

**SIMULATION OF ADAPTIVE EQUALIZATION
IN TWO-RAY, SIRCIM, AND SMRCIM MOBILE RADIO CHANNELS**

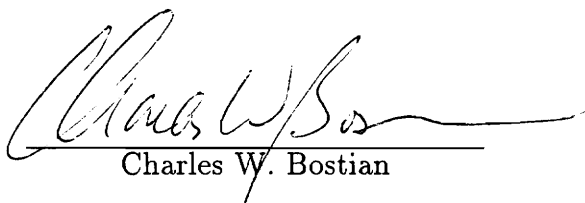
by
Weifeng Huang

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

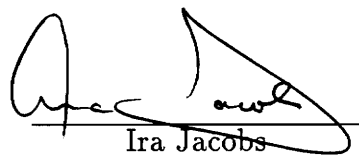
APPROVED:



Theodore S. Rappaport, Chairman



Charles W. Bostian



Ira Jacobs

December, 1991
Blacksburg, Virginia

c.2

LD

5655

V855

1991

H829

C.2

SIMULATION OF ADAPTIVE EQUALIZATION IN TWO-RAY, SIRCIM, AND SMRCIM MOBILE RADIO CHANNELS

by

Weifeng Huang

Theodore S. Rappaport, Chairman

Electrical Engineering

Abstract

This work presents a study of the adaptive equalization techniques designed to improve the bit error rates of digital transmissions degraded by intersymbol interference in radio communication. This thesis considers the following structures: the linear transversal equalizer (LTE), the decision feedback equalizer (DFE), the lattice equalizer, and the maximum likelihood sequence estimation (MLSE) equalizer. Least mean square (LMS) and recursive least squares (RLS) algorithms are used as the adaptive algorithms for these equalizers. Lattice-DFE, DFE, and MLSE with an RLS algorithm are recommended to be implemented in mobile systems because of their better performances. A two-ray Rayleigh fading channel model is used to simulate the mobile channels. The results show that adaptive equalization can significantly improve the performance of mobile communications if the channel does not change too fast. The simulation shows that if the delay (τ) of the second ray is too small, the adaptive equalization will degrade the BER performance, and the value of τ at which the adaptive equalizer can improve the BER is determined by the speed of the mobile channel variation. Also, simulation results obtained by using SIRCIM, a real world indoor channel simulator, shows that adaptive equalization has good performance in slowly varying channels. An equalizer working in indoor high data rate systems has a BER less than 10^{-3} at 15 dB E_b/N_o . The SMRCIM urban channel model is also developed and implemented for equalization simulation. Finally, equalization structures for differential modulation techniques are proposed.

Acknowledgements

I would like to thank Professor Theodore Rappaport, my academic advisor, for his guidance and support. I also want to thank my committee members, Professor Charles Bostian and Professor Ira Jacobs for their corrections and comments.

I thank every member of the Mobile and Portable Radio Research Group, who assisted me whenever I needed help. Special thank to Ken Blackard for reviewing my thesis.

This work is supported in part by the Mobile and Portable Radio Research Group Industrial Affiliates Program.

I must thank my parents for their love and encouragement, my brother and his wife for their moral and financial support. I also thank Professor Virginia C. Li in UCLA, who helped me to come to the United States.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	ISI and Its Effects	2
1.3	Adaptive Equalizers	3
1.4	Overview of Thesis.....	5
2	Channels and Equalization	7
2.1	Channel Parameters.....	7
2.2	Equalization.....	11
2.3	Structures of Adaptive Equalizers	14
3	Least Mean Square Algorithm.....	19
3.1	Derivation of the LMS Algorithm	19
3.2	Convergence Requirements of the LMS Algorithm	25
3.3	Excess Mean Squared Error	25
3.4	Convergence and Tracking Properties of LMS	27
3.5	Fast Converging LMS Algorithms	29
3.6	Numerical Example	30
4	Recursive Least Square Algorithms	34
4.1	Recursive Least Square Error	34
4.2	Derivation of Kalman RLS Algorithms.....	35
4.3	Excess MSE of RLS	39
4.4	Convergence and Tracking Properties of RLS.....	39
4.5	Fast Transversal Filter Algorithm.....	41
4.6	Least Square Adaptive Lattice Equalizer.....	44
4.7	Gradient Adaptive Lattice Equalizer.....	48
4.8	Decision Feedback Fast Kalman Algorithm.....	50
4.9	Least Square Lattice Decision Feedback Equalizer	52
4.10	Summary of algorithms.....	57
4.11	Numerical Example	57

5. Maximum Likelihood Sequence Estimation	61
5.1 Introduction.....	61
5.2 Channel Impulse Response Estimator.....	61
5.3 Matched Filter.....	64
5.4 Maximum Likelihood Sequence Estimation (Viterbi Decoder).....	66
5.5 Viterbi Algorithm	67
5.6 MLSE with a Fixed Matched Filter	69
5.7 Summary	70
 6 Equalization for Stationary Channels	 71
6.1 Stationary Channel Simulations	71
6.2 Equalization Simulation	73
6.3 Simulation Results	75
6.3.1 Convergence and SS-MSE	75
6.3.2 Tracking Capability	88
6.4 Summary and Discussion	97
 7 Simulation of Mobile Channels.....	 99
7.1 Two-Ray Rayleigh Fading Channel.....	99
7.1.1 Rayleigh Fading Channel.....	99
7.1.2 Two-Ray Channel.....	103
7.1.3 Power Loss in Wideband Channels.....	106
7.2 Evaluation of E_b/N_o	108
7.3 Simulation of Equalization in Two-Ray Channels.....	108
7.4 Simulation Results	110
7.5 Band-limited Channels.....	115
7.6 Simulation of Raised-cosine Filters.....	116
7.7 Results of Equalization in Systems with Raised-cosine Filters	120
7.8 Summary	125
 8 SIRCIM indoor channel.....	 129
8.1 SIRCIM	129
8.2 Implementation of SIRCIM.....	130
8.3 Characteristics of Indoor Channels and Equalization Simulation	133

8.4 Simulation Results	134
9 SMRCIM Urban Channel.....	152
9.1 SMRCIM	152
9.2 Equalization in SMRCIM Channels.....	154
9.3 Simulation Results	154
10 Novel Structure of Equalization for Differential Modulations.....	162
10.1 Differential Modulation	162
10.2 Equalization in $\pi/4$ DQPSK.....	163
10.3 Simulation Results	165
11 Conclusion	169
References	172
Appendix A. Software	176
Appendix B. Computer Programs.....	182
Vita.....	247

List of Figures and Tables

Figure 2.1-1.	Example of a measured impulse response	8
Figure 2.2-1.	Block diagram of a simplified communication system	12
Figure 2.3-1.	The structure of a linear transversal equalizer	15
Figure 2.3-2.	The structure of a decision feedback equalizer	17
Figure 4.6-1.	The structure of a lattice adaptive least square equalizer	45
Figure 4.7-1.	The structure of a gradient adaptive lattice equalizer	49
Figure 4.9-1.	The structure of a lattice decision feedback equalizer	53
Figure 5.1-1.	The structure of a maximum likelihood sequence estimator with an adaptive matched filter	62
Figure 5.1-2.	The structure of a maximum likelihood sequence estimator with a non-adaptive matched filter	62
Figure 5.2-1(a).	Implementation of an adaptive equalizer	63
Figure 5.2-1(b).	Implementation of a channel impulse response estimator	63
Figure 5.5-1.	Implementation of the Viterbi algorithm	68
Figure 6.1-1.	The power spectral densities of channel A and B	74
Figure 6.3-1.	Convergence of LMS DFE and LMS LTE in channel A. SNR=30dB, $\alpha=0.02$	76
Figure 6.3-2.	Convergence of LMS DFE and LMS LTE in channel B. SNR=30dB, $\alpha=0.02$	77
Figure 6.3-3.	Convergence of LSALE, GALE and L-DFE in channel A. SNR=30dB, $\lambda=0.997$	79
Figure 6.3-4.	Convergence of LSALE, GALE and L-DFE in channel B. SNR=30dB, $\lambda=0.997$	80
Figure 6.3-5.	Convergence of Kalman DFE and Kalman LTE in channel A. SNR=30dB, $\lambda=0.997$	81
Figure 6.3-6.	Convergence of fast Kalman DFE and FTF in	

	channel A. SNR=30dB, $\lambda=0.997$	82
Figure 6.3-7.	Convergence of Kalman DFE and Kalman LTE in channel B. SNR=30dB, $\lambda=0.997$	84
Figure 6.3-8.	Convergence of fast Kalman DFE and FTF in channel B. SNR=30dB, $\lambda=0.997$	85
Figure 6.3-9.	Convergence of LMS-DFE(6,5) for different α in channel A	86
Figure 6.3-10.	Convergence of RLS-DFE(6,5) for different λ in channel A	87
Figure 6.3-11.	Convergence of different sized LMS-DFE in channel A, $\alpha=0.02$	89
Figure 6.3-12.	Convergence of different sized RLS-DFE in channel A, $\lambda=0.95$	90
Figure 6.3-13.	Tracking performance of LMS-DFE(4,4) with different α in channel A	93
Figure 6.3-14.	Tracking performance of different sized LMS-DFE in channel A. $\alpha=0.02$	94
Figure 6.3-15.	Tracking performance of RLS-DFE(4,4) with different λ in channel A	95
Figure 6.3-16.	Tracking performance of different sized RLS-DFE in channel A $\lambda=0.95$	96
Figure 7.1-1.	Theoretical power spectrum of a Rayleigh fading channel (From Fung's M.S. thesis [23])	101
Figure 7.1-2.	Structure of a Rayleigh fading channel generator	102
Figure 7.1-3(a).	Envelope of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and the receiver speed is 75 km/h.....	104
Figure 7.1-3(b).	Phase of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and the receiver speed is 75 km/h.....	104
Figure 7.1-4(a).	Envelope of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and receiver speed is 20 km/h.....	105
Figure 7.1-4(b).	Phase of a simulated Rayleigh fading channel.	

	The carrier frequency is 850 MHz, and the receiver speed is 20 km/h.....	105
Figure 7.1-5.	Narrow band path loss of a two-ray Rayleigh fading channel.....	107
Figure 7.1-6.	Wide band path loss of a two-ray Rayleigh fading channel.....	107
Figure 7.4-1.	BER vs. speed of the mobile using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. The carrier frequency is 850 MHz, $E_b/N_o = \infty$. Two-ray Rayleigh fading channel, and the delay of the second ray is T_s	111
Figure 7.4-2.	BER vs. E_b/N_o in two-ray Rayleigh fading channel using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. The delay of the second ray is T_s	112
Figure 7.4-3.	P_1/P_2 vs. BER in a two-ray Rayleigh fading channel using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. $E_b/N_o = 20$ dB, and the carrier frequency is 850 MHz. Mobile speed is 75 km/h.....	114
Figure 7.5-1(a).	Magnitude frequency responses of raised-cosine rolloff filters with different rolloff factors.....	117
Figure 7.5-1(b).	Impulse response (time) of raised-cosine filters with different rolloff factors.....	117
Figure 7.5-2(a).	The impulse response of two individual rays using a raised-cosine filter. The delay of the second ray is $\frac{1}{3} T_s$	118
Figure 7.5-2(b).	The combined impulse response of a two-ray channel using a raised cosine filter. $\tau = \frac{1}{3} T_s$	118
Figure 7.6-1.	A baseband model of a QPSK system using raised-cosine filters and equalization	119
Figure 7.7-1.	BER vs. τ of two-ray Rayleigh fading channel using DFE(3,2) at different speeds. Carrier frequency is 850MHz.....	121

Figure 7.7-2.	BER vs. the roll off factor of the raised-cosine filter using RLS-DFE(3,2) in a two-ray Rayleigh fading channel. $\frac{\tau}{T_s}=0.5$. Carrier frequency is 850 MHz, $v=75\text{km/h}$, $E_b/N_o=\infty$	123
Figure 7.7-3	BER vs. E_b/N_o using RLS-DFE(3,2) in a two-ray Rayleigh fading channel, when $\frac{\tau}{T_s}=\frac{1}{6}$ and $\frac{\tau}{T_s}=\frac{7}{12}$. Carrier frequency is 850 MHz, $v=75\text{km/h}$, $E_b/N_o=\infty$	124
Figure 7.7-4.	BER vs. τ for a two-ray Rayleigh fading channel with 5 different numbers of feedback taps of RLS-DFE and the number of forward taps is 5. Carrier frequency is 850 MHz, $v=75\text{ km/h}$, $E_b/N_o=\infty$	126
Figure 7.7-5.	BER vs. τ for a two-ray Rayleigh fading channel with 5 different forward tap numbers of RLS-DFE and the number of feedback taps is 2. Carrier frequency is 850 MHz, $v=75\text{ km/h}$, $E_b/N_o=\infty$	127
Figure 8.4-1.	An indoor wide band impulse response named as HOBS2 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% Obstructed in an open-plan building, T-R separation is 25 m. The spatial mean rms delay spread of HOBS2 is 138 ns	135
Figure 8.4-2.	BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation and a 20 Mbps data rate.....	137
Figure 8.4-3.	BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 20 Mbps data rate.....	138

Figure 8.4-4.	BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 10 Mbps data rate.....	139
Figure 8.4-5.	BER vs. E_b/N_o using different numbers fo forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 10 Mbps data rate.....	140
Figure 8.4-6.	BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 4 Mbps data rate	141
Figure 8.4-7.	BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 4 Mbps data rate	142
Figure 8.4-8.	BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 2 Mbps data rate	143
Figure 8.4-9.	BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 2 Mbps data rate	144
Figure 8.4-10.	BER vs. E_b/N_o at different low data rates using RLS-DFE(2,2) in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation	146

Figure 8.4-11.	An indoor wide band impulse response named as LOS4 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% line-of-sight in an open-plan building, and the T-R separation is 25 m. The spatial mean rms delay spread of LOS4 is 95.6 ns	147
Figure 8.4-12.	An indoor wide band impulse response named as HARDLOS2 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% line-of-sight in a hard partitioned building, and the T-R separation is 25 m. The spatial mean rms delay spread of HARDLOS2 is 33.2 ns.....	148
Figure 8.4-13.	BER vs. E_b/N_o at different data rates for SIRCIM indoor channel LOS4 using the optimal RLS-DFE sizes obtained from the optimization of channel HOBS2. LOS4 has a spatial mean RMS delay spread of 95.6 ns. The simulation uses QPSK modulation	149
Figure 8.4-14.	BER vs. E_b/N_o at different data rates for SIRCIM indoor channel HARDLOS2 using the optimal RLS-DFE sizes obtained from the optimization of channel HOBS2. HARDLOS2 has a spatial mean RMS delay spread of 33.2 ns. The system uses QPSK modulation.....	150
Figure 9.1-1.	An urban wide band impulse response named as W1 simulated by SMRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. T-R separation is 1 km. The spatial	

	mean rms delay spread is 3.09 ns	153
Figure 9.1-2.	The rms delay spread distribution of 100 urban channels generated by SMRCIM. The median rms delay spread is 1.78 μ s, and the maximum rms delay spread is 17.16 μ s	155
Figure 9.1-3.	The narrow band signal power distribution of a 10 meter channel generated by SMRCIM	156
Figure 9.3-1.	BER vs. E_b/N_o at several channels generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h. The spatial mean rms delay spreads of these channels are from 1 to 3 μ s. Bold curves are the mean BER	157
Figure 9.3-2.	BER vs. E_b/N_o at several channels generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h. The spatial mean rms delay spreads of these channels are from 5 to 10 μ s. Bold curves are the mean BER	159
Figure 9.3-3.	Mean BER vs. E_b/N_o at 10 channels randomly generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h	160
Figure 10.2-1(a).	Structure of implementing adaptive equalization to differential modulations	164
Figure 10.2-1(b).	Structure of a decision feedback equalizer in a differential modulation system	164
Figure 10.3-1.	BER vs. τ of two-ray Rayleigh fading channel using RLS-DFE(3,2) and $\pi/4$ -DQPSK at different speeds. Carrier frequency is 850 MHz	166
Figure 10.3-2.	BER vs. E_b/N_o of SIRCIM channel HOBS2 using RLS-DFE using $\pi/4$ -DQPSK at different speeds, and the sizes of the DFE are obtained from QPSK modulation. Carrier frequency is 1.3 GHz	168
Figure A-1.	Structures of program <i>TWORAY</i> : the Kalman DFE in two-ray Rayleigh fading channels, <i>LDFEMAIN</i> : RLS L-DFE in two-ray Rayleigh	

	fading channels, and <i>COMMAIN</i> : the MLSE in two-ray Rayleigh fading channels.....	177
Figure A-2.	Structures of program <i>BN2CUBIC</i> : interpolating SIRCIM and SMRCIM channels, <i>INDOOR</i> : the Kalman DFE in SIRCIM indoor channels, <i>URBAN</i> : the Kalman DFE in SMRCIM urban channels.....	179
Figure A-3.	Structures of program <i>DIFKAL</i> : the Kalman DFE for $\pi/4$ DQPSK in two-ray Rayleigh fading channels, <i>DINDOOR</i> : the Kalman DFE for $\pi/4$ DQPSK in SIRCIM indoor channels	181
Table 4.10-1.	Computational requirement, advantages and disadvantages for different LMS and RLS algorithms.....	68
Table 6.3-1.	Summary of convergence rates and SS-MSE of different equalizers and structures.....	91

1. Introduction

1.1 Motivation

In recent years, wireless communication has experienced fast development and certainly will continue to grow in the future. Transmitting signals through radio channels instead of wirelines has the advantage of mobility for the users, and therefore, the customers can carry their transceivers and be reached anywhere within a specific service area. Cellular telephones offer this kind of service to people in developed countries. Consumers can use them as conventional telephones, placing and receiving calls, even while driving. As a result, cellular telephones can improve efficiency.

Currently, many cellular telephone systems are in operation around the world. Most of them are analog using frequency modulation (FM), which is not spectrally efficient. Because of the increasing demand and the limitation of available spectrum, more spectrally efficient methods have to be adopted. With careful source encoding, some digital modulation techniques need much less spectrum to send digitized voice signals which can increase the channel capacity [38]. Digital modulations can improve the quality of wireless communication and also provide new services, such as simultaneous data and voice transmission. Currently, the Pan-European Group Speciale Mobile (GSM) is the first digital wireless telephone system in operation. The U.S. Digital Cellular telephone (USDC) standard (IS-54) [22] has been developed in the United States.

Personal communication Service (PCS) is also a robust service which ideally will provide users a single wireless communication unit that can be used anywhere in the world [38].

Digital modulations can be evaluated by the bit-error-rate (BER), which is defined as the average percentage of errors received out of total number of bits sent. Intersymbol interference (ISI) is one of the main causes of bit errors in a digital system. Due to improper channel and system filtering, ISI is caused by the spreading of signal pulses in the time domain. This results in signal pulses being dispersed into adjacent time slots which cause interference to the adjacent signal

pulses [34]. Sending a high data rate signal requires a large frequency spectrum. If the channel coherence bandwidth, which is defined as the frequency separation for which the signals are strongly correlated [33], is less than the signal spectrum, ISI will appear, and the channel is a frequency-selective channel. If the coherence bandwidth is larger than the signal bandwidth, there will be little or no ISI. Then the channel is a flat channel. Without equalization, the influence of ISI can be reduced by decreasing the symbol rate, which results in a smaller signal bandwidth. Hence, ISI is one of the main factors limiting the data rate a system can send if the BER is required to be under a certain level.

There are various techniques that can be used to improve the performance of a digital system such as coding, diversity and equalization. Adaptive equalization is specially designed to mitigate the influence of ISI and reduce the BER, which in turn can be used to increase the data rates. In the USDC standard, adaptive equalization is chosen as a receiver option, so that a receiver can use it when there is significant ISI.

1.2 ISI and Its Effects

In radio channels, multipath propagation occurs, which may be considered as the energy traveling from the transmitter to the receiver via more than one path. Some paths have different lengths, so parts of the transmitted signal arrive at the receiver at different times. As a result, in the impulse response (time domain) of the channel there may be several impulses with different time delays which may cause ISI. In the frequency domain, the channel spectrum deviates from the ideal frequency response of constant amplitude and linear phase. The multipath channel becomes a frequency selective channel, which means that the channel loss varies with frequency.

In order to make efficient use of the spectrum, signals need to be sent in as small a bandwidth as possible. To minimize spectrum occupancy, pulse shaping is used to limit the signal spectrum without causing ISI in a flat channel [1].

Whether a channel causes significant ISI is dependent on the normalized root-mean-square (rms) delay spread, $\frac{\sigma_\tau}{T}$, of the channel impulse response [23,41]. The rms delay spread, σ_τ , is defined in Section 2.1 and T is the symbol duration.

If $\frac{\sigma_\tau}{T}$ is very small (very close to zero), the channel impulse response is close to an impulse. This type of channel will cause little ISI. If $\frac{\sigma_\tau}{T}$ is large, the channel impulse response may be represented by several impulses with different delays, and this type of channel will cause severe ISI. Therefore, whether a channel will cause ISI depends not only on the channel rms delay spread, but also on the symbol rate. Higher symbol rate systems are more vulnerable to ISI.

In UHF mobile radio channels, there are usually many paths. If the data rate is too high for a particular channel, these multipath components can cause significant ISI. Values of σ_τ determined by various measurements are given for urban cellular [2] and indoor personal communication network (PCN) [3]. Equalizers can be helpful to mobile radio system performance.

Due to the spatial variations in the channel as a transmitter or receiver moves, mobile radio channels are time-varying, so the channel power loss keeps changing. This is why mobile radio channels are called fading channels. As a result, the equalizers should be self-adaptive in order to track the time variation of the channels.

1.3 Adaptive Equalizers

An adaptive equalizer is a self-designing filter operating on a recursive algorithm to flatten the frequency response and linearize the phase of frequency selective channels [18]. It is capable of working in an unknown and time-varying environment.

The general operation of an adaptive equalizer includes training and tracking. First, a known training data sequence is sent by the transmitter. The training sequence can be made of pseudo-random binary signals. Then, using the received signal and the known training sequence, the adaptive equalizer at the receiver utilizes a recursive algorithm to evaluate the channel and chooses the proper coefficients for the equalizer. After adapting to the channel, the equalizer starts to receive the true data, and at the same time, the algorithm keeps working to track the changing channel [18]. In a TDMA system, data are sent in blocks. The training sequence usually is at the beginning of a block, and each time the receiver receives a new data block, the equalizer is retrained using the same

training sequence [22].

There are several adaptive equalizer structures. Lucky [4] did research on adaptive equalization for pulse amplitude modulation (PAM) in stationary wire channels in the early 60's and proposed zero-forcing linear transversal equalizers (LTE). After that, decision feedback equalizers (DFE) were studied and are discussed in [5]. The DFE shows superiority over the LTE in performance [5].

Also, lattice equalizers were developed which are made of lattice filters [10,11]. Further, a lattice DFE was designed [12,13,14] so that it can utilize the features of both lattice and decision feedback equalizers. A maximum likelihood sequence estimator that uses a channel impulse response estimator and the Viterbi decoder was proposed in the early 70's [15,16,17]. This structure has been shown to be an effective method of equalization.

Some recursive algorithms are needed to update equalizers. An algorithm starts from defined initial conditions. After the training sequence, the algorithm converges to the optimum solution. In a fading channel, the algorithm also offers a tracking capability, so that the equalizer can track time variations of the channel, provided that the variations are sufficiently slow.

