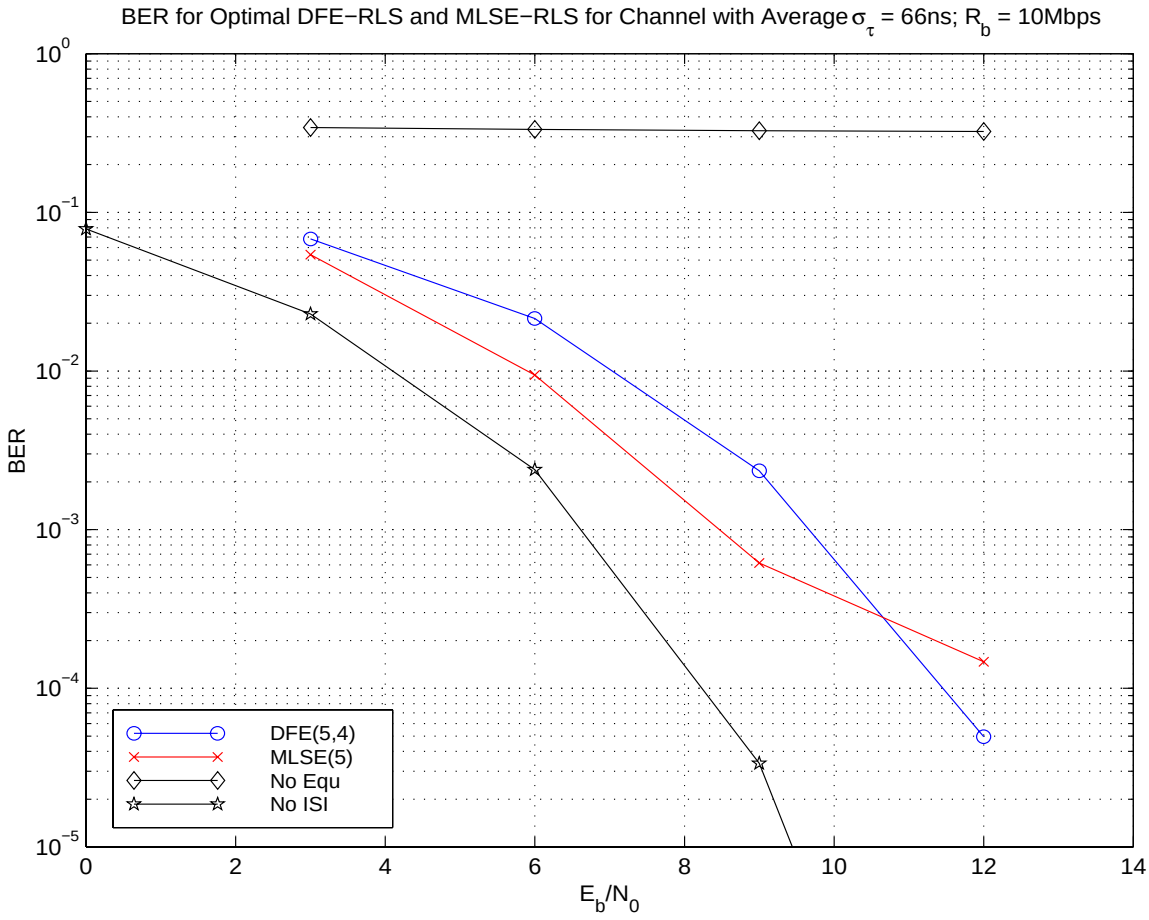

Figure 7.23: Comparison of DFE(5,4) and MLSE(5) for Mobile Radio Indoor Channel with $\sigma_\tau = 66$ ns; $R_b = 10$ Mbps

Computational Complexity

The results described previously clearly indicate that an MLSE equalizer provides better BER performance in almost all cases. However, this performance is gained at the cost of much more computational complexity. The MLSE equalizer requires many more computations per symbol than does the DFE. The DFE works as two FIR filters, which are fairly simple to implement in a real system. The MLSE, on the other hand, must calculate 2^{L+1} metrics during each symbol and relies on the Viterbi Algorithm which requires lots of memory and sorting routines.