Two types of recursive algorithms are widely used for the operation of adaptive equalizers. The simpler one is the least mean square (LMS) algorithm, which was described and understood by the late 1960's [6] to be a simple and effective algorithm. Because of LMS's slow convergence, the recursive least square (RLS) algorithm emerged as a fast converging algorithm. Kalman RLS algorithm was first proposed by Godard in 1974 [7], and later many researchers improved the RLS algorithm to be more computationally efficient [8,9].

The performance of an equalizer algorithm is determined by various factors [18]:

1. Rate of convergence: The number of iterations required for the algorithm to converge close enough to the optimum solution in response to stationary inputs. A fast rate of convergence allows the equalizer to adapt rapidly to a stationary unknown environment.
2. Tracking ability: The ability of an equalizer to follow the variation of the channel.

3. Misadjustment: A quantitative measure of the deviation of the final value of the mean squared error from the optimal minimum mean squared error.
4. Computational requirements: The number of arithmetic operations required to make one complete iteration of the algorithm.
5. Numerical properties: When an algorithm is implemented numerically, numerical round-off noise in the computer causes some inaccuracies. This kind of error influences the stability of the algorithm.

Compared to an RLS algorithm, an LMS algorithm has a very low computational requirement, but its rate of convergence is very poor. Consequently, it needs a long training sequence that does not send data, which results in low transmission efficiency because of the large training overhead. Furthermore, an LMS algorithm has poor tracking ability which produces a large misadjustment due to the tracking error [6,28,37]. Thus, an LMS algorithm is more appropriate for stationary and slow varying channels, while an RLS algorithm is good for fast changing channels and for low overhead training.

In order to be used in communication systems, equalizers have to be built in hardware, and the algorithm should be programmed in a digital signal processor (DSP). However, to get an idea how well an equalizer and algorithm work, the computer simulation is very important in the evaluation of equalizers and algorithms.

1.4 Overview of Thesis

Chapter 1 through Chapter 5 of this thesis introduces the basic theory of adaptive equalizers and describes their operation. Also, different adaptive equalizer structures and algorithms that have been developed are presented. The derivations of some algorithms are shown, and the theoretical features of different equalizer structures and algorithms are discussed.

To show the performance of different equalizer structures and algorithms, Chapter 6 uses stationary channels to test the convergence rate and misadjustment of equalizers. Also, the tracking capability of the adaptive equalizers is evaluated by switching the stationary channels. In Chapter 7, two-

ray Rayleigh fading channels are used to simulate mobile channels and the performance of adaptive equalization in the two-ray channel is evaluated.

Many researchers [19,20,21] have used two-ray Rayleigh fading channels as a simple representation of mobile channels, but the validity of the two-ray model as a representative real world channel is not certain. Therefore, Chapter 8 uses the SIRCIM indoor channel model [35,42] to evaluate the equalization in real world indoor channels. Also, SMRCIM, a measurement based mobile radio channel impulse response model, is developed to simulate outdoor mobile channels. Chapter 9 shows equalization results for SMRCIM channels, so that the results for the two-ray model and the SMRCIM model can be compared.

Chapter 10 presents a novel adaptive equalization structure for $\pi/4$ DQPSK, the modulation technique adopted by the USDC standard, but the performance of the structure is not as good as for QPSK. Chapter 11 concludes the thesis and gives ideas for future work.

2. Channels and Equalization

2.1 Channel Parameters

Equalization cannot be explained without discussing communication channels, because equalizers are used to recover signals distorted by frequency-selective channels. A channel can be described by its wideband impulse response, which can be estimated at the receiver by sending a short RF burst from the transmitter. Figure 2.1-1 shows an typical impulse response of an urban UHF radio channel that was measured in Dusseldorf, Germany at 942 MHz [36]. The RF pulse duration of the measurement is 500ns. There are about 12 resolvable multipath components above the noise threshold, and there also may be unresolvable components in the profile if two or more multipath components have the same excess delay. The excess delay is the largest delay of multipath components within a certain power level [3]. The 30 dB excess delay is the maximum delay of a multipath component within 30 dB of the component with the largest power. The channel in Figure 2.1-1 has about 9 μ s 30 dB excess delay and 6 μ s 10 dB excess delay.

The wideband impulse response of a channel can be written as

$$f(t, \tau) = \sum_n f_n(t) \delta(\tau - \tau_n) \quad (2.1.1)$$

where $f(t, \tau)$ is the signal voltage, $f_n(t) = \alpha_n(t) e^{j\theta_n(t)}$, $\alpha_n(t)$ is the attenuation factor and $\theta_n(t)$ is the phase of the n th multipath component, and $\delta(\cdot)$ is a unit power impulse. In (2.1.1), t represents the impulse response at different times, τ is the excess delay and τ_n is the excess delay of the n th multipath components. The wideband multipath channels are usually quantified into their mean time excess delay ($\bar{\tau}$) and rms delay spread (σ), which are defined as

$$\bar{\tau} = \frac{\sum_k \alpha_k^2 \tau_k}{G_r}, \quad G_r = \sum_k \alpha_k^2 \quad (2.1.2)$$

and

$$\sigma_\tau = \sqrt{\bar{\tau}^2 - (\bar{\tau})^2}, \quad \bar{\tau}^2 = \frac{\sum_k \alpha_k^2 \tau_k^2}{G_r} \quad (2.1.3)$$

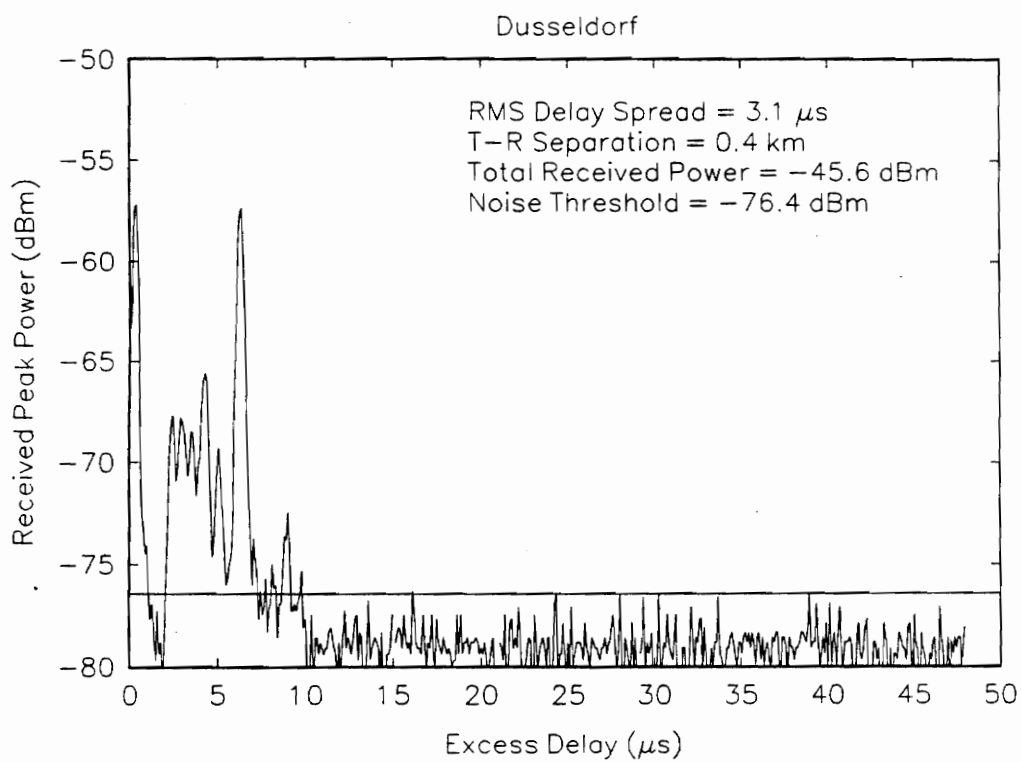


Figure 2.1-1. Example of a measured impulse response.

where G_r is the received power for a given profile, and the rms delay spread σ_r is in seconds [3]. Using (2.1.2) and (2.1.3), the rms delay spread of channel in Figure 2.1-1 is $3.1 \mu\text{s}$.

The BER performance of a radio communication system is highly dependent on σ_r of the channel. Without adaptive equalization, a larger σ_r usually causes higher BER [23,31]; therefore, σ_r is one of the important parameters of a channel for determining when to equalize.

Another important aspect of a channel is its time variation. A mobile channel changes constantly since a transmitter or receiver keeps moving, but how fast a mobile channel changes depends on many factors. The speed that a transceiver moves through the multipath channel is one factor; the faster the mobile, the faster the channel changes. Also, the carrier frequency is important to the variation of a channel, because a higher carrier frequency causes faster variations in amplitude and phase for a given travel distance. However, even if both transmitter and receiver stand still, the channel multipath components still keep changing because of the moving environment, such as moving cars and people. Therefore, if there exist fast moving objects in the propagation environment, the radio channel can change very fast even when both transmitter and receiver are stationary or move slowly. In addition, the symbol rate is also crucial in defining the channel variation, because for a given travel distance, the spatial separation between two consecutive symbols is smaller if the symbol rate is higher; thus, the channel change between the two symbols is smaller.

In the impulse response of a channel, the phase of a multipath component changes faster than its amplitude, because the phase of a radio carrier in a static channel changes 2π over the distance of one wavelength, while the power of the radio wave arriving by a single path changes much less in that short length. Although two consecutive impulse response profiles measured by a mobile receiver may not change much in amplitude, the phase of each multipath component definitely changes depending on the distance between the two profiles.

The velocity of the channel variation due to the moving of a transmitter or receiver can be defined by the Doppler frequency [32, 33]. The Doppler frequency f_D [33] is

$$f_D = \frac{v}{\lambda} \quad (2.1.4)$$

where v is the velocity of the mobile and λ is the wavelength of the carrier. It makes more sense to define the channel variation by the normalized Doppler frequency f_N

$$f_N = \frac{v}{\lambda f_s} \quad (2.1.5)$$

where f_s is the symbol rate, because a higher symbol rate results in a smaller variation of the mobile channel used by two consecutive symbols.

The variation of a mobile channel path loss is called fading. Fading may be divided into two types: narrow band continuous wave (CW) fading and wideband fading. The wideband fading can be measured at a receiver by sending an impulse or an approximation of an impulse, and then summing the power of all the multipath components in the impulse response. It can be written as [35]

$$E_w(t) = \sum_n |f_n(t)|^2 \quad (2.1.6)$$

Again as in (2.1.1) t represents the times that the impulse response is recorded, and $f_n(t)$ represents a multipath component in the profile which starts at time t . In wideband fading, multipath components with different path lengths cannot cancel each other, so the wideband channel will not be in deep fading unless all the multipath components f_n are small. This results in a small power fluctuation [39].

The CW fading can be measured by sending a continuous unmodulated radio wave. The CW fading can also be obtained from wideband measurements by calculating the phasor summation of all the multipath components in the wideband profile [35]:

$$E_n(t) = \left| \sum_n f_n(t) \right|^2 \quad (2.1.7)$$

Since the values of f_n are complex, they may cancel each other. Therefore, even

though all the multipath components have large amplitudes, the total energy may be very small at some instant of time.

The CW fading envelope usually has a Rayleigh or Rician distribution [32,33]. The Rayleigh fading channel is obtained from a scattering model, which has a large number of multipath components with equal power levels that arrive from random directions. When a stationary path with dominant power exists such as a line-of-sight (LOS) path, the Rician model is more appropriate in describing the envelope distribution [33].

Noise power is also a factor that affects the system performance. Signal-to-noise ratio (SNR), the ratio of the signal power and noise power at the receiver, is widely used as a system parameter indicating how the noise affects the system. In digital systems, E_b/N_o is used to quantify the performance in noise, where E_b is the energy per bit at the receiver, and N_o is the noise power spectral density after the detector.

2.2 Equalization

An equalizer is usually implemented at baseband. Since the baseband complex envelope expression can well represent the bandpass waveform [34], the channels and signals are simulated at baseband.

Figure 2.2-1 is a communication system with an adaptive equalizer at the receiver. The transmitter includes a modulator and RF transmitter. The receiver includes the RF stage, IF stage, matched filter and equalizer. The radio channel is between the transmitter and receiver. However, in the following analysis, the channel represents all the stages between the data input and equalizer instead of just the RF radio channel for the reason of simplification.

If the original information signal is $a(t)$, and the channel is represented as $f(t)$, the signal received by the equalizer is

$$y(t)=f^*(t)*a(t)+n_F(t) \quad (2.2.1)$$

where $*$ is a mathematical convolution, $f^*(t)$ is the complex conjugate of the channel impulse response $f(t)$, and $n_F(t)$ is the noise. Suppose the impulse

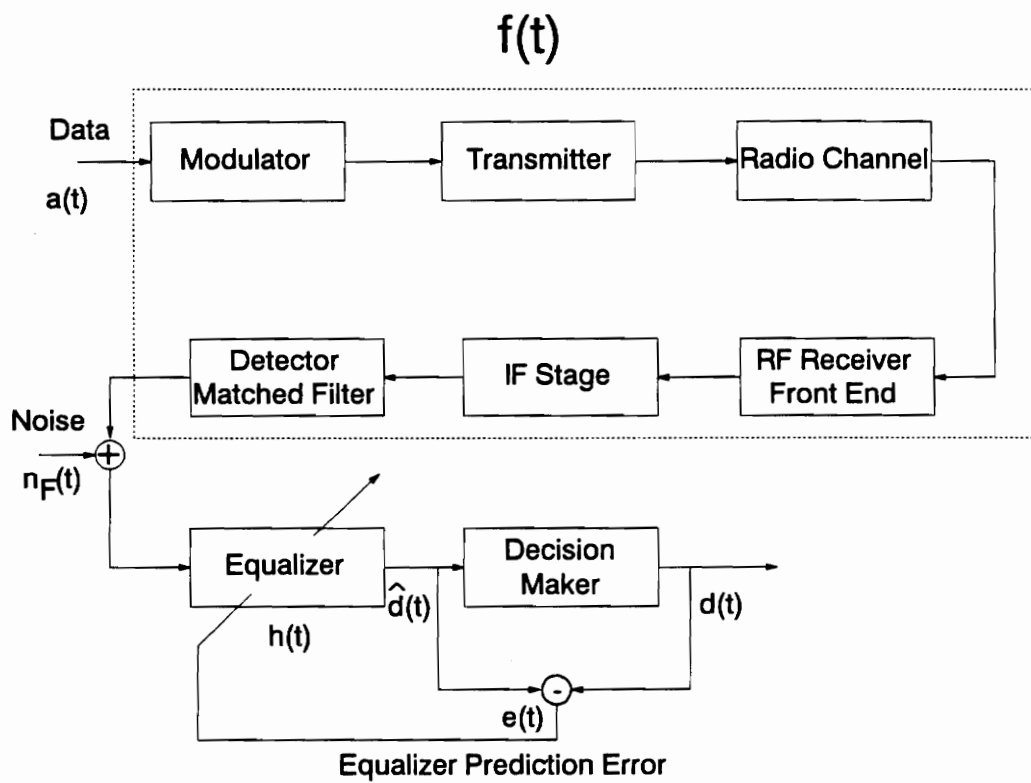


Figure 2.2-1. Block diagram of a simplified communication system.

response of the equalizer is $h(t)$, the output of the equalizer is

$$\begin{aligned}\hat{d}(t) &= f^*(t) * a(t) * h(t) + n_F(t) * h(t) \\ &= g(t) * a(t) + n_k(t)\end{aligned}\tag{2.2.2}$$

where $g(t) = f^*(t) * h(t)$. The impulse response of a transversal filter is

$$h(t) = \sum_n c_n \delta(t - nT)\tag{2.2.4}$$

$g(t)$ is actually the combined impulse response of the channel and the equalizer. The desired output of the equalizer is $a(t)$, which is the input of the channel. Assuming $n_k(t) = 0$, in order to get $\hat{d}(t) = a(t)$ in (2.2.2), the value of $g(t)$ must be

$$g(t) = f^*(t) * h(t) = \delta(t)\tag{2.2.5}$$

Equation (2.2.5) is the goal of the equalization. In the frequency domain, it can be expressed as

$$H^*(f)F(f) = 1\tag{2.2.6}$$

Equation (2.2.6) indicates that an equalizer is actually an inverse filter of the channel. If the channel is frequency selective, the equalizer enhances the low points and depresses the strong points in the channel frequency spectrum to get a flat combined spectrum. In other words, the basic function of an equalizer is to flatten the channel spectrum. In mobile communication, the channels change constantly. Adaptive equalizers are designed to track the variations so that Equation (2.2.6) can always be valid.

Because of the noise, n_k , the residual ISI, and the tracking error, (2.2.5) is hard to realize. Therefore, the combined spectrum will not always be flat. The prediction error of the equalizer is the error between the output of the equalizer before the decision making and the desired output after the decision making (see Figure 2.2-1). To express Equation (2.2.2) in the discrete form, let

$$\hat{d}(n)=g(n)*a(n)+n_k(n) \quad (2.2.7)$$

The prediction error is

$$e(n)=\hat{d}(n)-d(n)=\hat{d}(n)-[g(n)*a(n)+n_k(n)] \quad (2.2.8)$$

where n represents the time at nT . $E[|e(n)|^2]$, the mean squared error of an equalizer, is used as the criterion of how well an equalizer works. $E[|e(n)|^2]$ is the expected value (ensemble average) of the squared prediction error $|e(n)|^2$, but time-average can be used if $e(n)$ is stationary [34].

The minimum mean squared error usually results in a minimum bit error rate. The mean squared error is the variance of the prediction error. Suppose $|e(n)|$ is Gaussian distributed, the smaller the variance, the smaller the chance of the prediction error being larger than the threshold which results in a wrong decision, and consequently, there is a smaller probability of error. It would be better to minimize the probability of error (P_e) instead of mean squared error, but minimizing P_e results in non-linear equations, which are much harder to solve [1].

2.3 Structures of Adaptive Equalizers

There are several kinds of equalizers, which are actually baseband filters [1]. Some filters have been used as adaptive equalizer structures, such as the linear transversal filter [4], decision feedback equalizer [5], lattice filter [10,11], and maximum likelihood sequence estimator [15,16]. Each has an unique structure.

Linear Transversal Equalizer (LTE)

A linear transversal equalizer is made of a symbol-spaced (T -spaced) tapped-delay-line filter and a threshold detector as shown in Figure 2.3-1, where T is the symbol duration. The output of this transversal filter before the decision making is [27]

$$\hat{d}_k = \sum_{n=-N_1}^{N_2} c_n^* y_{k-n} \quad (2.3.1)$$

where the c_k 's are the complex coefficients or tap-gains of the filter, $\hat{d}_k$ is the output at time k , y_i is the input at time t_0+iT , t_0 is the equalizer starting time and $N=N_1+N_2+1$ is the number of taps.

STRUCTURE OF A TRANSVERSAL EQUALIZER

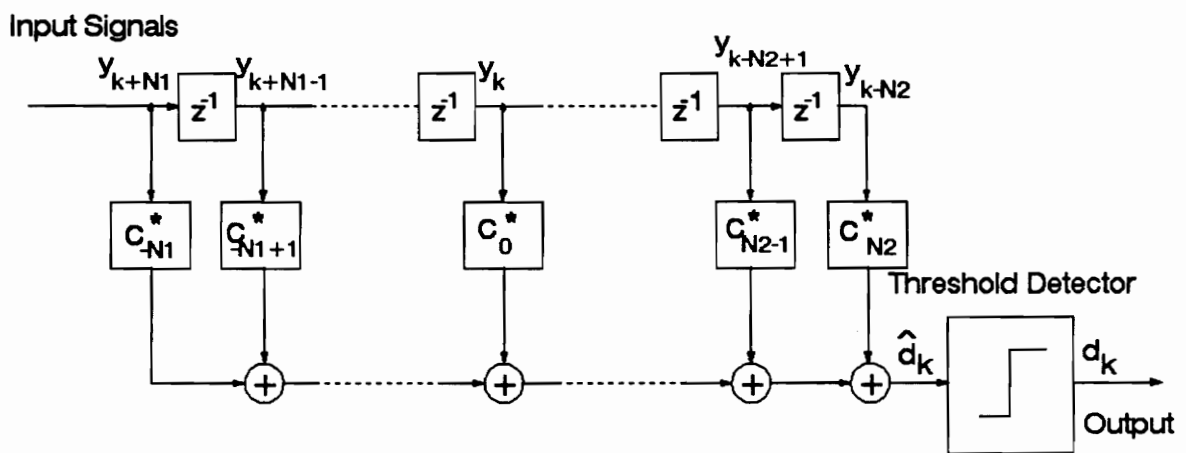


Figure 2.3-1. The structure of a linear transversal equalizer

In an LTE equalizer, the current and past values $y(t-nT)$ of the received signal are linearly weighted by tap gains c_i , and summed to produce the output $\hat{d}_k$. The detected k th symbol value d_k is the constellation point closest to the transversal filter's output $\hat{d}_k$. A constellation shows possible signal vectors corresponding to the possible digital signals [34].

The minimum mean squared error $E[|e(n)|^2]$ an LTE can achieve is [1]:

$$E[|e(n)|^2]_{min} = \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} \frac{N_0}{|F(e^{j\omega T})|^2 + N_0} d\omega \quad (2.3.2)$$

where $F(e^{j\omega T})$ is the frequency response of a channel, and N_0 is the noise spectral density.

Decision Feedback Equalizer(DFE)

A decision feedback equalizer is made of a forward transversal filter, a feedback filter, and a threshold detector. Figure 2.3-2 shows its structure. A DFE sums the outputs of the forward and feedback filters, and then makes decisions on symbol values. The output of a DFE before the threshold detector is [27]

$$\hat{d}_k = \sum_{n=-N_1}^{N_2} c_n^* y_{k-n} + \sum_{i=1}^{N_3} F_i^* d_{k-i} \quad (2.3.3)$$

where c_n and y_n are tap-gains and the inputs of the forward filter, the F_i is a tap-gain of the feedback filter, and the d_i for $(i < k)$ is the previous decision made on the detected signal. That is, once $\hat{d}_k$ is obtained using (2.3.3), d_k is decided from it. Then, d_k along with previous made decisions $d_{k-1}, d_{k-2}, \dots$ are fed back into the feedback equalizer, and $\hat{d}_{k+1}$ can be obtained using (2.3.3). This structure is particularly useful for channels with severe amplitude distortion in their frequency spectra [5].