Figure 7.24 is a graph comparing the computational complexity of a DFE(5,5) with different MLSE equalizer sizes. The computational complexity was found by determining the number of compares, adds, and multiplies required for each equalizer structure per symbol. The DFE requires one compare, $N_1 + N_3$ adds, and $N_1 + N_3 + 1$ multiplies per symbol. On the other hand, the MLSE requires M^L compares, M^{L+2} adds, and M^{L+1} multiplies, where M is the size of the symbol alphabet. In this case $M = 2$.

Even the simplest MLSE equalizer, the one with $L = 3$, requires almost three times as many operations than the most complicated DFE, the DFE(5,5), used in the simulations. For a system operating at 10 Mbps, most of the MLSE equalizers will require computational rates on the order of 10^9 operations per second. This requirement is much too high for most DSP chips available today. For this reason MLSE equalization can only be used with slower systems and not high-speed indoor data networks. However, with the leaps and bounds being made in DSP chip technology, processors that can perform at this speed may soon be available. When they do, MLSE equalization may be a good option for compensating for ISI in a high-speed system operating in an indoor environment.

7.5 Conclusions

The results of the simulations clearly show that MLSE equalization provides better BER performance than a decision-feedback equalizer for almost all indoor channel conditions.

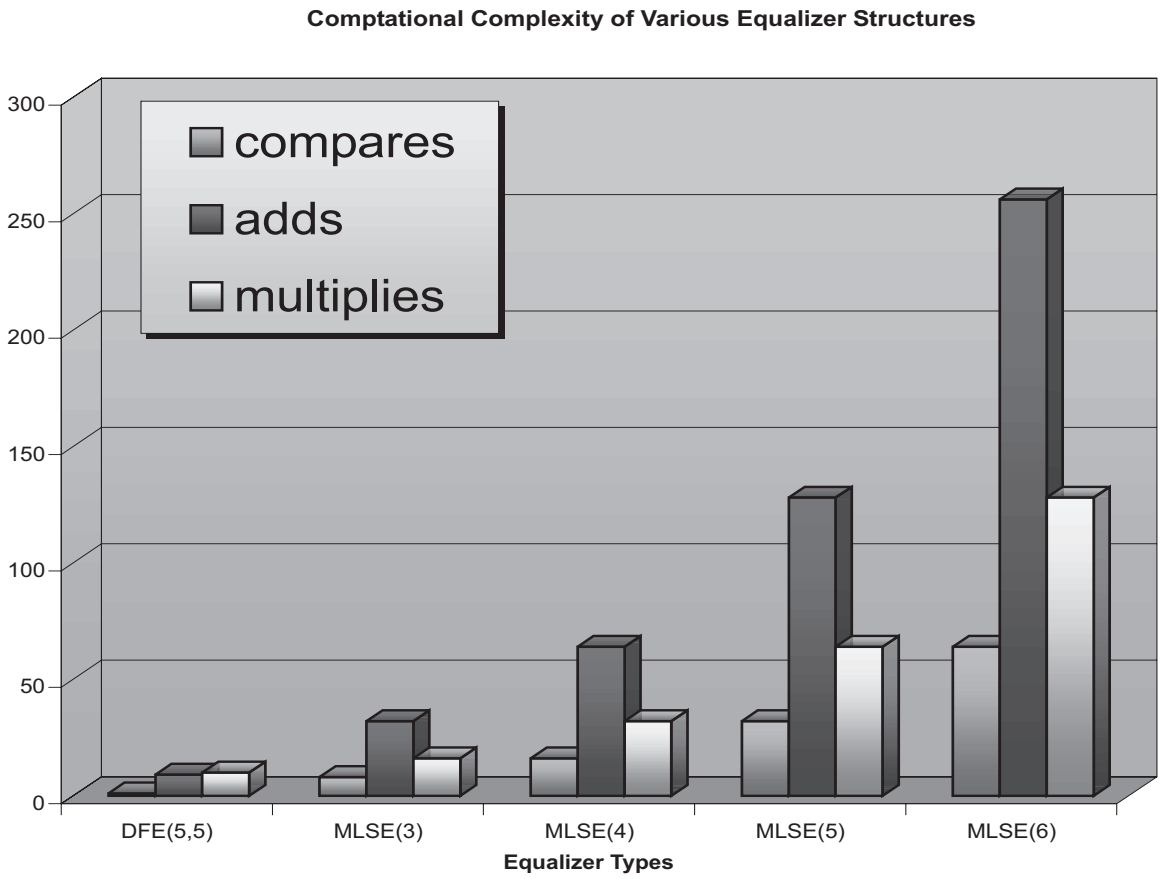

Figure 7.24: Computational Complexity Required by the Different Equalizer Structures