The minimum mean squared error an DFE can achieve is [1]:

$$E[|e(n)|^2]_{min} = \exp \left\{ \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} \ln \left[\frac{N_0}{|F(e^{j\omega T})|^2 + N_0} \right] d\omega \right\} \quad (2.3.4)$$

STRUCTURE OF A DECISION FEEDBACK EQUALIZER

Input Signals

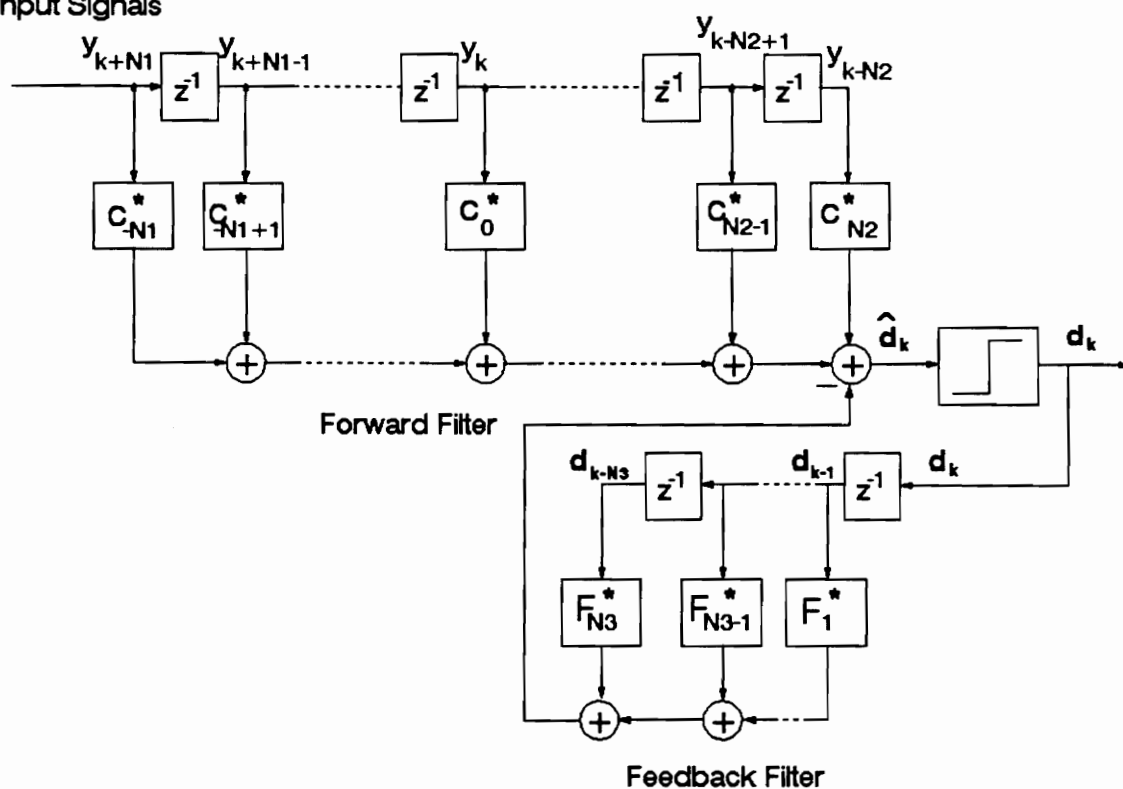


Figure 2.3-2. The structure of a decision feedback equalizer

It can be shown that the minimum MSE for DFE in (2.3.4) is always smaller than LTE in (2.3.3) unless $|F(e^{j\omega T})|$ is a constant (when equalization is not needed) [1]. If there are nulls in $|F(e^{j\omega T})|$, a DFE has significantly smaller minimum MSE than an LTE. Therefore, an LTE is well behaved when the channel spectrum is comparatively flat, but if the channel is severely distorted (the amplitude of the frequency response varies significantly with frequency within the signal spectrum), or exhibits nulls in the spectrum, the performance of an LTE deteriorates and the mean squared error of a DFE is much better than an LTE. In other words, a DFE is more appropriate for severely distorted channels [1].

Lattice Equalizer

A lattice equalizer is made of a lattice filter based on the Levinson-Durbin's algorithm, which is an order-recursive method to solve a set of linear equations [1]. A lattice equalizer can be used to mitigate ISI in frequency selective channels. There are several kinds of lattice equalizers, such as the LSALE and the lattice DFE [1,14], and they will be discussed Sections 4.6, 4.7, and 4.9. One of the advantages of a lattice equalizer is its high numerical stability. Also, the unique structure of the lattice filter allows the dynamic assignment of the most effective length of the lattice equalizer; hence, if a channel becomes more dispersive, the length of the equalizer can be increased without stopping the operation of the equalizer. However, the structure of a lattice equalizer is more complicated than a DFE or LTE.

Maximum Likelihood Sequence Estimator

A maximum likelihood sequence estimator (MLSE) is made of a channel estimator and a Viterbi decoder. Figure 5.1-1 is the structure of an MLSE. The channel estimator is an adaptive transversal filter estimating the channel impulse response. The Viterbi decoder [16] utilizes the Viterbi algorithm to test all the possible data sequences and chooses the one with the maximum probability as the detected data sequence. The MLSE techniques will be discussed in Chapter 5.

3. Least Mean Square Algorithm

Since an adaptive equalizer is capable of working in an unknown and time-varying environment, it needs an algorithm to update the equalizer coefficients for the unknown channel and also to track the channel variation. The least mean square (LMS) algorithm is one of the most widely used adaptive algorithms. It is simple to implement and does not require much computation. It can be used for both LTEs and DFEs.

3.1 Derivation of the LMS algorithm

The LMS algorithm is derived by minimizing the mean squared error of the equalizer. As shown in (2.3.3), the prediction error of a DFE can be expressed as

$$e_k = d_k - \hat{d}_k = d_k - \sum_{n=-N_1}^{N_2} c_n y_{k-n} - \sum_{n=1}^{N_3} F_n^* d_{k-n} \quad (3.1.1)$$

where all the variables are defined in (2.3.5), (2.3.6) and shown in Figure 2.3-2. As in (2.3.1), the prediction error of an LTE is

$$e_k = d_k - \hat{d}_k = d_k - \sum_{n=-N_1}^{N_2} c_n y_{k-n} \quad (3.1.2)$$

The mean squared error (MSE) $\epsilon = E[e_k^* e_k]$ is to be minimized. To simplify the expression in (3.1.1) and (3.1.2), the following condensed notations are used for DFE [27]:

$$\begin{aligned} \mathbf{v}_c &= (c_{-N_1}, \dots, c_0, \dots, c_{N_2})^H && : \text{vector of forward filter tap-gains} \\ \mathbf{v}_{a,k} &= (d_{k-1}, \dots, d_{k-N_2})^H && : \text{vector of signal detected} \\ \mathbf{v}_{y,k} &= (y_{k+N_1}, \dots, y_k, \dots, y_{k-N_2})^H && : \text{vector of signal received} \\ \mathbf{v}_F &= (F_1, \dots, F_{N_3})^H && : \text{vector of backward filter tap-gains} \end{aligned} \quad (3.1.3a)$$

where H represents the Hermitian transposition that is the operation of transposition combined with complex conjugation of each entry of the matrix [18]. Then,

$$\begin{aligned}\mathbf{v}_c^H \mathbf{v}_{y,k} &= \sum_{n=-N_1}^{N_2} c_n^* y_{k-n} \\ \mathbf{v}_F^H \mathbf{v}_{a,k} &= \sum_{n=1}^{N_3} F_n^* d_{k-n}\end{aligned}\quad (3.1.3b)$$

Therefore, applying the notations in (3.1.3) into (3.1.1), the prediction error for a DFE can be written as

$$e_k = d_k - \mathbf{v}_c^H \mathbf{v}_{y,k} - \mathbf{v}_F^H \mathbf{v}_{a,k} \quad (3.1.4)$$

By making the following definitions

$$\mathbf{w}_N = \begin{bmatrix} \mathbf{v}_c \\ \mathbf{v}_F \end{bmatrix}, \quad \mathbf{x}_N(k) = \begin{bmatrix} \mathbf{v}_{y,k} \\ -\mathbf{v}_{a,k} \end{bmatrix}, \quad N = N_1 + N_2 + 1 \quad (3.1.5)$$

Equation (3.1.4) can be written in the form of

$$e(n) = d(n) - \mathbf{w}_N^H \mathbf{x}_N(n) \quad (3.1.6)$$

where $\mathbf{w}_N$ is the tap-gain vector of both forward and feedback filters, and $\mathbf{x}_N(n)$ is the input data sequence vector for both forward and feedback filters.

If an LTE is analyzed, the following equations are used:

$$\mathbf{w}_N = \mathbf{v}_c, \quad \mathbf{x}_N = \mathbf{v}_{y,k}. \quad (3.1.7)$$

Then an LTE in (3.1.2) can also be written in the form of (3.1.6).

For a given channel, the prediction error $e(n)$ is dependent on the tap-gain vector $\mathbf{w}_N$, so the MSE of an equalizer is a function of $\mathbf{w}_N$. Let $J(\mathbf{w}_N)$ denote the mean squared error of an equalizer with a tap-gain vector $\mathbf{w}_N$, $J(\mathbf{w}_N)$ is

$$\begin{aligned}
J(\mathbf{w}_N) &= E[e(n)e^*(n)] \\
&= E[d(n)d^*(n)] - 2\mathbf{w}_N^H E[d(n)\mathbf{x}_N(n)] + \mathbf{w}_N^H E[\mathbf{x}_N(n)\mathbf{x}_N^H(n)] \mathbf{w}_N
\end{aligned} \tag{3.1.8}$$

where $E[\alpha(n)]$ is the expected value $\alpha(n)$, and it should be the ensemble average of the value of α at time n . If $\alpha(n)$ is stationary, the time average can be used instead [34]. By making the following definitions:

$$\mathbf{R}_{NN} = E[\mathbf{x}_N(n)\mathbf{x}_N^H(n)]$$

and

$$\mathbf{p}_N = E[\mathbf{x}_N(n)d^*(n)] \tag{3.1.9}$$

where $\mathbf{R}_{NN}$ is a N -by- N correlation matrix of the tap-input $\mathbf{x}_N(n)$, and $\mathbf{p}_N$ is the N -by-1 cross-correlation vector between the tap-input vector $\mathbf{x}_N(n)$ and the desired equalizer output $d(n)$ [18].

$\mathbf{R}_{NN}$ and $\mathbf{p}_N$ can be computed in the following way: let

$$r_x(m-k) = r_x(k-m) = E[x(n-m)x^*(n-k)] \tag{3.1.10}$$

$$r_{xd}(k) = E[d(n-k)x^*(n)] \tag{3.1.11}$$

$$\mathbf{R}_{NN} = \begin{bmatrix} r_x(0) & r_x(1) & \dots & r_x(N-1) \\ r_x(1) & r_x(0) & \dots & r_x(N-2) \\ \dots & \dots & \dots & \dots \\ r_x(N-1) & r_x(N-2) & \dots & r_x(0) \end{bmatrix} \tag{3.1.12}$$

$$\mathbf{p}_N = [r_{xd}(0), r_{xd}(1), \dots, r_{xd}(N-1)]^T \tag{3.1.13}$$

Equation (3.1.8) can be written as

$$J(\mathbf{w}) = E[d^*(n)d(n)] - 2\mathbf{w}_N^H \mathbf{p}_N + \mathbf{w}_N^H \mathbf{R}_{NN} \mathbf{w}_N \tag{3.1.14}$$

In order to minimize the MSE of $J(\mathbf{w}_N)$, its derivative is taken

$$\frac{\partial J(\mathbf{w}_N)}{\partial w_i} = -2r_{xd}(i) + r_x(i)w_0 + r_x(i-1)w_1 + \dots + r_x(N-i-1)w_{N-1} \text{ for } i=1,2,\dots,N \quad (3.1.15)$$

$$\frac{\partial J(\mathbf{w}_N)}{\partial \mathbf{w}_N} = -2\mathbf{p}_N + 2\mathbf{R}_{NN}\mathbf{w}_N \quad (3.1.16)$$

To get the minimum of $J(\mathbf{w}_N)$, set

$$\frac{\partial J(\mathbf{w}_N)}{\partial \mathbf{w}_N} = 0 \quad (3.1.17)$$

Then, from (3.1.16) and (3.1.17),

$$\mathbf{R}_{NN}\hat{\mathbf{w}}_N = \mathbf{p}_N \quad (3.1.18)$$

Equation (3.1.18) is called the normal equation which is obtained by minimizing the MSE in Equation (3.1.14); therefore, $\hat{\mathbf{w}}_N$ is the optimal tap-gain vector to achieve the minimum value of MSE. When Equation (3.1.18) is satisfied, the minimum MSE of the equalizer is:

$$J_{opt} = J(\hat{\mathbf{w}}_N) = E[d(n)d^*(n)] - \mathbf{p}_N^H \hat{\mathbf{w}}_N \quad (3.1.19)$$

This equation is obtained by substituting (3.1.18) into (3.1.10).

To get the optimal tap-gain vector $\hat{\mathbf{w}}_N$, the normal equation in (3.1.18) has to be solved. There are several methods to do this. One is simply to calculate

$$\hat{\mathbf{w}}_N = \mathbf{R}_{NN}^{-1} \mathbf{p}_N \quad (3.1.20)$$

However, inverting a matrix needs $O(N^3)$ arithmetic operations [24]. Other methods like Gauss elimination [24] and Cholesky factorization [25] also need $O(N^2)$ operations per iteration. The advantage of these methods of directly solving (3.1.18) is that they only need as few as N symbol inputs to solve the normal equation; consequently, a long training sequence is not necessary.

Another approach, derived from the steepest descent method [27], adaptively solves a linear equation. It works like this: let $S=f(\mathbf{X})$. The optimal $\mathbf{X}$ value, $\mathbf{X}_o$, which achieves the minimum of S needs to be solved. The iteration

$$\mathbf{X}(i+1)=\mathbf{X}(i)-\mu\frac{\partial f(\mathbf{X}(i))}{\partial \mathbf{X}} \quad (3.1.21)$$

can be used to find the optimal value of $\mathbf{X}_o$, where $\mathbf{X}(i)$ denotes the value of $\mathbf{X}$ at time i and μ is a small positive number. If μ is small enough, $\mathbf{X}(i)$ will converge to its optimal value $\mathbf{X}_o$ after some number of iterations [24].

The same strategy is used to solve the normal equation. Actually (3.1.18) does not have to be solved; instead mean squared error $J(\mathbf{w}_N)$ in (3.1.8) should be minimized using the iteration procedure

$$\mathbf{w}_N(n+1)=\mathbf{w}_N(n)-\alpha\frac{\partial J(\mathbf{w}_N)}{\partial \mathbf{w}_N}, \quad (3.1.22)$$

As in (3.1.6)

$$\frac{\partial J(\mathbf{w}_N)}{\partial \mathbf{w}_N}=-2\mathbf{p}_N+2\mathbf{R}_{NN}\mathbf{w}_N \quad (3.1.23)$$

and as in (3.1.9)

$$\mathbf{R}_{NN}=E[\mathbf{x}_N(n)\mathbf{x}_N^H(n)], \quad \mathbf{p}_N=E[\mathbf{x}_N(n)d^*(n)]$$

So

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{w}_N} &= -2E[\mathbf{x}(n)d^*(n)] + 2E[\mathbf{x}(n)\mathbf{x}^H(n)]\mathbf{w}_N \\ &= -2E[\mathbf{x}(n)d^*(n) - \mathbf{x}(n)\mathbf{x}^H(n)\mathbf{w}_N] \\ &= -2E\{\mathbf{x}(n)[d^*(n) - \mathbf{x}^H(n)\mathbf{w}_N]\} \\ &= -2E[e^*(n)\mathbf{x}(n)] \end{aligned} \quad (3.1.24)$$

Then, substitute (3.1.24) into (3.1.22), the iteration becomes

$$\mathbf{w}_N(n+1)=\mathbf{w}_N(n)-\alpha E[e^*(n)\mathbf{x}_N(n)] \quad (3.1.25)$$

where $e(n)=d(n) - \hat{d}(n)$. $E[e^*(n)\mathbf{x}_N]$ can be computed using time-averaging:

$$E[e^*(n)\mathbf{x}_N] = \frac{1}{K} \sum_{i=0}^{K-1} [e^*(n-i)\mathbf{x}_N(n-i)] \quad (3.1.26)$$

where K is the length of the data block used to compute $E[e^*(n)\mathbf{x}_N]$. Equation (3.1.26) takes $K \times N$ multiplications and $(K-1) \times N$ additions and N divisions. To obtain an accurate estimate of $E[e^*(n)\mathbf{x}_N]$, K should be large. Then, (3.1.26) requires a large amount of computation.

By replacing $E[e^*(n)\mathbf{x}_N(n)]$ by its instantaneous unbiased estimate $e^*(n)\mathbf{x}_N(n)$, (3.1.25) is simplified as

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) - \alpha e^*(n)\mathbf{x}_N(n) \quad (3.1.27)$$

(3.1.27) is called the LMS algorithm [26]. In (3.1.27), $\mathbf{w}_N(n)$ is the tap-gain vector of the equalizer, $\mathbf{x}_N(n)$ is the input data sequence, $e(n)$ the prediction error of the equalizer, and α is the step-size that controls the speed of the equalizer to adjust its coefficients. If α is too small, the convergence process will be very long, but if α is too large, the process may not converge at all [27].

The steepest descent method in (3.1.21) guarantees convergence if μ is small enough. In the LMS algorithm, the convergence condition in a stationary and noiseless channel is that α is small enough. The value of α is typically chosen between 0.01 to 0.2. The LMS algorithm also offers a tracking ability, so that the equalizer can adjust itself to the channel variation. The LMS algorithm is very simple to compute and program, as well as to build hardware for. It only needs $2N$ multiplications in (3.1.27). Also, it demonstrates good performance in slow time-varying channels.

LMS Algorithm

The LMS algorithm is the simplest equalization algorithm. It needs only $2N+1$ operations per iteration. This is the advantage of an LMS algorithm. The following is the LMS algorithm:

$$\hat{d}(n) = \mathbf{w}_N^H(n)\mathbf{x}_N(n) \quad (3.1.28)$$

$$e(n)=d(n)-\hat{d}(n) \quad (3.1.29)$$

$$\mathbf{w}_N(n+1)=\mathbf{w}_N(n)-\alpha e^*(n)\mathbf{x}_N(n); \quad (3.1.30)$$

where N is the number of taps and α is the step-size.

3.2 Convergence Requirement of the LMS Algorithm

It was stated earlier in Section 3.1 that the convergence of an LMS algorithm requires α to be small enough. The convergence condition of an LMS algorithm is [27,28]

$$0 < \alpha < \frac{2}{\sum_{i=1}^N \lambda_i} \quad (3.2.1)$$

where λ_i is the i th eigenvalue of the covariance matrix $\mathbf{R}_{NN}$; N is the number of taps in the equalizer. In fact,

$$\begin{aligned} \sum_{i=1}^N \lambda_i &= \text{sum of the mean-square values of the } N \text{ tap inputs} \\ &= \text{total input power} \\ &= \mathbf{x}_N^H(n)\mathbf{x}_N(n) \end{aligned} \quad (3.2.2)$$

Therefore, the step-size α can be controlled by the total input power in order to avoid instability in the equalizer [18].

3.3 Excess Mean Squared Error

The minimum MSE of LTE and DFE equalizers is given in Equations (2.3.2) and (2.3.4). However, the actual MSE is usually larger than the optimal values due to the size of the equalizer and channel variation, and the extra part is called misadjustment or excess MSE. The total MSE of an equalizer can be written as

$$J_{total}=J_{opt}+J_{exc} \quad (3.3.1)$$

where J_{opt} is the minimum MSE an equalizer can achieve, assuming the channel is stationary. J_{total} is the total MSE an equalizer will achieve.

In equalizing time-invariant channels, the excess MSE is caused by the gradient noise in the tap-gains, which is caused by using $e(n)\mathbf{x}_N(n)$ instead of the true gradient $E[e(n)\mathbf{x}_N]$ in the LMS algorithm (refer to (3.1.25) and (3.1.27)). Let

$$\mathbf{g}(n)=E[e^*(n)\mathbf{x}_N(n)]-e^*(n)\mathbf{x}_N(n) \quad (3.3.2)$$

$\mathbf{g}(n)$ is called the gradient noise. Because of the gradient noise, an LMS algorithm cannot converge to its optimal tap-gain values and produces the excess MSE.

For an LMS algorithm, the variance of the excess MSE due to the gradient noise is proportional to J_{opt} , the optimal MSE of the equalizer [28].

In a stationary environment, the total squared error can be expressed as

$$J_{total}=J_{opt}+J_{exc}=J_{opt}+J_{noise}=(1+\eta_{noise})J_{opt} \quad (3.3.3)$$

where η_{noise} is the gradient noise coefficient. For an LMS algorithm, η_{noise} can be written as [28]

$$\eta_{noise}=\frac{\alpha}{2-2\alpha N}\text{Tr}[\mathbf{R}_{NN}] \quad (3.3.4)$$

where α is the step-size of the LMS algorithm, and $\text{Tr}[\mathbf{R}_{NN}]$ is the trace of the matrix $\mathbf{R}_{NN}$, which can also be expressed as [28]

$$\text{Tr}[\mathbf{R}_{NN}]=N[|x(n)|^2]=\sum_{i=1}^N \lambda_i \quad (3.3.5)$$

where λ_i is the i th eigenvalue of the covariance matrix $\mathbf{R}_{NN}$ and $x(n)$ is the input of the equalizer at time n . From (3.3.4), it is clear that the excess MSE is proportional to the size step α . The smaller the α , the smaller the excess MSE due to the gradient noise of tap-gains.

Assuming unit input power to the equalizer at all times, which is valid because of the AGC power control at the receiver, $E[|x(n)|^2]=1$,

$$\eta_{noise}=\frac{\alpha}{2-2\alpha N} N \quad (3.3.6)$$

From Equations (3.3.5) and (3.3.6), it appears that an equalizer with a larger number of taps will have a larger error, but this is not necessarily true. A larger N increases η_{noise} , but at the same time the residual ISI is smaller which results in smaller optimal MSE in (3.3.3). There is an optimal equalizer size for minimum total MSE, J_{total} , which induces the minimum BER. Therefore, the size of the equalizer should be optimized to obtain the best result.

3.4 Convergence and Tracking Properties of LMS

The convergence time for an adaptive algorithm is the time required (usually specified by the number of iterations or sample periods) for performance measures, such as the MSE, to decrease below some prescribed acceptable level [18]. In an unknown or time-varying channel, the equalizer can converge and track the channel variation. If the channel changes, the equalizer coefficients will converge to the new optimal values exponentially as $e^{-T/\tau}$, where T is the time constant and τ is the time [6]. The convergence time is smaller when the time constant decreases. For LMS equalizers, each eigenvalue λ_i of $\mathbf{R}_{NN}$ corresponds to a time constant T_i [27,28]:

$$T_i = \frac{1}{(2\alpha\lambda_i)} \quad (3.4.1)$$

The largest time constant is

$$T_{max} = \frac{1}{(2\alpha\lambda_{min})} \quad (3.4.2)$$

As a result, the overall convergence time depends on the largest time constant decided by the smallest eigenvalue and step-size α . From (3.4.2), if α is larger, the time constant is smaller; then the equalizer converges faster. But there is a limitation on the choice of α : if α is too large, the algorithm will not converge at all. Further, from (3.3.6), if α increases, excess MSE due to gradient noise will increase too, so the equalization performance will be degraded.

A mobile channel keeps changing due to the movement of the transmitter, receiver, and the medium. Hence, the algorithm should offer the equalizers a tracking capability. Thus, the algorithm should remain in operation after

convergence. In a time-varying environment, the tracking ability also depends on the time constant [6]. Therefore, an LMS algorithm with a larger α (within the limits imposed by stability) has a better tracking ability, but yields a larger misadjustment due to gradient noise.

Hence, for a time-varying channel, the excess mean squared error includes not only misadjustment caused by the gradient noise of the tap-gain vector, but also the misadjustment due to channel tracking error. The tracking error is caused by the adaptive algorithm's memory of previous data that hinders the tracking ability [6]; therefore, the tap-gains will lag the optimal tap-gain somewhat. The excess MSE can be written as

$$J_{exc} = J_{noise} + J_{lag} \quad (3.4.3)$$

From [28],

$$J_{lag} = \frac{N}{2\alpha} \sigma_w^2 \quad (3.4.4)$$

where N is the number of taps in the equalizer, and σ_w is the variance of the tap-gain values, assuming the optimal tap-gain values are Gaussian distributed. The excess MSE of an LMS algorithm operating in a time-varying channel is [28]

$$J_{exc} = \frac{\alpha}{2-2\alpha N} N \sigma_u^2 + \frac{N}{2\alpha} \sigma_w^2 \quad (3.4.5)$$

where $\sigma_u^2 = J_{opt}$.

Note in (3.4.5) that as α decreases, the gradient noise component decreases while the tracking lag increases. An increase in α improves the tracking but makes the gradient noise component worse. By optimizing (derivative set to zero) J_{exc} in (3.4.5), it can be concluded that the best α is chosen to be

$$\alpha = \frac{h}{1+hN} \quad (3.4.6)$$

where $h = \frac{\sigma_w}{\sigma_u}$. The optimal excess MSE for LMS is

$$J_{exc}^* = N \sigma_u \sigma_w + \frac{N^2}{2} \sigma_w^2 \quad (3.4.7)$$

or

$$J_{exc}^* = Nh\sigma_u^2(1 + \frac{Nh}{2}) \quad (3.4.8)$$

In other words, α should be chosen according to the variance of the optimal tap-gains and the optimal MSE in order to obtain the minimum excess MSE.

The previous results are based on the assumption that the optimal tap-gains are Gaussian distributed which is usually not met; but, these results retain the essence of the problem common to more complicated and realistic situations, which is that a large channel variation will cause excess MSE; and the best α is dependent on N , the optimal MSE and variation of the channel. In practice, many unknown and nonstationary systems will be met, so the α may need to be approximated through trials for the best results.

3.5 Fast converging LMS algorithms

Due mainly to their simplicity and low storage requirements, LMS algorithms have found wide use in applications dealing with stationary and nonstationary channels. But their convergence rates are slow and depend on the eigenvalue spread of the signal covariance matrix, $\mathbf{R}_{NN}$. The eigenvalue spread is defined as the ratio of the maximum eigenvalue of $\mathbf{R}_{NN}$ to the minimum eigenvalue [1]. As the eigenvalue spread increases, the convergence rate decreases. In the frequency domain, the eigenvalue spread corresponds to the variance of the channel power spectral density. In fact, the ratio of maximum to minimum eigenvalues is approximately equal to the maximum value to the minimum value of the channel power spectral density [29]. Hence, the convergence of an LMS is quite slow, especially when nulls exist in the channel spectrum.

One method to speed up the convergence is to replace the scalar α by the pre-computed inverse matrix $\mathbf{R}_{NN}^{-1}$. This is a direct noniterative solution because, regardless of the initial conditions, the eigenvalue spread of $\mathbf{R}_{NN}$, or the equalizer size N , the optimum solution is obtained in one step [29]. If the signal is nonstationary, $\mathbf{R}_{NN}^{-1}$ has to be computed periodically, which takes a lot of computation power. However, this method may be used in the training period so that the equalizer can converge very fast. Then a long training sequence is not necessary. Afterwards, an LMS algorithm can still be used to update the

equalizer.

A variable step-size can also be used to improve the convergence. At the beginning, α is chosen to be large for the fast initial convergence, and then it can be decreased for better steady-state mean squared error (SS-MSE), which is the MSE after convergence [40].

Both methods for fast initial convergence do not improve the tracking ability of the LMS algorithm. However, in slow time-varying channels, such as indoor channels, the slow convergence is the main limitation of an LMS algorithm. With these fast convergence methods, LMS algorithms are still very useful.

3.6 Numerical Example

To illustrate the operation of an LMS algorithm, an numerical example is presented here. The channel is modeled by the transfer function

$$F(z) = \frac{1}{1 - \beta z^{-1}} \quad (3.6.1)$$

The output of the channel is

$$x(n) = \beta x(n-1) + y(n) \quad (3.6.2)$$

where $y(n)$ is the channel input, and $x(n)$ is the channel output. The channel is noiseless. A three tap LTE is used to equalize the channel, and the optimal transfer function of the equalizer should be

$$H(z) = 1 - \beta z^{-1} \quad (3.6.3)$$

In this example, β is set to 0.6, so the optimal tap-gain vector is

$$\hat{\mathbf{w}}_3 = \begin{bmatrix} 1 \\ -0.6 \\ 0 \end{bmatrix}, \quad (3.6.4)$$

If an LMS algorithm is applied, (3.1.27) is used to update the tap-gain vector

$$\mathbf{w}_3(n+1) = \mathbf{w}_3(n) - \alpha e(n) \mathbf{x}_3(n) \quad (3.1.27)$$

where $\mathbf{x}_3(n)=[x(n), x(n-1), x(n-2)]^T$ is the equalizer input data vector. The following shows the mechanism of an LMS algorithm.

Some initial values have to be set:

$$\alpha=0.1, \quad \mathbf{w}_3(1)=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_3(0)=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.6.5)$$

but they do not have to be these values. Then the equalizer starts to work. First, a bit $y(n)$ is sent into the channel, and then an output of the channel, $x(n)$, is obtained and applied to the equalizer. Using the previous tap-gain vector, $\mathbf{w}_3(n)$, the equalizer sends out the estimate of the bit value $\hat{d}(n)$ and the decision maker detects the bit value $d(n)$. The prediction error $e(n)=d(n)-\hat{d}(n)$ is computed, and it is used to update the tap-gain vector using (3.1.27).

$$y(1)=-1, x(1)=-1, \mathbf{x}_3^T(1)=[-1,0,0], \hat{d}(1)=\mathbf{w}_3^T(1)\mathbf{x}_3(1)=0, d(1)=y(1)=-1, e(1)=-1$$

Iteration 1: Equation 3.1.27 can be written as

$$\begin{bmatrix} w_1(2) \\ w_2(2) \\ w_3(2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 0.1 \times e(1) \times \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0 \\ 0 \end{bmatrix} \quad (3.6.6)$$

$$y(2)=-1, x(2)=-1.6, \mathbf{x}_3^T(2)=[-1.6,-1,0], \hat{d}(2)=\mathbf{w}_3^T(2)\mathbf{x}_3(2)=-0.16, \\ d(2)=y(2)=-1, e(2)=0.84,$$

Iteration 2:

$$\begin{bmatrix} w_1(3) \\ w_2(3) \\ w_3(3) \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0 \\ 0 \end{bmatrix} - 0.1 \times e(2) \times \begin{bmatrix} -1.6 \\ -1.0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.2344 \\ 0.084 \\ 0 \end{bmatrix} \quad (3.6.7)$$

$$y(3)=1, x(3)=0.04, \mathbf{x}_3^T(3)=[0.04,-1.6,-1.0], \hat{d}(3)=\mathbf{w}_3^T(3)\mathbf{x}_3(3)=-0.125, \\ d(3)=y(3)=1, e(3)=1.125$$

Iteration3 :

$$\begin{bmatrix} w_1(4) \\ w_2(4) \\ w_3(4) \end{bmatrix} = \begin{bmatrix} 0.2344 \\ 0.084 \\ 0 \end{bmatrix} - 0.1 \times e(3) \times \begin{bmatrix} 0.04 \\ -1.6 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.2389 \\ -0.096 \\ -0.1125 \end{bmatrix} \quad (3.6.8)$$

$$y(4)=-1, x(4)=-0.976, \mathbf{x}_3^T(4)=[-0.976, 0.04, -1.6], \hat{d}(4)=\mathbf{w}_3^T(4)\mathbf{x}_3(4)=-0.057, \\ d(4)=y(4)=-1, e(4)=0.943$$

It is clear that after 3 iterations, the LMS equalizer is no way close to convergence, and the prediction error is around 1. After 30 iterations, the equalizer starts to be close to the optimal value.

$$y(30)=1, x(30)=-0.2997, \mathbf{x}_3^T(30)=[-0.2997, -2.116, -1.944], \\ \hat{d}(30)=\mathbf{w}_3^T(30)\mathbf{x}_3(30)=0.7621, d(30)=y(30)=1, e(30)=-0.2379,$$

Iteration 30:

$$\begin{bmatrix} w_1(31) \\ w_2(31) \\ w_3(31) \end{bmatrix} = \begin{bmatrix} 0.8873 \\ -0.4651 \\ -0.0106 \end{bmatrix} - 0.1 \times e(30) \times \begin{bmatrix} -0.2997 \\ -2.116 \\ -1.944 \end{bmatrix} = \begin{bmatrix} 0.8802 \\ -0.5166 \\ -0.0569 \end{bmatrix} \quad (3.6.9)$$

$$y(31)=-1, x(31)=-1.180, \mathbf{x}_3^T(31)=[-1.18, -0.976, 0.04], \hat{d}(31)=\mathbf{w}_3^T(31)\mathbf{x}_3(31)=-0.7605, \\ d(31)=y(31)=-1, e(31)=-0.2379$$

The tap-gain vector is much closer to the optimal value and the prediction is about 0.2. After 60 iterations, the equalizer almost converges to the optimal value.

$$y(60)=-1, x(60)=-0.2561, \mathbf{x}_3^T(60)=[-0.2561, 1.24, 0.3996], \\ \hat{d}(60)=\mathbf{w}_3^T(60)\mathbf{x}_3(60)=-0.9824, d(60)=y(60)=-1, e(60)=0.01764$$

Iteration 60:

$$\begin{bmatrix} w_1(61) \\ w_2(61) \\ w_3(61) \end{bmatrix} = \begin{bmatrix} 0.9884 \\ -0.5881 \\ -0.0003 \end{bmatrix} - 0.1 \times e(60) \times \begin{bmatrix} 0.2561 \\ 1.24 \\ 0.3996 \end{bmatrix} = \begin{bmatrix} 0.9888 \\ -0.5903 \\ -0.001 \end{bmatrix} \quad (3.6.10)$$

$y(61)=-1$, $x(61)=-1.154$, $\mathbf{x}_3^T(61)=[-1.154,-0.2561,1.24]$,
 $\hat{d}(61)=\mathbf{w}_3^T(61)\mathbf{x}_3(61)=-0.9909$, $d(61)=y(61)=-1$, $e(61)=0.0091$

The tap-gain vector is very close to the optimal value and the prediction error is smaller than 0.01. Hence, it took about 60 iterations for the LMS algorithm to converge to the channel.

4 Recursive Least Square Algorithm

The LMS algorithms are developed from solving the normal equation in (3.1.18), which indicates that the approach is to minimize the MSE, the statistical measure of error. The LMS algorithms emulate the approach by using the steepest descent method to solve the normal equations.

Nevertheless, as discussed earlier, the convergence rate of an LMS algorithm is slow and adversely affected by the eigenvalue spread of the input covariance matrix $\mathbf{R}_{NN}$. The slow convergence is due to the fundamental limitation of a single adjustable step-size α in the LMS algorithms. Although using $\mathbf{R}_{NN}^{-1}$ can improve the convergence, the tracking ability of an LMS algorithm is still poor. Some other fast converging algorithms are needed to deal with unknown and fast time-varying channels.

The LMS algorithm is derived by using a statistical error measure, whereas its implementation utilizes the exact data available in the specific application. It is logical to find other techniques based on the error measures derived from the exact signal data. This idea leads to the family of powerful adaptive signal processing techniques known as the recursive least squares (RLS) approach, which significantly improves the convergence of adaptive equalizers [18].

4.1 Recursive Least Square Error

In deriving the faster convergence algorithm, a recursive least squares approach expresses the performance index in terms of a time mean instead of the statistical mean used by LMS algorithms. The least square error is defined as [18]

$$J(n) = \sum_{i=1}^n \lambda^{n-i} e^*(i,n) e(i,n) \quad (4.1.1)$$

where λ is the weighting factor close to 1, but smaller than 1, $e^*(i,n)$ is the complex conjugate of $e(i,n)$, and the error $e(i,n)$ is

$$e(i,n) = d(i) - \mathbf{x}_N^H(i) \mathbf{w}_N(n), \quad 0 \leq i \leq n \quad (4.1.2a)$$

$$\text{and} \quad \mathbf{x}_N(i)=[x(i),x(i-1),\dots,x(i-N+1)]^T \quad (4.1.2b)$$

$\mathbf{x}_N(i)$ is the data input vector at time i , and $\mathbf{w}_N(n)$ is the new tap-gain vector at time n ; therefore, $e(i,n)$ is the error using the new tap-gain at time n to test the old data at time i , and $J(n)$ is the cumulative squared error of the new tap-gains on all the old data.

The RLS problem requires finding the tap-gain vector of the equalizer $\mathbf{w}_N(n)$ such that the cumulative squared error measure $J(n)$ is minimized. It uses all the previous data to test the new tap-gains. The parameter λ is a data-weighting factor that weighs recent data more heavily in the computations, so that $J(n)$ tends to forget the old data in a nonstationary environment. If the data is stationary, λ may be set to 1 [1].

4.2 Derivation of Kalman RLS algorithm

RLS algorithms are derived by minimizing the least square error in (4.1.1). The Kalman algorithm is the simplest RLS algorithm. However, the computation requirement is high, especially when the size of the equalizer is large (N is large).

To obtain the minimum of least square error $J(n)$, the gradient of $J(n)$ in (4.1.1) is set to zero,

$$\frac{\partial}{\partial \mathbf{w}_N} J(n)=0 \quad (4.2.1)$$

It can be shown from (4.1.1) and (4.1.2), that [1]

$$\mathbf{R}_{NN}(n)\hat{\mathbf{w}}_N(n)=\mathbf{p}_N(n) \quad (4.2.2)$$

where $\hat{\mathbf{w}}_N$ is the optimal tap-gain vector of the RLS equalizer,

$$\mathbf{R}_{NN}(n)=\sum_{i=1}^n \lambda^{n-i} \mathbf{x}_N(i) \mathbf{x}_N^H(i) \quad (4.2.3)$$

$$\text{and} \quad \mathbf{p}_N(n)=\sum_{i=1}^n \lambda^{n-i} d^*(i) \mathbf{x}_N(i) \quad (4.2.4)$$

$\mathbf{R}_{NN}(n)$ in (4.2.3) is the deterministic correlation matrix of input data of the equalizer $\mathbf{x}_N(i)$, and $\mathbf{p}_N(n)$ in (4.2.4) is the deterministic cross-correlation vector between inputs of the equalizer $\mathbf{x}_N(i)$ and the desired output $d(i)$.

Theoretically, it would be possible to compute

$$\hat{\mathbf{w}}_N(n) = \mathbf{R}_{NN}^{-1}(n) \mathbf{p}_N(n) \quad (4.2.5)$$

but it needs a lot of computation to do the matrix inversion directly. Instead, it can be solved recursively.

From the definition of $\mathbf{R}_{NN}(n)$ in (4.2.3), it is easy to get

$$\mathbf{R}_{NN}(n) = \lambda \mathbf{R}_{NN}(n-1) + \mathbf{x}_N(n) \mathbf{x}_N^H(n) \quad (4.2.6)$$

Equation (4.2.6) is the update for $\mathbf{R}_{NN}(n)$ in the terms of $\mathbf{R}_{NN}(n-1)$. Since the three terms of the Equation (4.2.6) are all N -by- N matrices, a matrix inverse lemma can be used to derive a recursive update for $\mathbf{R}_{NN}^{-1}(n)$ in terms of the previous inverse, $\mathbf{R}_{NN}^{-1}(n-1)$. The lemma [25] states that if a square matrix can be written as

$$\mathbf{H}_{NN} = \mathbf{A}_{NN} + \mathbf{b}_N \mathbf{c}_N^H \quad (4.2.7)$$

then

$$\mathbf{H}_{NN}^{-1} = \mathbf{A}_{NN}^{-1} - \frac{\mathbf{A}_{NN}^{-1} \mathbf{b}_N \mathbf{c}_N^H \mathbf{A}_{NN}^{-1}}{1 + \mathbf{c}_N^H \mathbf{A}_{NN}^{-1} \mathbf{b}_N} \quad (4.2.8)$$

Implementing the matrix inverse lemma on Equation (4.2.6), the recursive update for $\mathbf{R}_{NN}^{-1}(n)$ becomes

$$\mathbf{R}_{NN}^{-1}(n) = \frac{1}{\lambda} \left[\mathbf{R}_{NN}^{-1}(n-1) - \frac{\mathbf{R}_{NN}^{-1}(n-1) \mathbf{x}_N(n) \mathbf{x}_N^H(n) \mathbf{R}_{NN}^{-1}(n-1)}{\lambda + \mu(n)} \right] \quad (4.2.9)$$

where

$$\mu(n) = \mathbf{x}_N^H(n) \mathbf{R}_{NN}^{-1}(n-1) \mathbf{x}_N(n) \quad (4.2.10)$$

Hence, $\mathbf{R}_{NN}^{-1}(n)$ can be updated recursively instead of computing the matrix inverse directly. To further simplify the expression, the following definitions are made.

$$\mathbf{C}_{NN}(\mathbf{n}) = \mathbf{R}_{NN}^{-1}(\mathbf{n}) \quad (4.2.11)$$

$$\mathbf{k}_N(\mathbf{n}) = \frac{\mathbf{C}_{NN}(\mathbf{n}-1)\mathbf{x}_N(\mathbf{n})}{\lambda + \mu(\mathbf{n})} \quad (4.2.12)$$

Then (4.2.9) becomes

$$\mathbf{C}_{NN}(\mathbf{n}) = \frac{1}{\lambda} [\mathbf{C}_{NN}(\mathbf{n}-1) - \mathbf{k}_N(\mathbf{n})\mathbf{x}_N^H(\mathbf{n})\mathbf{C}_{NN}(\mathbf{n}-1)] \quad (4.2.13)$$

By rearranging (4.2.12),

$$\mathbf{k}_N(\mathbf{n}) = \frac{1}{\lambda} [\mathbf{C}_{NN}(\mathbf{n}-1) - \mathbf{k}_N(\mathbf{n})\mathbf{x}_N^H(\mathbf{n})\mathbf{C}_{NN}(\mathbf{n}-1)]\mathbf{x}_N(\mathbf{n}) \quad (4.2.14)$$

From Equations (4.2.13) and (4.2.14)

$$\mathbf{k}_N(\mathbf{n}) = \mathbf{C}_{NN}(\mathbf{n})\mathbf{x}_N = \mathbf{R}_{NN}^{-1}(\mathbf{n})\mathbf{x}_N(\mathbf{n}) \quad (4.2.15)$$

The recursion in (4.2.13) provides a substantial saving in computation, and it needs $O(N^2)$ operations instead of $O(N^3)$ for the computation of the matrix inverse.

To obtain $\hat{\mathbf{w}}_N(\mathbf{n})$, the update of $\mathbf{p}_N(\mathbf{n})$ is still needed. From Equation (4.2.4),

$$\mathbf{p}_N(\mathbf{n}) = \lambda \mathbf{p}_N(\mathbf{n}-1) + \mathbf{d}^*(\mathbf{n})\mathbf{x}_N(\mathbf{n}) \quad (4.2.16)$$

From Equations (4.2.2), (4.2.13), and (4.2.16)

$$\begin{aligned} \mathbf{w}_N(\mathbf{n}) &= \mathbf{R}_{NN}^{-1}(\mathbf{n})\mathbf{p}_N(\mathbf{n}) = \mathbf{C}_{NN}(\mathbf{n})\mathbf{p}_N(\mathbf{n}) \\ &= \frac{1}{\lambda} [\mathbf{C}_{NN}(\mathbf{n}-1) - \mathbf{k}_N(\mathbf{n})\mathbf{x}_N^H(\mathbf{n})\mathbf{C}_{NN}(\mathbf{n}-1)] [\lambda \mathbf{p}_N(\mathbf{n}-1) + \mathbf{d}^*(\mathbf{n})\mathbf{x}_N(\mathbf{n})] \end{aligned} \quad (4.2.17)$$

Finally,

$$\mathbf{w}_N(\mathbf{n}) = \mathbf{w}_N(\mathbf{n}-1) + \mathbf{k}_N(\mathbf{n})[\mathbf{d}^*(\mathbf{n}) - \mathbf{w}_N^H(\mathbf{n}-1)\mathbf{x}_N(\mathbf{n})]$$

$$\begin{aligned}
&= \mathbf{w}_N(n-1) + \mathbf{k}_N(n)[d^*(n) - \hat{d}^*(n)] \\
&= \mathbf{w}_N(n-1) + \mathbf{k}_N(n)e^*(n, n-1)
\end{aligned} \tag{4.2.18}$$

where (4.2.18) is the tap-gain updating equation. Equations (4.2.12), (4.2.13), (4.2.16), and (4.2.18) make up the RLS Kalman algorithm, which first computes the N components of the new gain vector $\mathbf{k}_N(n)$, then uses those components to update $\mathbf{w}_N(n)$ in an optimal least squares fashion. An initial value of $\mathbf{R}_{NN}^{-1}(0)$ or $\mathbf{C}_{NN}(0)$ has to be assumed. Godard [7] found that the choice is not crucial, as long as $\mathbf{R}_{NN}(0)$ is nonsingular. Usually $\mathbf{C}_{NN}(0)$ is set to $\delta \mathbf{I}_{NN}$, where the constant $\delta \gg 1$ and $\mathbf{I}_{NN}$ is the N -by- N identity matrix.

Unlike an LMS algorithm, the convergence rate of an RLS algorithm is insensitive to the eigenvalue spread of the correlation matrix of the tap-input vector $\mathbf{x}_N(n)$ [1,18]. The LMS update in (3.1.27) is similar to the Kalman tap-gain update equation in (4.2.18), but the important difference is that the RLS uses different scalars in the gain vector $\mathbf{k}_N(n)$ to update the different tap gains. This is the reason the RLS is insensitive to the eigenvalue disparity ratio [1]. The advantage of an RLS algorithm is its fast convergence and good tracking ability. The disadvantage of the Kalman algorithm is its high computational requirement of $O(N^2)$ operations per iteration.

Recursive Least Square Kalman Algorithm.

The following is the Kalman algorithm, which is taken from [1]. It is used to update the equalizer coefficients. The Kalman algorithm converges very fast and has a good tracking capability.

Initialization: $\mathbf{w}(0) = \mathbf{k}(0) = \mathbf{x}(0) = \mathbf{0}$, $\mathbf{C}(0) = \delta \mathbf{I}_{NN}$, where $\mathbf{I}_{NN}$ is an N -by- N identity matrix, and δ is a large positive constant.

$$\hat{d}(n) = \mathbf{w}^H(n-1)\mathbf{x}(n); \tag{4.2.19}$$

$$e(n) = d(n) - \hat{d}(n) \tag{4.2.20}$$

$$\mathbf{k}(n) = \frac{\mathbf{C}(n-1)\mathbf{x}(n)}{\lambda + \mathbf{x}^H(n)\mathbf{C}(n-1)\mathbf{x}(n)} \quad (4.2.21)$$

$$\mathbf{C}(n) = \frac{1}{\lambda} [\mathbf{C}(n-1) - \mathbf{k}(n)\mathbf{x}^H(n)\mathbf{C}(n-1)] \quad (4.2.22)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)e^*(n) \quad (4.2.23)$$

where λ is the weighting coefficient that can change the performance of the equalizer. If a channel is time-invariant, λ can be set to 1. Usually $0.8 < \lambda < 1$ is used. The value of λ has no influence on the rate of convergence, but it determines the tracking ability of the RLS equalizers. The smaller the λ , the better the tracking ability. However, if λ is too small, the equalizer will be unstable [6]. The Kalman algorithm needs $2.5N^2 + 4.5N$ arithmetic operations per iteration [1].

4.3 Excess MSE of RLS

An RLS algorithm's excess MSE due to the gradient noise of the tap vector in a stationary environment is [6]

$$J_{noise} \simeq \frac{(1-\lambda)}{(1+\lambda)} N J_{opt} = \frac{(1-\lambda)}{(1+\lambda)} N \sigma_u^2 \quad (4.3.1)$$

$$\text{or} \quad J_{total} \simeq (1 + \eta_{noise}) J_{opt} \quad (4.3.2)$$

$$\text{where} \quad \eta_{noise} = \frac{(1-\lambda)}{(1+\lambda)} N \quad (4.3.3)$$

Hence, the excess MSE due to the gradient noise depends on the weighting factor λ , the number of taps in the equalizer and the optimal MSE. A larger λ causes a smaller η_{noise} .