This includes both stationary and mobile radio channels. For channels with longer delay spreads, the more complex the equalizer needs to be. In the case of the DFE, more taps in the feedforward and feedback filters are needed. In the MLSE equalizer more state metrics need to be computed. This extra complexity, especially in the case of the MLSE equalizer, may cause the equalizer to become unrealizable in a real high-speed indoor communication system. Until chips become faster and more streamlined, decision-feedback equalization is the most pragmatic equalization scheme for high-speed ($>$ 1 Mbps) communications in an indoor environment.

Chapter 8

Conclusion and Future Work

The use of equalization techniques to compensate for intersymbol interference caused by a wireless multipath channel has been discussed in depth. Also discussed was the relationship between adaptive equalization and adaptive arrays. Many different simulation programs were developed to help readers further understand the many different equalizer structures and algorithms along with the use of adaptive equalization in both mobile and indoor environments. Simulation programs that model adaptive array systems were also included. Most of these simulations were explained in detail to allow readers to learn how to simulate wireless systems that use adaptive equalizers and adaptive arrays.

Several different equalizer structures were studied. These included the linear transversal equalizer, the fractionally spaced equalizer, the decision-feedback equalizer, and the MLSE equalizer. Both the DFE and MLSE equalizers provided the best performance of all the different structures. The MLSE equalizer provided better performance than the DFE, but at the expense of much more computational complexity.

Both the least mean square algorithm and the recursive least square algorithm were discussed in this work. It was found that the LMS algorithm required many iterations to converge, but required only a few computations per iteration. The RLS algorithm converged much more quickly, but required many more calculations per iteration.

To model a mobile radio channel, a two-ray Rayleigh fading channel model was used. It

was found that in order to track this rapidly changing channel, many training sequences needed to be included with the transmitted data, or the equalizer would need to adapt using the decisions made by the receiver. This decision-directed mode allowed for a higher data throughput, but required many extra computations.

Adaptive arrays operate much like linear, symbol spaced equalizers, and can use many of the same adaptive algorithms as an equalizer. Adaptive arrays operate by multiplying the output of each element in the array with a complex weight. These weights are adjusted such that interfering signals are severely attenuated while desired signals are received clearly. Adaptive algorithms can be used to adjust the weights without knowledge of different signals' angle of arrival and signal strengths.

In an indoor environment, the delay spread of the channel is relatively small. Even though this is true, an adaptive equalizer is still needed to transmit data at high rates ($>$ 1 Mbps). Both DFE and MLSE equalizers do a good job in mitigating the ISI caused by the indoor channel. MLSE equalizers provide better performance than the DFE equalizers, but require many more computations per symbol than the DFE. For high-speed wireless data networks, this high computational complexity limits the speed at which data can be sent. For this reason, a decision-feedback equalizer is a better choice when using an equalizer in a high-speed indoor system.

Future Work

Equalization for indoor wireless systems is an idea that has only been around for the last few years. There are still many problems that need to be solved. For one, efforts should be made to make MLSE a more realizable equalization technique for high-speed indoor systems. This could involve simplifying the metric calculations as well as making the Viterbi algorithm as efficient as possible.

The DFE could also be examined to see if its performance could be improved. For example, research could be done to see if changing the feedforward filter to fractionally spaced from symbol spaced would improve the performance of the equalizer.

Another topic that deserves more study is the comparison of DFE and MLSE equalization in mobile indoor channels that exhibit deep fades. In this thesis, only one such channel was simulated, so not much was learned about this phenomenon. Researchers should simulate both DFE and MLSE equalizer operation with many other sets of mobile indoor channel profiles that also contain deep fades and compare the performance of the different equalizer types.