4.4 Convergence and Tracking Properties of RLS

An RLS algorithm can converge to an unknown channel and track a time-varying channel. Its tracking capability is dependent on the time constant of the RLS, which is [6]

$$T_{RLS} \simeq \frac{1}{2(1-\lambda)} \quad (4.4.1)$$

So the time constant of an RLS algorithm decreases as the weighting factor λ decreases. Although the tracking time constant is determined completely by the λ , the initial convergence of RLS algorithms is independent of λ [6].

Since an adaptive algorithm that is capable of tracking rapidly changing signal characteristics will yield a larger excess MSE, it is not meaningful to compare the tracking characteristics of LMS and RLS algorithms unless the misadjustments they produced are known.

From (3.3.4), when α is small, the gradient noise coefficient for an LMS algorithm is

$$\eta_{noiseLMS} = \frac{\alpha}{2} \text{Tr}[\mathbf{R}] \quad (4.4.2)$$

Since $\text{Tr}[\mathbf{R}] = N \cdot \text{Ave}[\lambda_i]$, where $\text{Ave}[\lambda_i]$ is the mean value of eigenvalues and N is the number of equalizer taps,

$$\eta_{noiseLMS} = \frac{\alpha}{2} N \cdot \text{Ave}[\lambda_i] \quad (4.4.3)$$

If the LMS and the RLS algorithms have the same excess MSE due to gradient noise, that is $\eta_{noiseLMS} = \eta_{noiseRLS}$,

$$1 - \lambda = \alpha \cdot \text{Ave}[\lambda_i] \quad (4.4.4)$$

Substituting (4.4.4) into the expression of T_{RLS} in (4.4.1),

$$T_{RLS} = \frac{1}{2\alpha \text{Ave}[\lambda_i]} \quad (4.4.5)$$

Comparing T_{RLS} in (4.4.5) to the largest time constant of the LMS algorithm in (3.4.2) which is

$$T_{LMS} = \frac{1}{(2\alpha \lambda_{min})} \quad (4.4.6)$$

it can be seen that when $\text{Ave}[\lambda_i] \gg \lambda_{min}$, which means the eigenvalue disparity

ratio of the matrix $\mathbf{R}_{NN}$ is big, the tracking ability of RLS algorithms is much better than that of LMS algorithms when both have the same excess MSE value.

As mentioned earlier, the excess MSE includes the error caused by the gradient noise of the tap-gain vector and the channel tracking error. For RLS algorithms, this tracking error is [37]

$$J_{lag} = \frac{N}{2(1-\lambda)} \sigma_w^2 \quad (4.4.7)$$

so
$$J_{exc} = J_{noise} + J_{lag} = N \frac{1-\lambda}{1+\lambda} \sigma_u^2 + \frac{N}{2(1-\lambda)} \sigma_w^2 \quad (4.4.8)$$

where σ_u^2 is the optimal mean squared error of the equalizer, and σ_w^2 is the variance of the optimal tap-gain which is assumed to be Gaussian distributed. A larger σ_w^2 indicates that the channel has a larger fluctuation. From Equation (4.4.8), one can see when $(1-\lambda)$ decreases, the gradient noise component decreases while the tracking error increases. By optimizing J_{exc} in Equation (4.4.8), the optimal MSE for RLS is [37]

$$J_{RLS} \simeq \begin{cases} Nh\sigma_u^2(1+Nh/2) & h < 2.5 \\ N\sigma_u^2(.6+.56h^2) & h \geq 2.5 \end{cases} \quad (4.4.9)$$

Comparing the J_{RLS} value in (4.4.9) with J_{LMS} in (3.4.8) for the LMS algorithm, it can be seen that an RLS algorithm has much better tracking ability when h is large, which means the variation of optimal tap-weight vector is large with respect to the optimal MSE. However, it has no tracking advantage over the LMS if h is small. Therefore, if the channel changes very slowly, there is no advantage for RLS in the tracking ability.

4.5 Fast Transversal Filter Algorithm

The superior converging and tracking advantages of RLS algorithms over LMS algorithms are achieved at the expense of increased computation per iteration in the Kalman algorithm. Specifically, to compute the gain vector $\mathbf{k}(n)$ in Equation (4.2.12), the N -by- N matrix $\mathbf{C}_{NN}$ in Equation (4.2.11) must be updated and stored once per iteration. Accordingly, on the order of N^2 arithmetic operations must be performed per iteration for the Kalman algorithm, while an

LMS algorithm requires only $2N$ operations per iteration. The complexity of the Kalman algorithm limits the usefulness of the RLS algorithms, because occasionally there is not enough time to finish the computation due to high data rate and limited computation power.

To overcome this limitation of the RLS Kalman algorithm, many fast RLS algorithms have been developed. The fast transversal filter (FTF) algorithm developed by Cioffi and Kailath [9] requires the least computation.

The development of fast algorithms relies on certain update relations that arise in the characterization of adaptive forward and backward linear predictions optimized in the least squares sense. Up to four transversal filters that have the same input are used to produce the RLS algorithm [18]. As a result, an efficient recursive algorithm is derived for updating the Kalman gain vector $\mathbf{k}_N(n)$ in (4.2.12) without explicit computation of the inverse matrix $\mathbf{C}_{NN}$ in (4.2.15). Therefore, the computational requirement of the FTF algorithm just increases linearly with the number of taps. Different techniques such as matrix operations and projection [9,18,26] can be used to derive the FTF algorithm.

The FTF algorithm is a fast RLS algorithm which requires only $O(N)$ operations per iteration. There are several versions of the FTF algorithm [18]. The one introduced here is called the normalized FTF algorithm, which needs the least computation and is more stable compared to other versions. The algorithm is taken from [18], and all the variables are defined and explained in detail in [18].

Initialization:

$$\mathbf{a}_N(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{c}_N(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \hat{\mathbf{k}}_N(0) = \mathbf{0}, \mathbf{w}_N(0) = \mathbf{0};$$

$\gamma_N(0) = 1.0$; $\mathfrak{F}_N(0) = \mathfrak{B}_N(0) = \delta$, where δ is a small positive constant, which should be much smaller than 1.

Start from $n=1$;

$$\eta_N(n) = \mathbf{a}_N^H(n) \mathbf{x}_{N+1}(n) \quad (4.5.1)$$

$$\mathbf{f}_N(n) = \gamma_N(n-1) \eta_N(n) \quad (4.5.2)$$

$$\mathfrak{T}_N(\mathfrak{n}) = \lambda \mathfrak{T}_N(\mathfrak{n}-1) + \mathbf{f}_N^*(\mathfrak{n}) \eta_N(\mathfrak{n}) \quad (4.5.3)$$

$$\gamma_{N+1}(\mathfrak{n}) = \frac{\mathfrak{T}_N(\mathfrak{n}-1)}{\mathfrak{T}_N(\mathfrak{n})} \gamma_N(\mathfrak{n}-1) \quad (4.5.4)$$

$$\hat{\mathbf{k}}_{N+1}(\mathfrak{n}) = \begin{bmatrix} 0 \\ \hat{\mathbf{k}}_N(\mathfrak{n}-1) \end{bmatrix} + \frac{\eta_N(\mathfrak{n})}{\lambda \mathfrak{T}_N(\mathfrak{n}-1)} \mathbf{a}_N(\mathfrak{n}-1) \quad (4.5.5)$$

$$\mathbf{a}_N(\mathfrak{n}) = \mathbf{a}_N(\mathfrak{n}-1) - \mathbf{f}_N^*(\mathfrak{n}) \begin{bmatrix} 0 \\ \hat{\mathbf{k}}_N(\mathfrak{n}-1) \end{bmatrix} \quad (4.5.6)$$

$$\psi_N(\mathfrak{n}) = \lambda \mathfrak{B}_N(\mathfrak{n}-1) \hat{\mathbf{k}}_{N+1, N+1}(\mathfrak{n}) \quad (4.5.7)$$

$$\gamma_N(\mathfrak{n}) = [1 - \gamma_{N+1}(\mathfrak{n}) \psi_N^*(\mathfrak{n}) \hat{\mathbf{k}}_{N+1, N+1}(\mathfrak{n})]^{-1} \gamma_{N+1}(\mathfrak{n}) \quad (4.5.8)$$

$$\text{Rescue variable} = [1 - \gamma_{N+1}(\mathfrak{n}) \psi_N^*(\mathfrak{n}) \hat{\mathbf{k}}_{N+1, N+1}(\mathfrak{n})] \quad (4.5.9)$$

$$\mathbf{b}_N(\mathfrak{n}) = \gamma_N(\mathfrak{n}) \psi_N(\mathfrak{n}) \quad (4.5.10)$$

$$\mathfrak{B}_N(\mathfrak{n}) = \lambda \mathfrak{B}_N(\mathfrak{n}-1) + \psi_N(\mathfrak{n}) \mathbf{b}_N^*(\mathfrak{n}) \quad (4.5.11)$$

$$\begin{bmatrix} \hat{\mathbf{k}}_N(\mathfrak{n}) \\ 0 \end{bmatrix} = \hat{\mathbf{k}}_{N+1}(\mathfrak{n}) - \hat{\mathbf{k}}_{N+1, N+1} \mathbf{c}_N(\mathfrak{n}-1) \quad (4.5.12)$$

$$\mathbf{c}_N(\mathfrak{n}) = \mathbf{c}_N(\mathfrak{n}-1) - \mathbf{b}_N^*(\mathfrak{n}) \begin{bmatrix} \hat{\mathbf{k}}_N(\mathfrak{n}) \\ 0 \end{bmatrix} \quad (4.5.13)$$

$$\hat{\mathbf{d}}(\mathfrak{n}) = \mathbf{w}_N^H(\mathfrak{n}-1) \mathbf{x}_N(\mathfrak{n}) \quad (4.5.14)$$

$$\alpha_N(\mathfrak{n}) = \mathbf{d}(\mathfrak{n}) - \hat{\mathbf{d}}(\mathfrak{n}) \quad (4.5.15)$$

$$\mathbf{e}_N(\mathfrak{n}) = \gamma_N(\mathfrak{n}) \alpha_N(\mathfrak{n}) \quad (4.5.16)$$

$$\mathbf{w}_N(\mathfrak{n}) = \mathbf{w}_N(\mathfrak{n}-1) + \hat{\mathbf{k}}_N(\mathfrak{n}) \mathbf{e}_N^*(\mathfrak{n}) \quad (4.5.17)$$

The performance of the FTF algorithm is basically the same as the Kalman

algorithm, because the mathematical bases of the two algorithms are the same. The FTF algorithm requires $7N+14$ arithmetic operations per iteration.

FTF's computational complexity increases linearly with N , the number of taps contained in the adaptive filter. Thus, the FTF algorithm has the convergence properties of the RLS algorithm, and the desired computational simplicity. However, the simplification is achieved at the expense of a much more complicated program statement, as proven by comparing length of the FTF algorithm with the LMS and Kalman algorithms.

Although the FTF algorithm needs less computation, it tends to be unstable due to the round off error. It is especially true when a low mantissa processor is used [9]. A four-byte floating point processor may be needed to implement an FTF algorithm [19]. One method to solve the problem of instability is to define a rescue variable:

$$\text{Rescue variable} = [1 - \psi_{N+1}(n) \gamma_{N+1}(n) k_{N+1, N+1}^*(n)] \quad (4.5.9)$$

If the rescue variable is below zero, $\mathbf{w}_N(n)$ is saved and then the equalizer is reset. An LMS algorithm may be used to during the period of time when the RLS algorithm is retraining itself [18].

4.6 Least Squares Adaptive Lattice Equalizer

The least squares adaptive lattice equalizer (LSALE) uses the RLS algorithm and a lattice structure filter, which offers a number of potential advantages over ordinary tapped-delay-line (TDL) filters. First, an $(N-1)$ -stage lattice filter generates all the outputs that can be generated by N different TDL filters with lengths from 1 to N filter taps. Therefore, the size of equalizer can be changed at anytime without interrupting the operation of the equalizer by selecting the different output $\hat{d}_i$ in Figure 4.6-1 [11]. The second advantage of the lattice structure is that longer lattice filters may be built up from shorter ones by simply adding on more lattice stages [11]. This is useful in designing large scale systems. Finally, an important property of the lattice structure is its insensitivity to roundoff error [11], so a lattice equalizer is more stable.

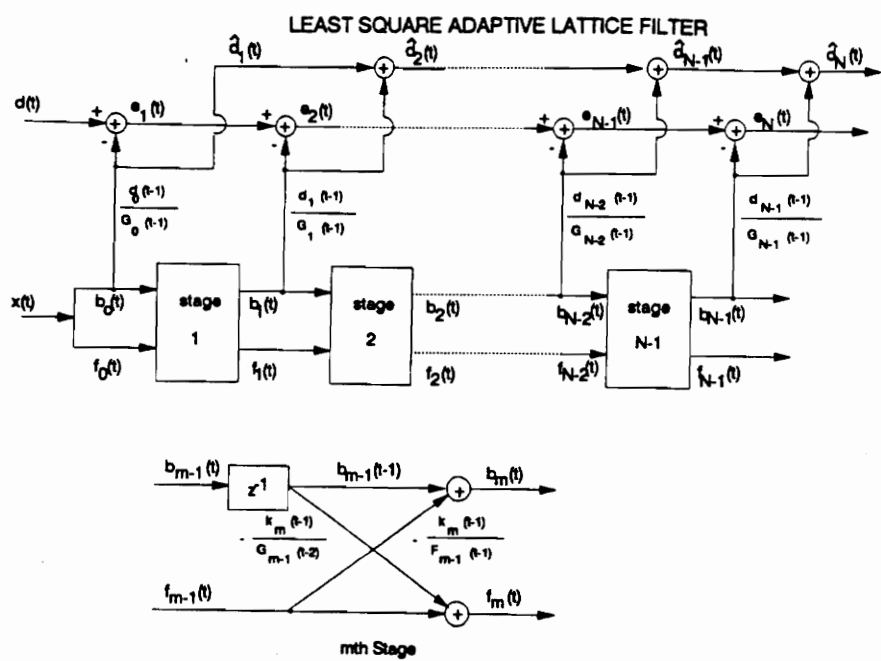


Figure 4.6-1. The structure of a lattice adaptive least square equalizer

Figure 4.6-1 shows the structure of an LSALE. The LSALE is made of stages such as the one shown in Figure 4.6-1. There are two reflection coefficients, $-\frac{k_m(t-1)}{G_{m-1}(t-2)}$ and $-\frac{k_m(t-1)}{F_{m-1}(t-1)}$ is a stage. The operation of a stage can be expressed as

$$b_m(t) = b_{m-1}(t-1) - \frac{k_m(t-1)}{G_{m-1}(t-2)} \cdot f_{m-1}(t) \quad (4.6.1a)$$

and

$$f_m(t) = f_{m-1}(t) - \frac{k_m(t-1)}{F_{m-1}(t-1)} \cdot b_{m-1}(t-1) \quad (4.6.1b)$$

where $b_m(t)$ is called the error of an m th-order backward predictor, and $f_m(t)$ is called the error of an m th-order forward predictor [1]. The output of the channel $x(t)$, and the errors of the backward predictors $b_1, \dots, b_i$ are weighted and added together, and the result is the output of the lattice equalizer $\hat{d}_i$. A decision is made upon $\hat{d}_i$ to obtain the output of equalizer, d . The value of i can be anything between 1 and the length of the lattice filter N , and it can be changed without interrupting the operation of the equalizer. If the local time average of the $|e_i|^2$ is the minimum, $\hat{d}_i$ can be chosen as the output of the equalizer [1], where e_i is the i th-order equalizer prediction error (See Figure 4.6-1).

The performance of the LSALE algorithm is basically the same as that of the Kalman algorithm. But the LSALE needs only $O(N)$ operations per iteration, which is less than Kalman algorithm and it is more stable. The derivation of the LSALE algorithm can be found in [1].

The following is the LSALE algorithm used to update the lattice equalizer. Compared to other RLS algorithms such as the Kalman and FTF algorithm, it is more stable and more flexible in implementation. The algorithm is taken from [1], and the algorithm and its variables are explained in detail in [1].

Initialization:

$$\epsilon_m^e(-1) = \epsilon_m^r(-2) = \epsilon_m^r(-1) = \delta;$$

$$r_m(-1) = k_m(-1) = \bar{k}_m(-1) = \gamma_m(-1) = 0;$$

where δ is a small positive number and should be much less than 1.

Start from $n=0$;

$$e_0(n)=r_0(n)=x(n) \quad (4.6.1)$$

$$\epsilon_0^e(n)=\lambda\epsilon_0^e(n-1)+|x(n)|^2 \quad (4.6.2)$$

$$y_{-1}(n)=\gamma_{-1}(n)=\gamma_{-1}(n-1)=0 \quad (4.6.3)$$

$$k_m(n)=\lambda k_m(n-1)+\gamma_{m-1}(n-1)e_m^*(n)r_m(n-1) \quad (4.6.4)$$

$$e_{m+1}(n)=e_m(n)-\frac{k_m(n-1)r_m^*(n-1)}{\epsilon_m^r(n-2)} \quad (4.6.5)$$

$$r_{m+1}(n)=r_m(n-1)-\frac{k_m(n-1)e_m(n)}{\epsilon_m^e(n-1)} \quad (4.6.6)$$

$$\epsilon_{m+1}^e(n)=\epsilon_m^e(n)-\frac{|k_m^2(n)|}{\epsilon_m^r(n-1)} \quad (4.6.7)$$

$$\epsilon_{m+1}^r(n)=\epsilon_m^r(n-1)-\frac{|k_m^2(n)|}{\epsilon_m^e(n)} \quad (4.6.8)$$

$$\gamma_m(n)=\gamma_{m-1}(n)+\frac{|(1-\gamma_{m-1}(n))r_m(n)|^2}{\epsilon_m^r(n)} \quad (4.6.9)$$

These recursions are computed for $m=0, \dots, N$, and the following steps are computed for $m=0, 1, \dots, N-1$; where N is the number of the stage of lattice filter.

$$y_m(n)=y_{m-1}(n)+\frac{\bar{k}_{m-1}^*(n-1)r_m(n)}{\epsilon_m^r(n-1)} \quad (4.6.10)$$

$$\bar{e}_{-1}(n)=d(n) \quad (4.6.11)$$

$$\bar{e}_m(n)=d(n)-y_m(n) \quad (4.6.12)$$

$$\bar{k}_m(n)=\lambda\bar{k}_m(n-1)+(1-\gamma_{m-1}(n))\bar{e}_{m-1}^*(n)r_m(n) \quad (4.6.13)$$

The $x(n)$ is the output of the channel. λ is the weighting factor. LSALE needs $19N-12$ arithmetic operations per iteration [11].

4.7 Gradient Adaptive Lattice Equalizer

The gradient adaptive lattice equalizer (GALE) is another kind of lattice filter that can be used to implement adaptive equalization. It has fewer adjustable parameters than the LSALE shown in Section 4.6, so it needs less computation than the LSALE. The results of the simplification are a slightly slower rate of convergence and a slightly larger mean squared error compared to the LSALE.

The structure of a GALE equalizer is in Figure 4.7-1. The form of the lattice filter is identical to the LSALE in the Section 4.6, except that there is only one reflection coefficient, $k_m(t)$ in a GALE. Its computation is approximately twice as efficient as the LSALE. The derivation of the GALE algorithm can be found in [1].

The following is the GALE algorithm used to update lattice equalizers. It is similar to the LSALE algorithm with a little worse performance. The algorithm is taken from [1]:

Initialization: $v_m(0)=\delta$; set anything else to zero. δ is a small positive constant, which is much smaller than 1.

$$f_0(n)=b_0(n)=x(n) \quad (4.7.1)$$

$$u_m(n)=\lambda u_m(n-1)+2f_{m-1}^*(n)b_{m-1}(n-1) \quad (4.7.2)$$

$$v_m(n)=\lambda v_m(n-1)+|f_{m-1}^2(n)|+|b_{m-1}^2(n-1)| \quad (4.7.3)$$

$$k_m(n)=u_m(n)/v_m(n) \quad (4.7.4)$$

$$y_{-1}(n)=0 \quad (4.7.5)$$

$$y_m(n)=y_{m-1}(n)+w_m^*(n)b_m(n) \quad (4.7.6)$$

$$e_m(n)=d(n)-y_m(n) \quad (4.7.7)$$

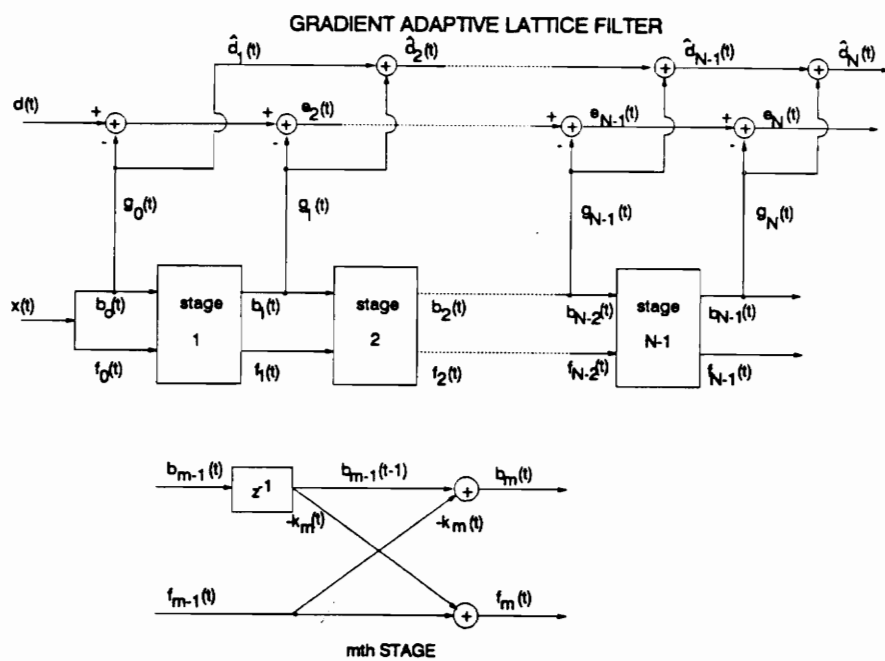


Figure 4.7-1. The structure of a gradient adaptive lattice equalizer.

$$w_m(n) = w_m(n-1) + \frac{2e_m^*(n-1)b_m(n-1)}{v_m(n-1)} \quad (4.7.8)$$

$$e(n) = e_N(n) \quad (4.7.9)$$

$m=1, \dots, N;$

λ is the weighting factor as the other RLS algorithms, y_m is the output of the equalizer before the decision making, and $e(n)$ is the equalizer prediction error. For most cases, the power estimates $v_m(n)$ in (4.7.3) will not have to be updated for every iteration, because it does not change very fast [10]. The GALE algorithm requires $13N-8$ arithmetic operations per iteration [10].

4.8 Decision Feedback Fast Kalman Algorithm

A decision feedback equalizer works better than a linear transversal equalizer on highly distorted channels, and the FTF algorithm is simple in computation. Therefore, it would be better to have an algorithm that combines the two techniques. Such an equalizer is called decision feedback fast transversal filter, which utilizes the advantages of both DFE and FTF: fast initial convergence and low MSE. However, this algorithm is not available in the literature. Then, the fast Kalman algorithm developed by Falconer and Ljung [8] for DFE is used.

The fast Kalman algorithm can be implemented in both the LTE and DFE structures. N_1 is the number of forward taps and N_2 is the number of feedback taps. $C_N(n)$ is the tap-gain vector. $y(n)$ is the channel output that is applied to the equalizer, and $\hat{d}(n)$ is the output of the equalizer before the decision making. $d(n)$ is the decision made upon the value of $\hat{d}(n)$, while $e(n) = d(n) - \hat{d}(n)$ is the equalizer prediction error. $k_N(n)$ is the gain vector as the one in (4.2.18), which is used to update the equalizer tap-gain $C_N(n)$.

The algorithm is taken from [8], and the algorithm and the variables are explained in [8]. Some mistakes in the original algorithm are corrected and some improper notations are changed here.

Explanation of notations:

$$\mathbf{x}_N(n-1)=[y(n-1),y(n-2),...,y(n-N_1),d(n-2),d(n-3),...,d(n-N_2-1)]^T$$

$\mathbf{x}_N(n)$ is the input vector of the forward and feedback filters.

$$\boldsymbol{\xi}_2(n)=[y(n), d(n-1)]^T$$

$$\boldsymbol{\rho}_2(n)=[y(n-N_1), d(n-N_2)]^T$$

$$\mathbf{x}_M(n)=[y(n), y(n-1),...,y(n-N_1), d(n-1), d(n-2),..., d(n-N_2)]^T$$

$\mathbf{S}_{MM}$ and $\mathbf{Q}_{MM}$ are permutation matrices

$$\mathbf{S}_{MM}\mathbf{x}_M(n)=\begin{bmatrix} \boldsymbol{\xi}_p(n) \\ \mathbf{x}_N(n-1) \end{bmatrix} \quad (4.8.1)$$

$$\mathbf{Q}_{MM}\mathbf{x}_M(n)=\begin{bmatrix} \mathbf{x}_N(n) \\ \boldsymbol{\rho}_p(n) \end{bmatrix} \quad (4.8.2)$$

$$M=N+2$$

Initialization:

$$\mathbf{A}_{Np}=\mathbf{D}_{Np}=\mathbf{0}_{Np}$$

$$\mathbf{E}_{pp}=\delta\mathbf{I}_{pp}, \quad \delta \text{ is a small positive number which is much smaller than 1.}$$

$$\mathbf{k}_N=\mathbf{0}_N$$

begin:

$$\boldsymbol{\epsilon}_p(n)=\boldsymbol{\xi}_p(n)+\mathbf{A}_{Np}^H(n-1)\mathbf{x}_N(n-1) \quad (4.8.3)$$

$$\mathbf{A}_{Np}(n)=\mathbf{A}_{Np}(n-1)-\mathbf{k}_N(n)\boldsymbol{\epsilon}_p(n)^H \quad (4.8.4)$$

$$\boldsymbol{\epsilon}_p(n)'=\boldsymbol{\xi}_p(n)+\mathbf{A}_{Nn}^H(n)\mathbf{x}_N(n) \quad (4.8.5)$$

$$\mathbf{E}_{pp}(n)=\lambda\mathbf{E}_{pp}(n-1)+\boldsymbol{\epsilon}_p(n)\boldsymbol{\epsilon}_p(n)^H \quad (4.8.6)$$

$$\bar{\mathbf{k}}_M(n) = \mathbf{S}_{MM}^T \begin{bmatrix} \mathbf{E}_{pp}(n)^{-1} \boldsymbol{\epsilon}_p(n)' \\ \mathbf{k}_N(n) + \mathbf{A}_{Np}(n) \mathbf{E}_{pp}(n)^{-1} \boldsymbol{\epsilon}_p(n)' \end{bmatrix} \quad (4.8.7)$$

$$\mathbf{Q}_{MM} \bar{\mathbf{k}}_M(n) = \begin{bmatrix} \mathbf{m}_N(n) \\ \boldsymbol{\mu}_p(n) \end{bmatrix} \quad (4.8.8)$$

$$\boldsymbol{\eta}_p(n) = \boldsymbol{\rho}_p(n) + \mathbf{D}_{Np}^H(n-1) \mathbf{x}_N(n) \quad (4.8.9)$$

$$\mathbf{D}_{Np}(n) = [\mathbf{D}_{Np}(n-1) - \mathbf{m}_N(n) \boldsymbol{\eta}_p^H(n)] [\mathbf{I}_{pp} - \boldsymbol{\mu}_p(n) \boldsymbol{\eta}_p^H(n)] \quad (4.8.10)$$

$$\mathbf{k}_N(n+1) = \mathbf{m}_N(n) - \mathbf{D}_{Np}(n) \boldsymbol{\mu}_p(n) \quad (4.8.11)$$

$$\hat{\mathbf{d}}(n) = \mathbf{C}_N^H(n) \mathbf{x}_N(n) \quad (4.8.12)$$

$$\mathbf{e}(n) = \mathbf{d}(n) - \hat{\mathbf{d}}(n) \quad (4.8.13)$$

$$\mathbf{C}_N(n+1) = \mathbf{C}_N(n) + \mathbf{k}_N(n) \mathbf{e}^*(n) \quad (4.8.14)$$

The performance of the fast Kalman DFE is the same as the Kalman DFE, but the computational requirement is $20N+5$ arithmetic operations per iteration [8].

4.9 Least Squares Lattice Decision Feedback Equalizer

The least squares lattice decision feedback equalizer (L-DFE) is an adaptive decision feedback equalizer based on the lattice structure. It has the advantages of both lattice equalizers and DFEs. An L-DFE requires less computation, is more stable and has smaller SS-MSE, especially on severely distorted channels. The structure of the L-DFE equalizer is in Figure 4.9-1. N_1-1 is the number of forward stages and N_2-1 is the number of feedback stages, and it is equivalent to a DFE(N_1, N_2). The L-DFE structure is divided into N_1-N_2 scalar stages and N_2-1 two dimensional stages. The scalar stages belong to the forward filter, but the two dimensional stages combine the forward and feedback stages [14]. As a result, the number of the forward stages must be larger than the number of the feedback stages ($N_1 > N_2$) [14].

The operation of an L-DFE is similar to that of the LSALE. In Figure 4.9-

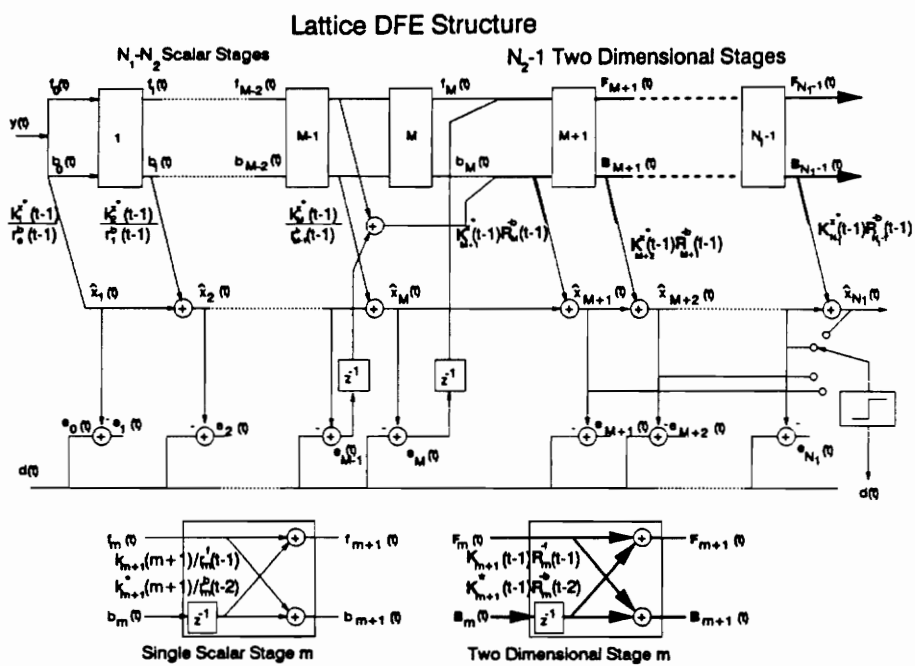


Figure 4.9-1. The structure of a lattice decision feedback equalizer

1, $k_{m+1}(t-1)/r_m^f(t-1)$ and $k_{m+1}^*(t-1)/r_m^b(t-2)$ are the reflection coefficients of a scalar stage, and $K_{m+1}(t-1)R_m^f(t-1)$ and $K_{m+1}^*(t-1)R_m^b(t-2)$, which are two dimensional vectors, are the reflection coefficients of a two dimensional stage. $y(t)$ is the channel output that is applied into the equalizer. $\hat{x}_i(t)$ is the output of the lattice equalizer at different stages and each of them can be chosen as the output of the equalizer. The value of $e_i(t)$ is the prediction error if $\hat{x}_i(t)$ is used as the output of the equalizer, so the size of the equalizer i can be selected at which the local time average of $|e_i(t)|^2$ is the minimum [14]. $d(t)$ is the decision of the symbol value based on $\hat{x}_i(t)$.

The algorithm used to update lattice DFEs, was derived by Ling and Proakis in [12,13,14], and the algorithm and the variables are discussed in more detail in these papers.

Initialization:

$$k_M^b(0)=0$$

$$k_M^x(0)=0$$

$$r_m^f(0)=r_m^b(0)=\delta \quad \delta \text{ is small positive number which is much smaller than 1, } m=1 \text{ to } N_1-N_2-1$$

$$k_m(0)=k_m^x(0)=0$$

$$K_m(0)=K_m^x(0)=0$$

$$R_m^f(0)=R_m^b(0)=\delta I \quad m=N_1-N_2 \text{ to } N_1$$

$$b_0(t)=f_0(t)=y(t)$$

$$r_0^f(t)=r_0^b(t)=wr_0^f(t-1)+|y(t)|^2$$

$$e_0(t)=x(t)$$

$$\hat{x}_0(t)=0$$

$$\alpha_0(t)=1 \quad (4.9.1)$$

Scalar Stage:

($0 < m \leq N_1 - N_2$) unless specified

$$f_m(t) = f_{m-1}(t) - \frac{k_m^*(t-1)b_{m-1}(t-1)}{r_{m-1}^b(t-2)} \quad (4.9.2)$$

$$b_m(t) = b_{m-1}(t-1) - k_m(t-1)f_{m-1}(t)/r_{m-1}^f(t-1) \quad (4.9.3)$$

$$k_m(t) = \lambda k_m(t-1) + \alpha_{m-1}(t-1)f_{m-1}^*(t)b_{m-1}(t-1) \quad (4.9.4)$$

$$r_m^f(t) = r_{m-1}^f(t) - \frac{|k_m(t)|^2}{r_{m-1}^b(t-1)} \quad (0 < m < N_1 - N_2) \quad (4.9.5)$$

$$r_m^b(t) = r_{m-1}^b(t-1) - \frac{|k_m(t)|^2}{r_{m-1}^f(t)} \quad (0 < m < N_1 - N_2) \quad (4.9.6)$$

$$\hat{x}_m(t) = \hat{x}_{m-1}(t) + \frac{k_m^X(t-1)b_{m-1}(t)}{r_{m-1}^b(t-1)} \quad (4.9.7)$$

$$\alpha_m(t) = \alpha_{m-1}(t) - \frac{|b_{m-1}(t)\alpha_{m-1}(t)|^2}{r_{m-1}^b(t)} \quad (4.9.8)$$

$$e_m(t) = d_m(t) - \hat{x}_m(t) \quad (4.9.9)$$

$$k_m^X(t) = \lambda k_m^X(t-1) + \alpha_{m-1}(t)e_{m-1}^*(t)b_{m-1}(t) \quad (4.9.10)$$

Transitional Stage:

$$\hat{b}_m(t) = e_{M-1}(t-1) - \frac{k_m^b(t-1)f_{M-1}(t)}{r_{M-1}^f(t-1)} \quad (4.9.11)$$

$$k_M^b(t) = \lambda k_M^b(t-1) + \alpha_{M-1}(t-1)f_{M-1}^*(t-1)e_{M-1}(t-1) \quad (4.9.12)$$

$$F_M(t) = [f_M(t), e_M(t-1)]^T \quad (4.9.13)$$

$$\mathbf{B}_M(t)=[\mathbf{b}_M(t), \hat{\mathbf{b}}_M(t)]^T \quad (4.9.14)$$

$$\mathbf{R}_M^f(t)=\lambda\mathbf{R}_M^f(t-1)+\alpha_M(t-1)\mathbf{F}_M(t)\mathbf{F}_M^H(t) \quad (4.9.15)$$

$$\mathbf{R}_M^b(t)=\lambda\mathbf{R}_M^b(t-1)+\alpha_M(t)\mathbf{B}_M(t)\mathbf{B}_M^H(t) \quad (4.9.16)$$

Two Dimensional Stage:

$(N_1-N_2 < m < N_1)$ unless specified

$$\mathbf{F}_m(t)=\mathbf{F}_{m-1}(t)-\mathbf{k}_m^H(t-1)[\mathbf{R}_{m-1}^b(t-2)]^{-1}\mathbf{B}_{m-1}(t-1) \quad (4.9.17)$$

$$\mathbf{B}_m(t)=\mathbf{B}_{m-1}(t-1)-\mathbf{k}_m(t-1)[\mathbf{R}_{m-1}^f(t-1)]^{-1}\mathbf{F}_{m-1}(t) \quad (4.9.18)$$

$$\mathbf{k}_m(t)=\lambda\mathbf{k}_m(t-1)+\alpha_{m-1}(t-1)\mathbf{B}_{m-1}(t-1)\mathbf{F}_{m-1}^H(t) \quad (4.9.19)$$

$$\mathbf{R}_m^f(t)=\lambda\mathbf{R}_m^f(t-1)+\alpha_m(t-1)\mathbf{F}_m(t)\mathbf{F}_m^H(t) \quad (4.9.20)$$

$$\mathbf{R}_m^b(t)=\lambda\mathbf{R}_m^b(t-1)+\alpha_m(t)\mathbf{B}_m(t)\mathbf{B}_m^H(t) \quad (4.9.21)$$

$$\alpha_m(t)=\alpha_{m-1}(t)-\alpha_{m-1}^2(t)\mathbf{B}_{m-1}^H(t)[\mathbf{R}_{m-1}^b(t)]^{-1}\mathbf{B}_{m-1}(t) \quad (4.9.22)$$

$$\hat{\mathbf{x}}_m(t)=\hat{\mathbf{x}}_{m-1}(t)+\mathbf{k}_{m-1}^X(t)[\mathbf{R}_{m-1}^b(t-1)]^{-1}\mathbf{B}_{m-1}(t) \quad (N_1-N_2 < m \leq N_1) \quad (4.9.23)$$

$$\mathbf{e}_m(t)=\mathbf{d}(t)-\hat{\mathbf{x}}(t) \quad (4.9.24)$$

$$\mathbf{k}_m^X(t)=\lambda\mathbf{k}_m^X(t-1)+\alpha_{m-1}(t)\mathbf{B}_{m-1}(t)\mathbf{e}_{m-1}^*(t) \quad (N_1-N_2 < m \leq N_1) \quad (4.9.25)$$

This algorithm needs $18N_1+39N_2-39$ arithmetic operations per iteration [14]. The performance of L-DFE is similar to that of Kalman DFE, and is much better than LSALE when the eigenvalue disparity ratio of the channel data is large. This is especially true when there are nulls in the channel spectrum because it uses the DFE structure. However, this L-DFE structure has a disadvantage that $M=N_1-N_2$ must be larger than zero, which means that the number of the forward stages

must be larger than the number of the feedback stages. This requirement limits the flexibility of the equalizer.

4.10 Summary of Algorithms

Table 4.10-1 shows the computational requirements of different algorithms, and it also concludes the advantages and disadvantages of each algorithm. All the RLS algorithms have similar convergence and tracking performances, which are much better than an LMS algorithm. However, these RLS algorithms usually have high computational requirement and complex program structures. Also, some RLS algorithms tend to be unstable. FTF requires the least computation among the RLS algorithms, and it can use a rescue variable to avoid the instability, so it is a good choice. Lattice equalizers are more stable and their structures are flexible, so they are worth considering as possible choices.

4.11 Numerical Example

The following is an example of the operation of an RLS algorithm. The same channel used in Section 3.6 for the LMS algorithm is used here. An LTE with 3 taps are applied as the equalizer structure.

If an RLS algorithm is used to update the LTE, (4.2.2) is utilized to obtain the new tap-gain vector

$$\mathbf{R}_{33}(n)\mathbf{w}_3(n)=\mathbf{p}_3(n) \quad (4.2.2)$$

and the updates for $\mathbf{R}_{33}(n)$ and $\mathbf{p}_3(n)$ are from (4.2.6) and (4.2.16):

$$\mathbf{R}_{33}(n)=\lambda \mathbf{R}_{33}(n-1)+\mathbf{x}_3(n)\mathbf{x}_3^T(n) \quad (4.2.6)$$

$$\mathbf{p}_3(n)=\lambda \mathbf{p}_3(n-1)+d(n) \mathbf{x}_3(n) \quad (4.2.16)$$

Although different methods can be used to update $\mathbf{w}_3(n)$ instead of (4.2.2) as discussed in the sections, they have the same results.

Initial values:

Table 4.10-1. Computational requirement, advantages and disadvantages for different LMS and RLS algorithms.

Algorithm	Number of operations	Advantages	Disadvantages
LMS gradient	$2N+1$	Low computation requirement, simply programmed.	Slow convergence, Poor tracking.
Kalman	$2.5N^2+4.5N$	Fast convergence, good tracking ability.	High computation requirements, unstable.
FTF	$7N+14$	Fast convergence, good tracking ability, low computation requirement.	Complex programing unstable(but can use rescue method)
Gradient lattice	$13N-8$	Stable, low computation flexible structure.	Performance not as good as other RLS, complex programing
LSALE	$19N-12$	Fast convergence, good tracking, stable, flexible structure.	Complex programing computation not low.
Fast Kalman DFE	$20N+5$	Can be used for DFE, fast convergence and good for tracking.	Complex programing computation not low unstable.
L-DFE	$18N_1+39N_2-39$	Fast convergence, good tracking, stable, flexible structure, used for DFE	Complex programing high computation.

$$\mathbf{w}_3(0)=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{R}_{33}(0)=\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}, \quad \mathbf{p}_3(0)=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{x}_3(0)=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \lambda=0.9 \quad (4.11.1)$$

The equalizer operates in the same way as the LMS equalizer in Section 3.6, but (4.2.2) is used to update tap-gain vector.

$$y(1)=1, x(1)=1, \mathbf{x}_3^T(1)=[1,0,0], \hat{d}(1)=\mathbf{w}_3^T(0)\mathbf{x}_3(1)=0, d(1)=y(1)=1, e(1)=1$$

$y(n)$ is the n th input of the channel, $x(n)$ is the n th output of the channel, $\hat{d}(n)$ is the output of the equalizer, $d(n)$ is the decision made upon the n th bit value and $e(n)$ is the prediction error.

In each iteration, first $\mathbf{R}_{33}(n)$ is updated using (4.2.6) and $\mathbf{p}_3(n)$ is updated using (4.2.16), and then Equation (4.2.2) is set up. The optimal tap-gain vector $\mathbf{w}_3(n)$ is obtained by solving Equation (4.2.2).

Iteration 1: Using (4.2.6) and (4.2.16) to update $\mathbf{R}_{NN}(1)$ and $\mathbf{p}_N(1)$, Equation (4.2.2) can be written as

$$\begin{bmatrix} 1.009 & 0 & 0 \\ 0 & 0.009 & 0 \\ 0 & 0 & 0.009 \end{bmatrix} \begin{bmatrix} w_1(1) \\ w_2(1) \\ w_3(1) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \Rightarrow \begin{bmatrix} w_1(1) \\ w_2(1) \\ w_3(1) \end{bmatrix} = \begin{bmatrix} 0.9911 \\ 0 \\ 0 \end{bmatrix} \quad (4.11.2)$$

$$y(2)=-1, x(2)=-0.4, \mathbf{x}_3^T(2)=[-0.4,1,0], d(2)=y(2)=-1, \\ \hat{d}(2)=\mathbf{w}_3^T(1)\mathbf{x}_3(2)=-0.3964, e(2)=-0.6036$$

Iteration 2:

$$\begin{bmatrix} 1.0681 & -0.4 & 0 \\ -0.4 & 1.0081 & 0 \\ 0 & 0 & 0.0081 \end{bmatrix} \begin{bmatrix} w_1(2) \\ w_2(2) \\ w_3(2) \end{bmatrix} = \begin{bmatrix} 1.3 \\ -1 \\ 0 \end{bmatrix}, \Rightarrow \begin{bmatrix} w_1(2) \\ w_2(2) \\ w_3(2) \end{bmatrix} = \begin{bmatrix} 0.9932 \\ -0.5979 \\ 0 \end{bmatrix} \quad (4.11.3)$$

$$y(3)=-1, x(3)=-1.24, \mathbf{x}_3^T(3)=[-1.24, -0.4, 1], d(3)=y(3)=-1, \\ \hat{d}(3)=\mathbf{w}_3^T(2)\mathbf{x}_3(3)=-0.9985, e(3)=-0.0015$$

After the second iteration, one can find the tap-gain vector is already very close to the optimal value in (3.6.4), and the prediction error is about 0.001. It clearly shows than an RLS algorithm converges extremely fast compared to the converging process of the LMS in Section 3.6.

Iteration 3:

$$\begin{bmatrix} 2.4989 & 0.136 & -1.24 \\ 0.136 & 1.0673 & -0.4 \\ -1.24 & -0.4 & 1.0065 \end{bmatrix} \begin{bmatrix} \mathbf{w}_1(3) \\ \mathbf{w}_2(3) \\ \mathbf{w}_3(3) \end{bmatrix} = \begin{bmatrix} -2.41 \\ -0.5 \\ -1 \end{bmatrix}, \Rightarrow \begin{bmatrix} \mathbf{w}_1(3) \\ \mathbf{w}_2(3) \\ \mathbf{w}_3(3) \end{bmatrix} = \begin{bmatrix} 0.9933 \\ -0.5978 \\ 0.0076 \end{bmatrix} \quad (4.11.4)$$

$$x(4)=1, y(4)=0.256, \mathbf{x}_3^T(4)=[0.256, -1.24, -0.4], d(4)=x(4)=1, \\ \hat{d}(4)=\mathbf{w}_3^T(3)\mathbf{x}_3(4)=1.002, e(4)=-0.002;$$

Iteration 4:

$$\begin{bmatrix} 2.3145 & -0.1950 & -1.2184 \\ -0.1950 & 2.4985 & 0.136 \\ -1.2184 & 0.136 & 1.0659 \end{bmatrix} \begin{bmatrix} \mathbf{w}_1(4) \\ \mathbf{w}_2(4) \\ \mathbf{w}_3(4) \end{bmatrix} = \begin{bmatrix} 2.425 \\ -1.69 \\ -1.3 \end{bmatrix}, \Rightarrow \begin{bmatrix} \mathbf{w}_1(4) \\ \mathbf{w}_2(4) \\ \mathbf{w}_3(4) \end{bmatrix} = \begin{bmatrix} 0.9930 \\ -0.5985 \\ -0.8239 \end{bmatrix} \quad (4.11.5)$$

$$x(5)=1, y(5)=1.154, \mathbf{x}_3^T(4)=[1.154, 0.256, -1.24], d(5)=x(5)=1, \\ \hat{d}(5)=\mathbf{w}_3^T(4)\mathbf{x}_3(5)=0.9883, e=0.0117;$$

This example shows the way an RLS algorithm works and proves that its fast convergence. In this channel, the equalizer converges in 2 iterations .