Finally, the performance of different equalization techniques with different modulation types could be looked at. The HIPERLAN standard developed in Europe uses GMSK. Some researchers have looked at using decision-feedback with the standard [12], but few have examined using MLSE equalizers.

Bibliography

[1] S. U. H. Qureshi, "Adaptive equalization," *Proc. IEEE*, 1985.

[2] J. G. Proakis, *Digital Communications*. New York: McGraw-Hill, 3rd ed., 1995.

[3] T. S. Rappaport, *Wireless Communications*. Prentice Hall, 1996.

[4] S. Haykin, *Adaptive Filter Theory*. Prentice Hall, 1986.

[5] S. Haykin, *Communication Systems*. John Wiley and Sons, 1994.

[6] L. W. Couch, *Digital and Analog Communication Systems*. Macmillan, 1997.

[7] R. E. Ziemer and R. L. Peterson, *Digital Communications and Spread Spectrum systems*. Macmillan, 1985.

[8] T. S. Rappaport, "Characterization of UHF multipath radio channels in factory buildings," *IEEE Trans. Antennas & Propag.*, vol. 37, pp. 1058–1069, Aug. 1989.

[9] T. A. Sexton and K. Pahlavan, "Channel modeling and adaptive equalization of indoor radio channels," *IEEE J. Select. Areas Commun.*, vol. 7, pp. 114–121, Jan. 1989.

[10] D. C. Cox and R. P. Leck, "Distributions of multipath delay spread and average excess delay for 910 MHz urban mobile radio paths," *IEEE Transactions on Antennas and Propagation*, vol. AP-23, pp. 206–213, March 1975.

[11] R. Ganesh and K. Pahlavan, "Effects of traffic and local movements on multipath characteristics of an indoor radio channel," *Electronics Letters*, vol. 26, pp. 810–2, June 1990.

[12] J. Tellado, E. Khayata, and J. M. Cioffi, "Equalizing GMSK for high data rate wireless LANs," in *Proc., IEEE PIMRC*, vol. 1, pp. 16–20, 1995.

[13] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*. Plenum Press, 1992.

[14] W. H. Tranter, *Basic Simulation Models of Phase Tracking Devices Using MATLAB*. Prentice Hall. to be published.

[15] W. H. Tranter, "Simulation of communication systems." Class Notes, Fall 1997. Virginia Tech, EE5984.

[16] J. G. Proakis, *Digital Communications*. New York: McGraw-Hill, 2nd ed., 1989.

[17] R. W. Lucky, "Automatic equalization for digital communications," *Bell Systems Technical Journal*, vol. 44, pp. 547–588, April 1965.

[18] B. Widrow, "Adaptive filters, I: Fundamentals," Tech. Rep. 6764-6, Stanford University, Stanford, CA, December 1966.

[19] R. W. Lucky, "Signal filtering with the transversal equalizer," in *7th Ann. Allerton Conf. on Circuits and Systems Theory*, pp. 792–803, Oct. 1969.

[20] G. Ungerboeck, "Fractional tap-spacing equalizer and consequences for clock recovery in data modems," *IEEE Trans. Commun.*, vol. COM-24, pp. 856–64, August 1976.

[21] M. E. Austin, "Decision-feedback equalization for digital communication over dispersive channels," Tech. Rep. 437, MIT Lincoln Laboratory, Lexington, Mass., August 1967.

[22] C. Belfiore and J. H. Park Jr., "Decision-feedback equalization," *Proc. IEEE*, vol. 67, pp. 1143–1156, Aug. 1979.

[23] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[24] G. D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363–378, May 1972.

[25] G. Ungerboeck, "Adaptive maximum-likelihood receiver for carrier-modulated data-transmission systems," *IEEE Trans. Commun.*, vol. COM-22, pp. 624–636, May 1974.

[26] G. E. Bottomley and S. Chennakeshu, "Unification of MLSE receivers," in *Proc., Virginia Tech Symp. on Wireless Personal Commun.*, 1997.