5. Maximum Likelihood Sequence Estimation

5.1 Introduction

Several adaptive equalizers have been developed to mitigate the intersymbol interference. Among them, the maximum-likelihood sequence estimation (MLSE) has a distinct approach. Using a channel impulse response simulator for the channel, the MLSE tests all possible data sequences, and chooses the one with the maximum probability as the output. An MLSE usually has a large computational requirement, especially when the delay spread of the channel is large. Using the MLSE as an equalizer was first proposed by Forney in [15] in which he set up a basic MLSE estimator structure and implemented it with the Viterbi algorithm. Ungerboeck [16] and Magee and Proakis [17] made the estimator adaptive so that it can be used in unknown and time-varying channels. Magee and Proakis proposed a structure with a fixed matched filter and an adaptive channel channel impulse response estimator that is similar to a linear equalizer (see Figure 5.1-1). Ungerboeck suggested that both the matched filter and the channel impulse response estimator should be adaptive (see Figure 5.1-2).

5.2 Channel Impulse Response Estimator

Similar to a linear equalizer, a channel impulse response estimator (CIRE) can be constructed from a tapped-delay-line (TDL) filter. An ideal adaptive equalizer contains the inverse transfer function of a channel (see Section 2.2), but a CIRE tries to achieve the same transfer function as the channel. The structure of a CIRE is shown in Figure 5.2-1(b), and it is compared with a linear transversal equalizer in Figure 5.2-1(a).

Since both the channel and CIRE have the same input signal sequence, if they have the same output, the CIRE is guaranteed to have the same impulse response as the channel.

Let h_k be the impulse response of a channel and α_k be a data sequence, then the output sequence z_k of a channel is

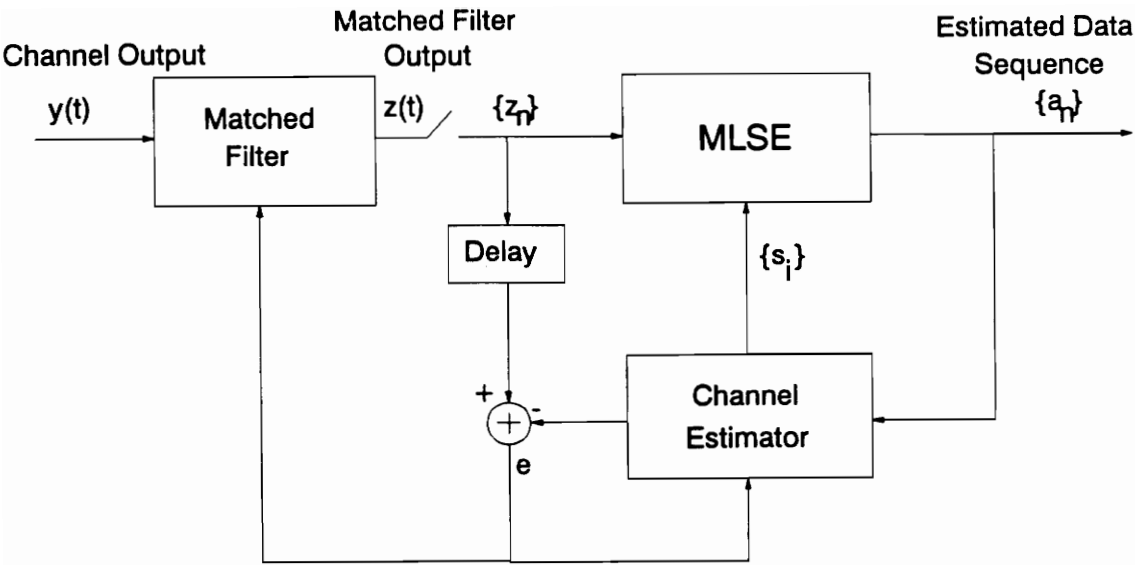


Figure 5.1-1. The structure of a maximum likelihood sequence estimator with an adaptive matched filter

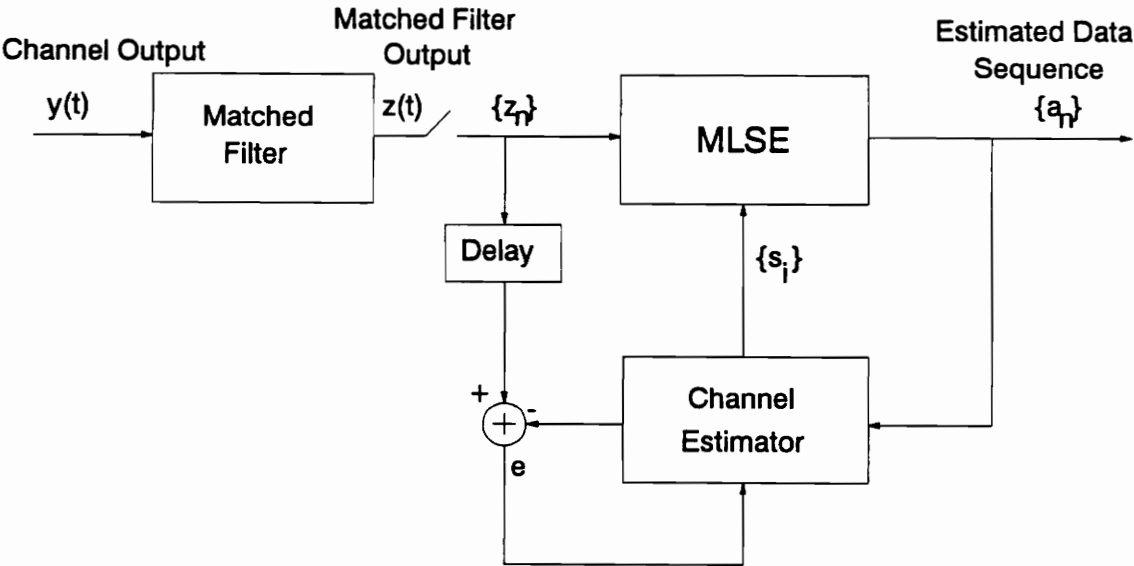


Figure 5.1-2. The structure of a maximum likelihood sequence estimator with a non-adaptive matched filter

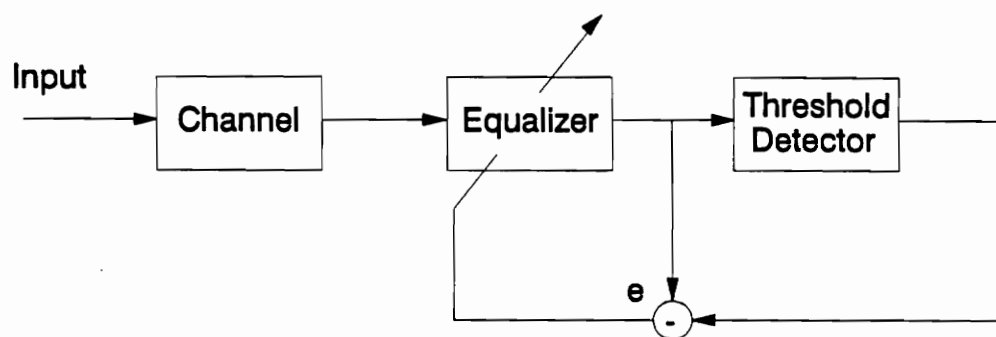


Figure 5.2-1(a). Implementation of an adaptive equalizer

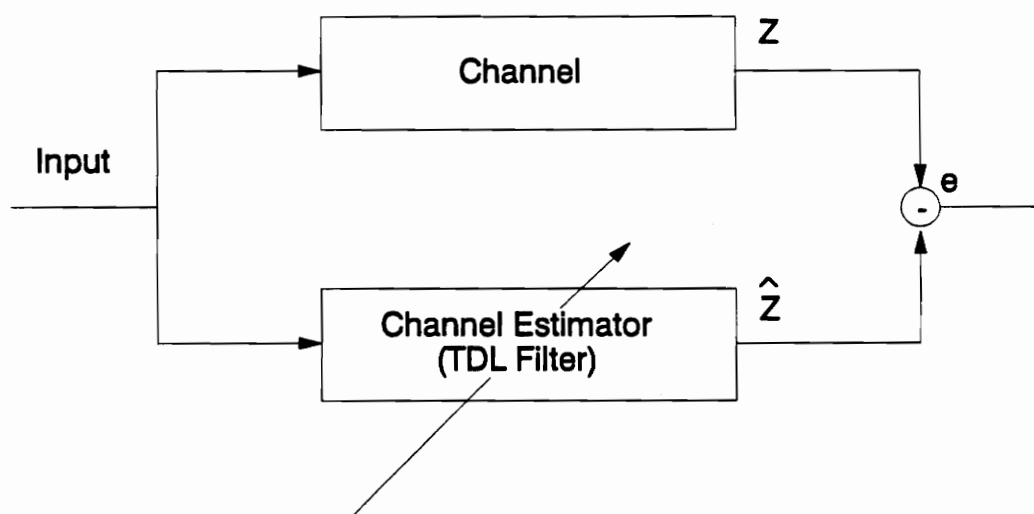


Figure 5.2-1(b). Implementation of a channel impulse response estimator

$$z_k = h_k * \alpha_k \quad (5.2.1)$$

The CIRE uses the same signal sequence as the input

$$\hat{z}_k = \hat{h}_k * \hat{\alpha}_k \quad (5.2.2)$$

where $\hat{\alpha}_k$ can be either the original transmitted signal sequence α_k , or the estimated data sequence, $\hat{h}_k$ is the impulse response of the CIRE or tap-gains, and $\hat{z}_k$ is the output of the CIRE.

To make $\hat{h}_k$ close to h_k , $E[|z_k - \hat{z}_k|^2]$ has to be minimized. If $E[|z_k - \hat{z}_k|^2] = 0$, $h_k = \hat{h}_k$; therefore, the CIRE has the same impulse response as the channel. Because there exists noise, residual ISI, and channel tracking error, the perfect match cannot be obtained. If $E[|z_k - \hat{z}_k|^2]$ is small enough, the CIRE is very close to the channel. To work in unknown and time-varying channels, the CIRE updates itself using adaptive algorithms, such as an LMS and RLS algorithm. The error between outputs of the channel and CIRE, $e_k = z_k - \hat{z}_k$, is used to update the tap-gains.

5.3 Matched Filter

The matched filter used in the MLSE should be an adaptive filter, which mitigates the ISI induced by a frequency selective fading channel. The matched filter also whitens colored noise, so that the sampled noise components are independent identically distributed Gaussian variables [15,34].

Let $h(t)$ be the impulse response of a channel and the matched filter implemented in the transmitter and $w(t)$ be the noise. The autocorrelation function of the noise is

$$W(\tau) = E[w^*(t)w(t+\tau)] = N_0 K(\tau) \quad (5.3.1)$$

If $w(t)$ is white Gaussian noise, $K(\tau) = \delta(\tau)$ [16]. Ideally, the impulse response of the matched filter is

$$g_{MF}(t) \simeq h^*(-t) * K^{-1}(t) \quad (5.3.2)$$

where $h^*(-t)$ is the matched filter of $h(t)$ and $K^{-1}(t)*K(t)=\delta(t)$. Therefore, the matched filter can mitigate ISI and whiten noise [16]. Then the CIRE will estimate the combination of the channel and matched filters, which is [16]

$$s(t) \simeq g_{MF}(t)*h(t) \quad (5.3.3)$$

However, due to the variation of the channel impulse response, it is impossible to design a fixed ideal matched filter. An adaptive matched filter is one way to solve this problem. Similar to an adaptive equalizer, it consists of a TDL filter that has adaptive tap-gains using an LMS or RLS algorithm.

A matched filter and the CIRE should be adaptive in a mobile channel. However, for fast changing channels, the adaptation of the filters has to be fast enough to track the channel, so an RLS algorithm is usually needed. RLS algorithms need at least $7N$ operations per iteration, where N is the number of taps including taps of the matched filter and the CIRE. For example, if a 3-tap adaptive matched filter and 5-tap CIRE are used, and the FTF algorithm is used to update them, $N=8$. The number of operations needed is about $7 \times (5+3) + 14 = 70$ per iteration. Suppose the symbol rate of the system is 24.3 ksp/s, a processor needs to have 1.7 million instructions per second (MIPS). Remember that the Viterbi decoder needs a lot of computation too; therefore, a very powerful DSP has to be used. Hence, in order to do the job in real time, N must be small.

Obviously, a receiver with an adaptive matched filter is much more complex in both its structure and computation. In high data-rate communication systems, the computation requirement of an adaptive equalizer is vital to its usefulness, because the equalizer has to be updated every symbol duration. A higher symbol rate means a smaller symbol duration, and thus a higher computational requirement per unit time.

A CIRE is capable of eliminating the problem caused by ISI, so an adaptive matched filter does not have to be used. On the other hand, correlated noise does not cause notable degradation in MLSE [16]. Hence, it is possible to

use a fixed filter and an adaptive CIRE without losing performance [16,17].

5.4 Maximum Likelihood Sequence Estimation (Viterbi Decoder):

The Viterbi decoder uses the channel information from the CIRE and the signals out of the matched filter to estimate the data sequences. Suppose the impulse responses of a channel and the CIRE are exactly the same. After a data sequence is transmitted through the channel, an output signal sequence is received at the Viterbi decoder. The Viterbi decoder tries all the possible input data sequences, convolves them with the CIRE, and then compares outputs of the CIRE with the output of the channel. Since the channel and the CIRE have the same impulse response, only the sequence exactly the same as the information sequence has the same output as the channel. Therefore, the data sequence can be detected. Since a perfect CIRE usually cannot be obtained due to residual ISI and noise, the most likely sequence according to a specific criterion is chosen as the estimated sequence.

If an adaptive matched filter is used at the receiver, the channel output that is supplied in the matched filter is

$$y(t) = h(n) * a_n \quad (5.4.1a)$$

where $h(n)$ is the channel impulse response and a_n is the transmitted data. Then, the output of the matched filter is

$$\begin{aligned} z_n &= g_{MF}(t) * y(t) |_{t=nT} = h^*(-n) * K^{-1} * h(n) * a_n \\ &= s_n * a_n \end{aligned} \quad (5.4.1b)$$

where s_l is the l th tap-gain of the CIRE, a_n is transmitted data sequence, z_n is the output of the matched filter, and g_{MF} is the impulse response of the matched filter at the receiver. From (5.4.1b) the impulse response of CIRE is

$$s_l = g_{MF}(t) * h(t) |_{t=lT} = h^*(-t) * K^{-1} * h(t) |_{t=nT} = s_l^* \quad (5.4.2)$$

According to [15], the sequence that maximizes the metric

$$J_n(\{\alpha_n\}) = \sum_{i=1}^n 2\text{Re}(\alpha_i z_i) - \sum_{i=1}^n \sum_{k=1}^n \alpha_i s_{i-k} \alpha_k \quad (5.4.3)$$

is the most likely sequence, where α_i is the possible data sequence, z_i is the matched filter's output that is applied into the CIRE, s_i is a tap-gain of the CIRE. The metric $J_n(\{\alpha_n\})$ tries all the possible data sequences $\{\alpha_n\}$ and chooses the one that maximizes $J_n(\{\alpha_n\})$ as the output of the MLSE.

5.5 Viterbi Algorithm

The obstacle of implementing the MLSE is its huge computational requirement that grows exponentially with the number of states per sample and the length of a block of estimation. Let an information sequence be $\{\sigma_n\} = \{\alpha_{n-N+1}, \alpha_{n-L}, \dots, \alpha_n\}$, where α_i is i th symbol. There are M possible values of α_i , so there are N^M possible $\{\sigma_n\}$, where N is the length of a block to be estimated; therefore, N^M probabilities need be computed to get the maximum likely sequence.

The Viterbi algorithm, originally designed to decode convolutional codes, can be used in the MLSE to avoid the huge computation. The Viterbi algorithm is best described by a trellis diagram, as shown in Figure 5.5-1. In Figure 5.5-1, 0 and 1 inside a circle is the choice of a state, and a line leaves a state and goes to another state of the next symbol. Solid lines represent the paths that are the possible maximum likely data sequence, while dotted lines are the discarded paths. The numbers above circles are the $J\{\}$ values, and the numbers above the lines are the extra $J\{\}$ values contributed by the path it located. These numbers are used to help explain how the Viterbi algorithm works.

Suppose all the symbol values before the first symbol of the block are known. The first symbol α_1 has two possible choices, 0 and 1, so J_1 has two values $J_1\{0\} = -0.9$ and $J_1\{1\} = -0.2$ computed using (5.4.3). A decision on the value of α_1 will not be made immediately on J_1 .

After obtaining both J_1 values, J_2 values are computed using (5.4.3) based on J_1 and the value of the second symbol, α_2 , which also has two choices, 0 and 1. First, let $\alpha_2 = 0$ and recall that the previous symbol α_1 has two choices, then the

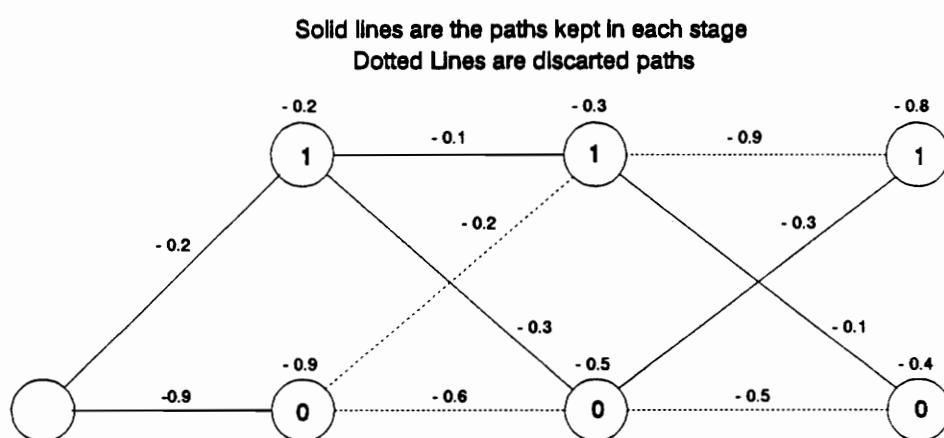


Figure 5.5-1. Implementation of the Viterbi algorithm

possible choices of $\{\alpha_1, \alpha_2\}$ are $\{0,0\}$ and $\{1,0\}$. Using (5.4.3), the $J_2\{0,0\}=-0.9-0.6=-1.5$ and $J_2\{1,0\}=-0.2-0.3=-0.5$. $\{1,0\}$ is selected because of its larger J_2 value, and both its path and its J value are kept. The path $\{0,0\}$ is discarded. The reason the smaller J value and its path can be ignored is that when choosing the third symbol, it can be guaranteed that $J_3(0,0,\alpha_3) < J_3(1,0,\alpha_3)$ no matter what α_3 is and so with subsequent J_n values. Thus, $\{\alpha_1, \alpha_2\}=\{0,0\}$ can never be part of the most likely sequence. Then, let $\alpha_2=1$, $J\{0,1\}=-0.9-0.2=-1.1$ and $J\{1,1\}=-0.2-0.1=-0.3$, so $\{1,1\}$ is kept as the possible path, and $J\{0,1\}$ is discarded. After that, let $\alpha_3=0$, $J\{1,1,0\}=-0.3-0.1=-0.4$, $J\{1,0,0\}=-0.5-0.5=-1$, so $\{1,1,0\}$ is selected. Let $\alpha_3=1$, $J\{1,1,1\}=-0.3-0.9=-1.2$, $J\{1,0,1\}=-0.5-0.3=-0.8$, so $\{1,0,1\}$ is chosen.

After this step, the power of the Viterbi algorithm can be seen. That is, at the first step instead of 4 possible paths, only 2 are kept. One is $\{\alpha_1, 0\}$, and the other is $\{\alpha'_1, 1\}$. Then, for the third symbol, only 4 possible sequences need to be evaluated instead of $2^3=8$, and 2 of them are kept as further evaluation. For the n th symbol, there are still only 4 possible sequences instead of 2^n . After comparing the J value of the 4 sequences, two choices are left for further evaluation. The Viterbi algorithm obviously saves a lot of computations without any loss of performance.

After the whole block with N symbols is evaluated, there are only two possible data sequence paths are left, and the one with the larger J value is chosen and its path is kept as the output of the estimator.

5.6 MLSE with A Fixed Matched Filter

The previous example is only suitable for an MLSE estimator with an adaptive matched filter. Recall that a non-adaptive matched filter also can be used for mobile radio communication due to its simplicity in structure and computation. If a fixed matched filter is used, there is an easy and straight forward method to evaluate the probability of a sequence using mean squared error.

The mean squared error plays an important role in adaptive equalization, and it can also be used in the MLSE estimation. Setting the J value

$$J_n\{\alpha_n, \alpha_{n-1}, \dots, \alpha_{n-M+1}\} = - \sum_{k=n-M+1}^n |z_k - \sum_{i=0}^L s_i^* \alpha_{k-i}|^2 \quad (5.6.1)$$

where z_k is the k th received signal, s_i is the tap-gain of the CIRE, α_i is the possible symbol value at time i , M is the length of data block to be estimated, n is the first point of the block and L is the length of the channel estimator. (5.6.1) sums the square of

$$e = (z_k - \sum_{i=0}^L s_i^* \alpha_{k-i}) \quad (5.6.2)$$

which is the error between z_k , the output of the channel and $\sum_{i=0}^L s_i^* \alpha_{k-i}$, the output of the CIRE. Assuming the CIRE has the same impulse response as the channel, the sequence that is the same as the input of the channel has the minimum cumulative squared error $J_n\{\}$. Therefore, if $J_n\{\alpha_N\}$ is the minimum, the data sequence α_N is the most likely data sequence. The Viterbi algorithm can be used in the same way to find the data sequence which minimizes the J value.

5.7 Summary

The MLSE has two kinds of structures. The one with a fixed adaptive matched filter is used because of its simple structure and computation. The adaptive channel impulse response estimator can be updated by an LMS or RLS algorithm, and the RLS algorithm is recommended because of its fast convergence. The Viterbi algorithm is used to estimate the data sequence, and it effectively reduces the computational requirement of the MLSE.

6. Equalization for Stationary Channels

Although adaptive equalizers usually work in time-varying environments, stationary channels are useful for evaluating many properties of adaptive equalizers, such as the convergence rate and the steady-state mean squared error (SS-MSE). By using two stationary channels, the tracking capability can also be evaluated. After the equalizer converges to the first channel, the second channel is used to observe how the adaptive equalizer tracks the variation.

6.1. Stationary Channel simulation

In order to evaluate an adaptive equalizer, frequency-selective channels are needed. There are several methods to simulate frequency-selective channels, such as using a hardware simulator or measured data. But, a software channel model is easy and inexpensive to design, control, and implement in computer simulation.

An FIR filter is used to simulate the channel that has the impulse response

$$f(n) = \sum_{n=0}^{N-1} c_n \delta(t-nT) \quad (6.1.1)$$

where N is the number of taps in the FIR filter, and $(N-1)T$ is the time span of the channel impulse response. There are N multipath components in this channel assuming $c_n \neq 0$. This channel can be set to be stationary or time-varying by choosing the c_n 's as constants or variables. After the tap-gains c_n 's are decided, a random data sequence $x(n) = \pm 1$, is generated and passed through the channel. Then, the output of a channel with ISI is generated.

To simulate the channel output with a specific signal-to-noise ratio (SNR), some Gaussian noise is added at the output of the channel. A function generating random Gaussian noise with zero mean and variance $\sigma^2 = 1$ is used where σ^2 is the noise power. The noise is multiplied with the expected σ , and then it is added to the output of the channel. In order to generate a specific SNR, the signal power at the receiver has to be quantified. Assume the channel input is ± 1 , the input power is unity. The channel power gain is

$$G_c = \sum_{n=0}^{N-1} |c_n|^2 \quad (6.1.2)$$

Because the signal power at the channel output is G_c and the noise power is σ^2 , the SNR is

$$\text{SNR} = \frac{G_c}{\sigma^2} \quad (6.1.3)$$

(6.1.3) can be used to decide the SNR value, or choose σ if an SNR value is given. In order to simplify the simulation, the channel power loss is usually set to 0 dB ($G_c=1$), then Equation (6.1.3) becomes

$$\text{SNR} = \frac{1}{\sigma^2} \quad (6.1.4)$$

Therefore, the SNR is independent of the channel used if channel power loss is normalized.