[27] G. E. Bottomley and S. Chennakeshu, "Adaptive MLSE equalization forms for wireless communications," in *Proc., Virginia Tech Symp. on Wireless Personal Commun.*, 1995.

[28] R. W. Lucky, "Techniques for adaptive equalization of digital communications," *Bell Systems Technical Journal*, vol. 45, pp. 255–286, 1965.

[29] D. N. Godard, "Channel equalization using a Kalman filter for fast data transmission," *IBM J. Res. Development*, vol. 18, pp. 267–273, May 1974.

[30] B. Picinbono, *Adaptive Signal Processing for Detection and Communication*. Alphen aan den Rijn, The Netherlands: Sijthoff and Nordoff, 1978.

[31] E. Dahlman, "New adaptive Viterbi detector for fast-fading mobile-radio channels," *Electronics Letters*, vol. 26, pp. 1572–1573, July 1990.

[32] B. Sklar, "Rayleigh fading channels in mobile digital communication systems part I: Characterization," *IEEE Commun. Magazine*, pp. 90–109, July 1997.

[33] R. H. Clarke, "A statistical theory of mobile-radio reception," *Bell Systems Technical Journal*, vol. 47, pp. 957–1000, 1968.

[34] M. J. Gans, "A power spectral theory of propagation in the mobile radio environment," *IEEE Trans. Veh. Tech.*, vol. VT-21, pp. 27–38, February 1972.

[35] J. I. Smith, "A computer generated multipath fading simulation for mobile radio," *IEEE Trans. Veh. Tech.*, vol. VT-24, pp. 39–40, Aug. 1975.

[36] W. C. Jakes, *Microwave Mobile Communications*. John Wiley and Sons, 1974.

[37] H. Hashemi, "Simulation of the urban radio propagation channel," *IEEE Trans. Veh. Tech.*, vol. VT-28, Aug. 1979.

[38] G. L. Turin, "A statistical model of urban multipath propagation," *IEEE Trans. Veh. Tech.*, vol. VT-21, pp. 1–9, Feb. 1972.

[39] W. Huang and T. S. Rappaport, "Simulation of adaptive equalization in two-ray, SIRCIM, and SMRCIM mobile radio channels," Master's thesis, Virigina Polytechnic Institute and State University, Jan. 1992.

[40] T. S. Rappaport, W. Huang, and M. J. Feuerstein, "Performance of decision feedback equalizers in simulated urban and indoor radio channels," *IECE Trans. Commun.*, vol. E76-B, pp. 78–89, Feb. 1993.

[41] J. C. Liberti and T. S. Rappaport, *Analysis of CDMA Cellular Radio Systems Employing Adaptive Antennas*. PhD thesis, Virginia Tech, Blacksburg, VA, September 1995.

[42] B. Daneshrad and L. J. Cimini Jr., "Equalization requirements for 30 Mbps indoor wireless data transmission," in *Proc., IEEE Veh. Tech. Conf.*, vol. 1, pp. 71–5, 1996.

[43] R. A. Valenzuela, "Performance of adaptive equalization for indoor radio communications," *IEEE Trans. Commun.*, vol. 37, pp. 291–293, Mar. 1989.

Vita

John (Jack) Morton was born in Towson, Maryland on March 30, 1974. In 1996 he received his B.S. degree in Electrical and Computer Engineering with honors from Carnegie Mellon University in Pittsburgh, Pennsylvania. During the summer of 1996 he worked as an intern for the Microelectronics division of Lucent Technologies in Allentown, Pennsylvania where he designed fuse circuits for A/D converters. In the Fall of 1996 he began his graduate studies at the University of Illinois at Urbana-Champaign, but after completing one semester, he transferred to Virginia Tech where he was awarded a Bradley Fellowship by the Electrical Engineering department. In the Spring of 1997 he joined the Mobile and Portable Radio Research Group (MPRG) of Virginia Tech under the supervision of Dr. Brian D. Woerner. While at the MPRG he developed a book on adaptive equalization and adaptive arrays as part of the National Science Foundation's Combined Research Curriculum Development (CRCD) project. In August of 1998, he will join Motorola's Semiconductor Product Sector in Austin, Texas where he will design high-bandwidth data applications for Motorola's DSP chipsets.