To analyze the characteristics of adaptive equalizers, it is important to choose the proper channels that represent the different power spectral densities. One of the simplest channel models which is called channel A in this thesis, is

$$(A). \begin{aligned} c(iT) &= (1 + \cos(2(1-i)\pi/w)) / (1 + 2\cos^2(2\pi/w)) \quad \text{for } i=0,1,2; \\ c(t) &= 0, \quad \text{otherwise} \end{aligned} \quad (6.1.5)$$

where T is the symbol duration, and w is a variable used to control the characteristics of the channel. $w=3.1$ is chosen in channel A. Then, eigenvalue disparity ratio of the channel correlation matrix $\mathbf{R}_{NN}$ is 11. When $w=3.3$, the eigenvalue disparity ratio is 21 [10]. The rms delay spread of channel A is 0.369 T computed using (2.1.2) and (2.1.3).

Ling and Proakis used another channel in their work [12,13,14], which is called channel B

$$(B). \begin{aligned} c(0) &= 0.408, \quad c(T) = 0.816, \quad c(2T) = 0.408. \\ c(t) &= 0 \quad \text{otherwise} \end{aligned} \quad (6.1.6)$$

Since there is a null in its spectrum, the eigenvalue disparity ratio is infinite. This channel model is good for analyzing the performance of the equalizers on channels with severe distortion. The rms delay spread of channel B is $0.574T$.

These two discrete-time channel models are used in the following simulations. The spectra of the two channels are given in Figure 6.1-1 which shows that channel A is not quite as severely distorted as channel B , which has a null in its spectrum. To have a channel with a power loss of 0 dB, the summation of the power coefficients of different multipath components is set to unity; hence, one may choose the σ of the white Gaussian noise using (6.1.4) to get the desired SNR.

6.2. Equalization Simulation

The equalizers and algorithms simulated in this thesis for stationary channels are

- Least mean square linear transversal equalizers (LMS-LTE),
- Least mean square decision feedback equalizers (LMS-DFE),
- Gradient lattice equalizers (GALE),
- Least squares lattice equalizers (LSALE),
- Least squares lattice decision feedback equalizers (L-DFE),
- RLS Kalman linear transversal equalizers (Kalman-LTE),
- RLS Kalman decision feedback equalizers (Kalman-DFE),
- RLS fast transversal filter equalizers (FTF),
- RLS fast Kalman decision feedback equalizers (fast Kalman DFE).

The sizes of the equalizers are variable in order to fit into channels with different delay spreads.

To examine the performance of all these equalizers, channel A and B above are tested by each equalizer and plots of MSE versus the number of iterations are used to examine the convergence performance and SS-MSE. Different-sized equalizers are used, as well as different step-sizes α for the LMS and weighting factors λ for the RLS. To save computation time, a specified small number of iterations (e.g. 10) are executed, and then the MSE results of these iterations are computed. Computing the MSE for the following periods in the same way, the curves of convergence can be plotted. The MSE results for every 100 iterations

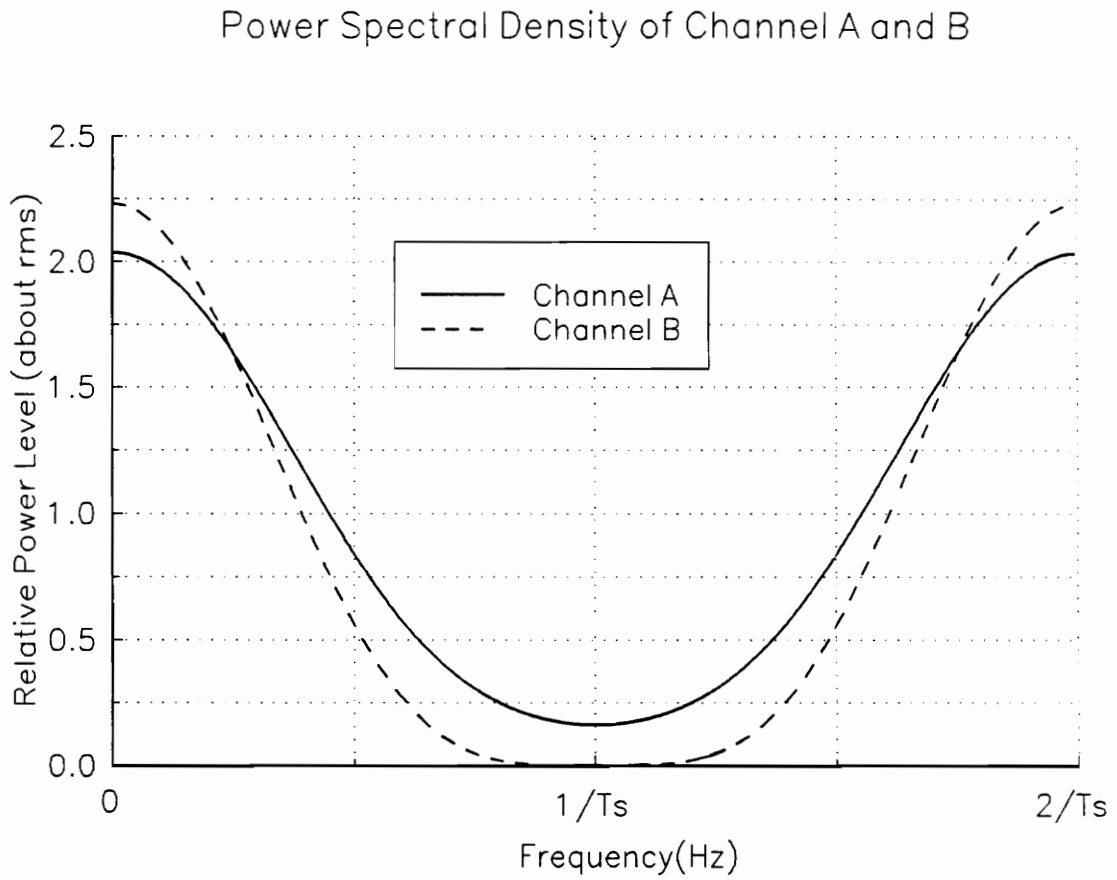


Figure 6.1-1. The power spectral densities of channel A and B

are used to evaluate the LMS equalizers. The MSE results for every 10 iterations are used in the the RLS equalizers, because of their fast convergence. The mean performance plots may also be generated by ensemble-averaging the squared equalization prediction error over a large number of individual learning curves, which will result in a smoother curve. But this method requires more computer time to generate the MSE curve. On the other hand, using local mean has similar results as the ensemble average, and using local mean emphasizes more on the fluctuation of the error; that is, the equalizer prediction error does not always decrease, but it fluctuates in the convergence process.

6.3. Simulation results

6.3.1 Convergence and SS-MSE

The convergence performance of different equalizers along with different adaptive algorithms is evaluated in these simulations in terms of convergence rate and SS-MSE. The simulation results of these equalizers and algorithms for the same channels are compared. Also, the results of using different-sized equalizers and different α and λ values are compared.

The equalizers used have 11 taps, and the DFE uses 6 forward taps and 5 feedback taps, unless specified otherwise. From this point onward, a DFE is denoted as DFE(n, m), where n is the number of forward taps, and m is the number of feedback taps. Only the total number of taps is shown in the parentheses for a non-DFE equalizer. The step-size α for LMS equalizers is chosen to be 0.02, and the weighting factor λ for the RLS algorithms is 0.997 because they will have the similar SS-MSE.

Figure 6.3-1 shows the simulation results for channel *A* by LMS-LTE and LMS-DFE. Both equalizers converge after 600 to 900 iterations. The SS-MSE of the DFE is a bit smaller than the LTE. Since channel *A* has the eigenvalue disparity of 11 dB, the result means the LMS-DFE does not have significant advantages over LMS-LTE in good channels that are not severely distorted.

Figure 6.3-2 presents the simulation results of the two equalizers using LMS algorithms for channel *B*, which is severely distorted. It clearly shows the

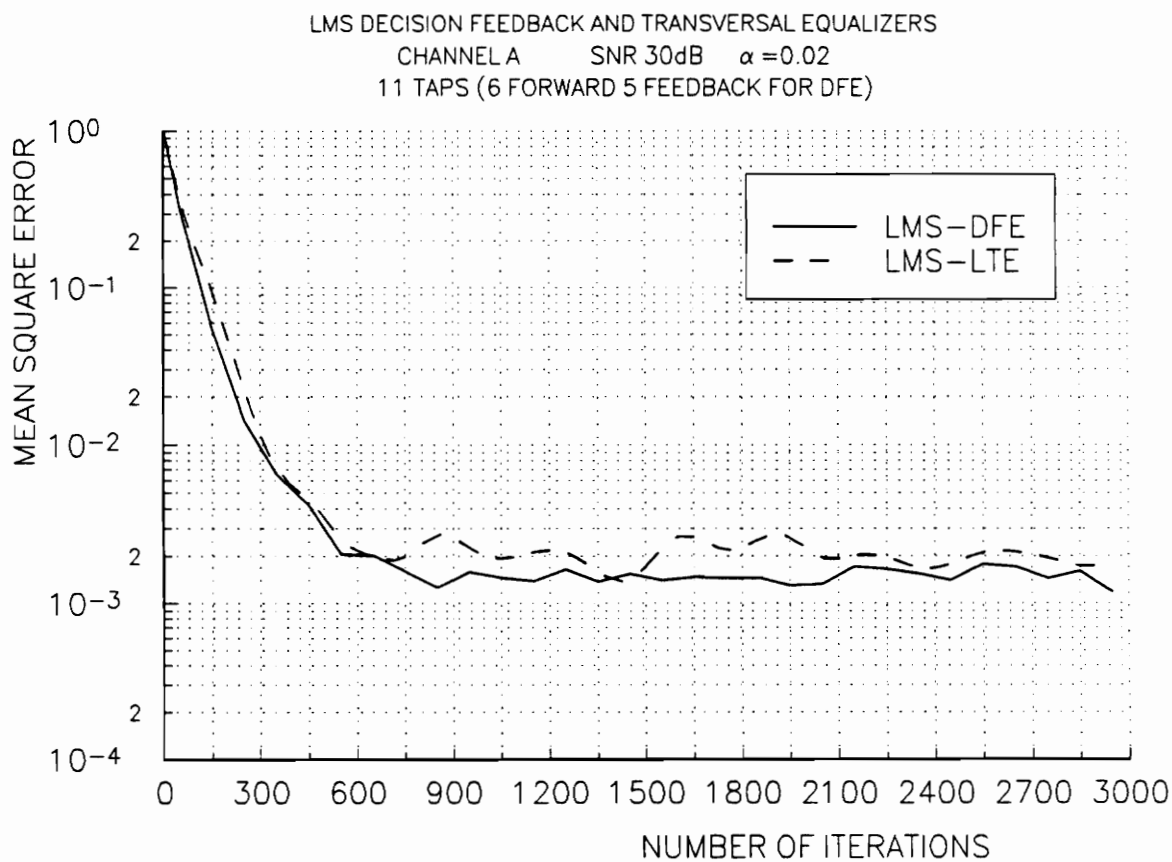


Figure 6.3-1. Convergence of LMS DFE and LMS LTE in channel A.
SNR=30dB, $\alpha=0.02$.

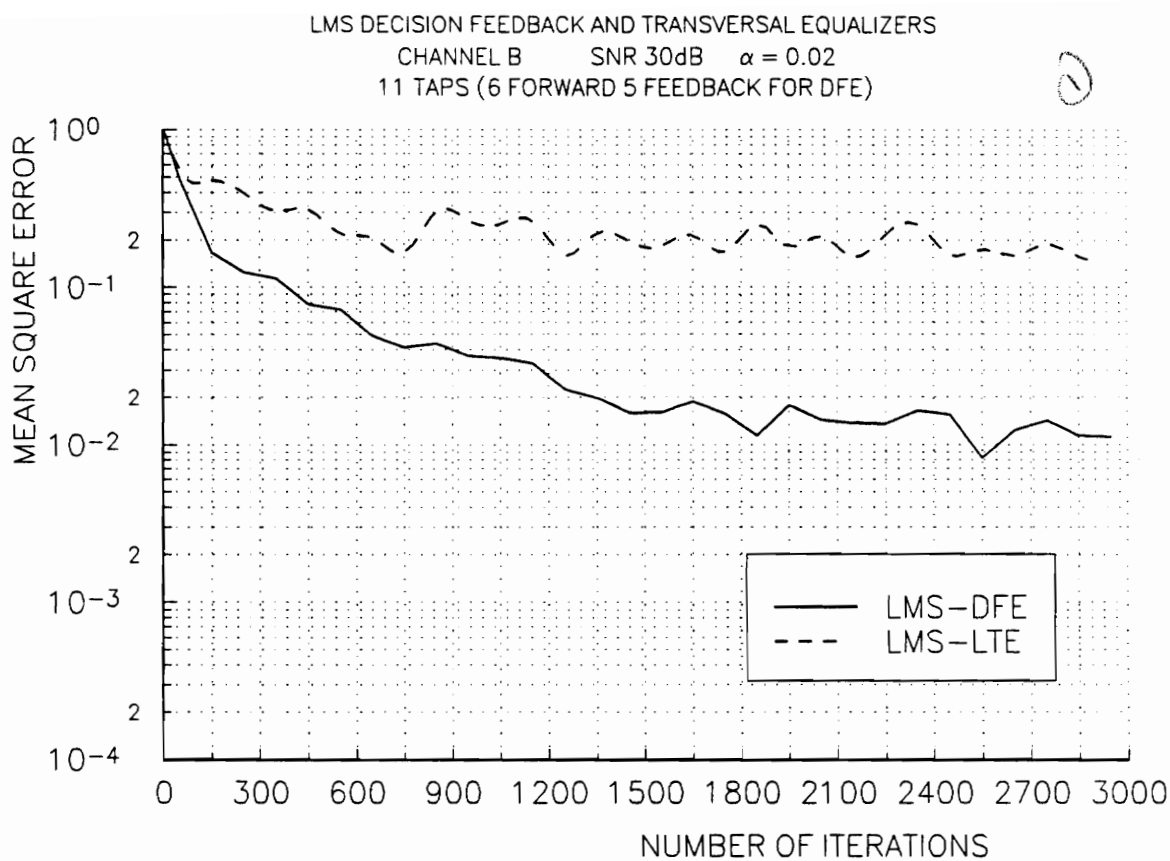


Figure 6.3-2. Convergence of LMS DFE and LMS LTE in channel B. SNR=30dB, $\alpha=0.02$.

advantage of the DFE in that the SS-MSE of LMS-LTE is up to 0.1 which is too large for proper communication [10], while the SS-DFE of LMS-DFE is almost twenty times smaller than that of LMS-LTE. The LMS-DFE in channel B converges at 1800 iterations, which is much slower than in channel A , and the SS-MSE is larger, also. This confirms that the performance of LMS equalizers depends on the eigenvalue disparity ratio of the channel covariance matrix. Even so, a DFE works much better than an LTE when the channel is severely distorted.

In Figure 6.3-3, the performance of LSALE, GALE and L-DFE in channel A is shown. Compared to Figure 6.3-1, all the equalizers in Figure 6.3-3 converge much faster than equalizers using the LMS algorithm. The LSALE and L-DFE converge in about 30 to 60 iterations, and the GALE converges in about 90 iterations. The results show the much faster convergence of these RLS equalizers over the LMS equalizers, but the SS-MSE remains almost the same. As expected, both the convergence rate and SS-MSE of LSALE are a little better than those of GALE. The L-DFE does not show any advantages over LSALE in channel A . In short, Figure 6.3-3 shows RLS equalizers have a very high convergence rate.

Figure 6.3-4 shows the convergence of the three equalizers using channel B . The similarity of the LSALE and GALE remains the same, while the SS-MSE of L-DFE is much better than LSALE and GALE. This shows that the L-DFE works much better in severely distorted channels. These equalizers still converge rapidly. The convergence rate of L-DFE is a little slower than LSALE and the equalizer converges in about 60 iterations. The LSALE converges in about 30 iterations. The reason is that the latter has a larger SS-MSE, so it is easier to converge to the optimal value. Comparing Figure 6.3-4 with Figure 6.3-3, the SS-MSE of L-DFE of channel A is better than channel B ; therefore, the eigenvalue disparity ratio still affects a DFE, although it affects an LTE more severely.

Figure 6.3-5 shows the performances of the Kalman and Kalman DFE using Channel A , and Figure 6.3-6 shows the fast versions of RLS equalizers: FTF and fast Kalman DFE. The fast Kalman algorithm has the same performance as the Kalman algorithm, although it needs less computation. Both equalizers converge in about 30 iterations and their SS-MSE values are the same and equal to that of LMS equalizers and lattice equalizers.

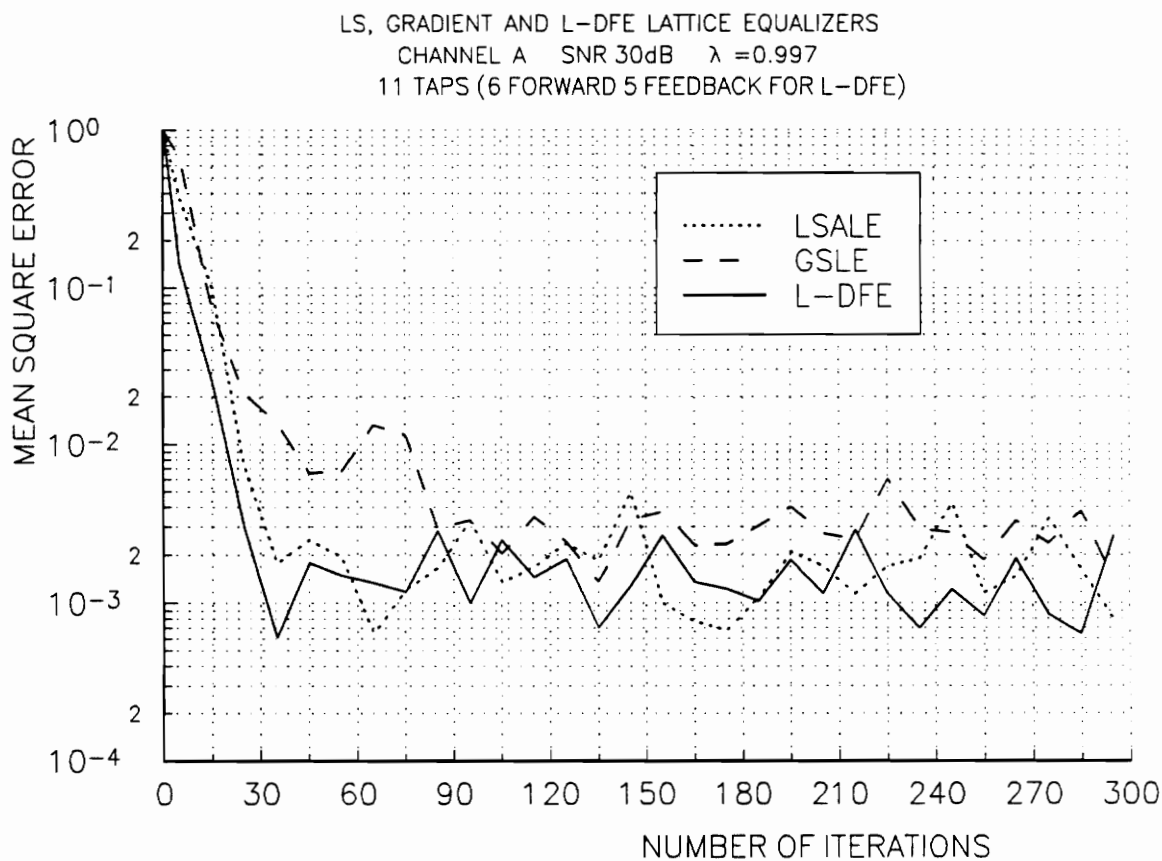


Figure 6.3-3. Convergence of LSALE, GALE and L-DFE in channel A. SNR=30dB, $\lambda=0.997$.

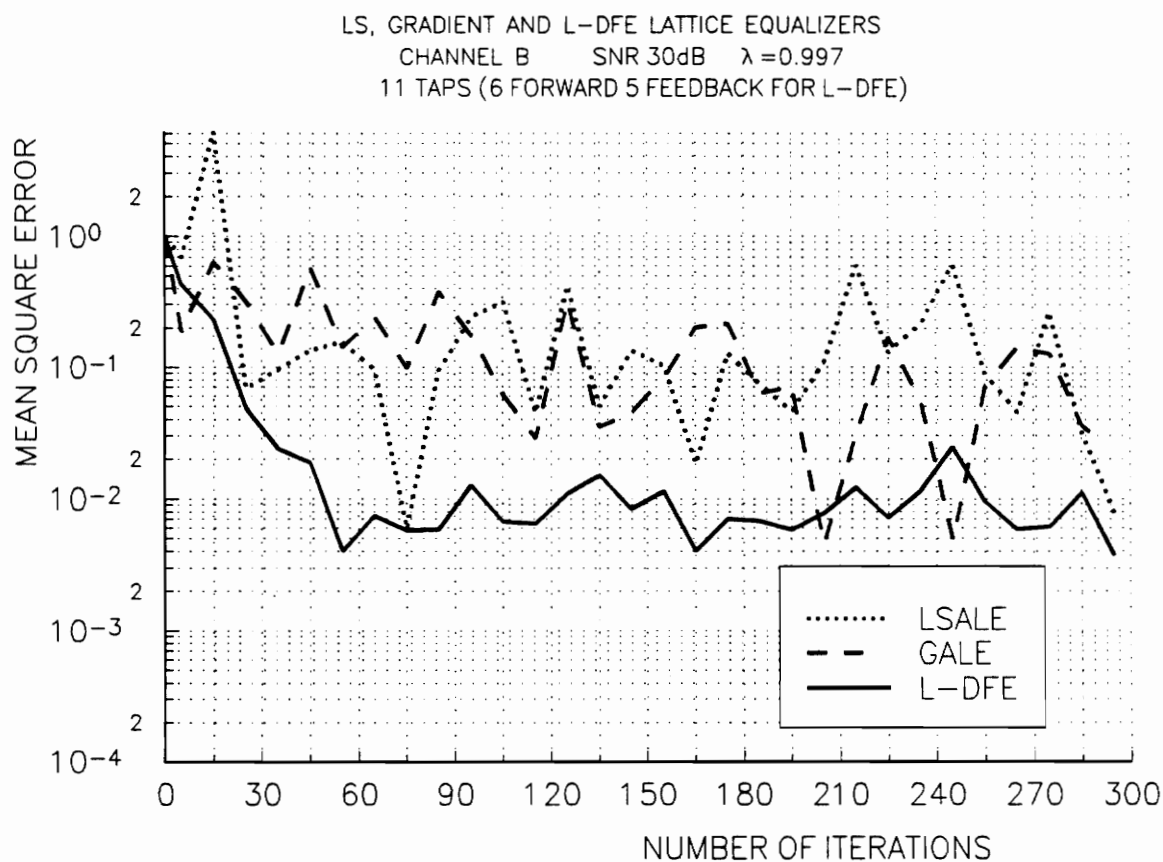


Figure 6.3-4. Convergence of LSALE, GALE and L-DFE in channel B. SNR=30dB, $\lambda=0.997$.

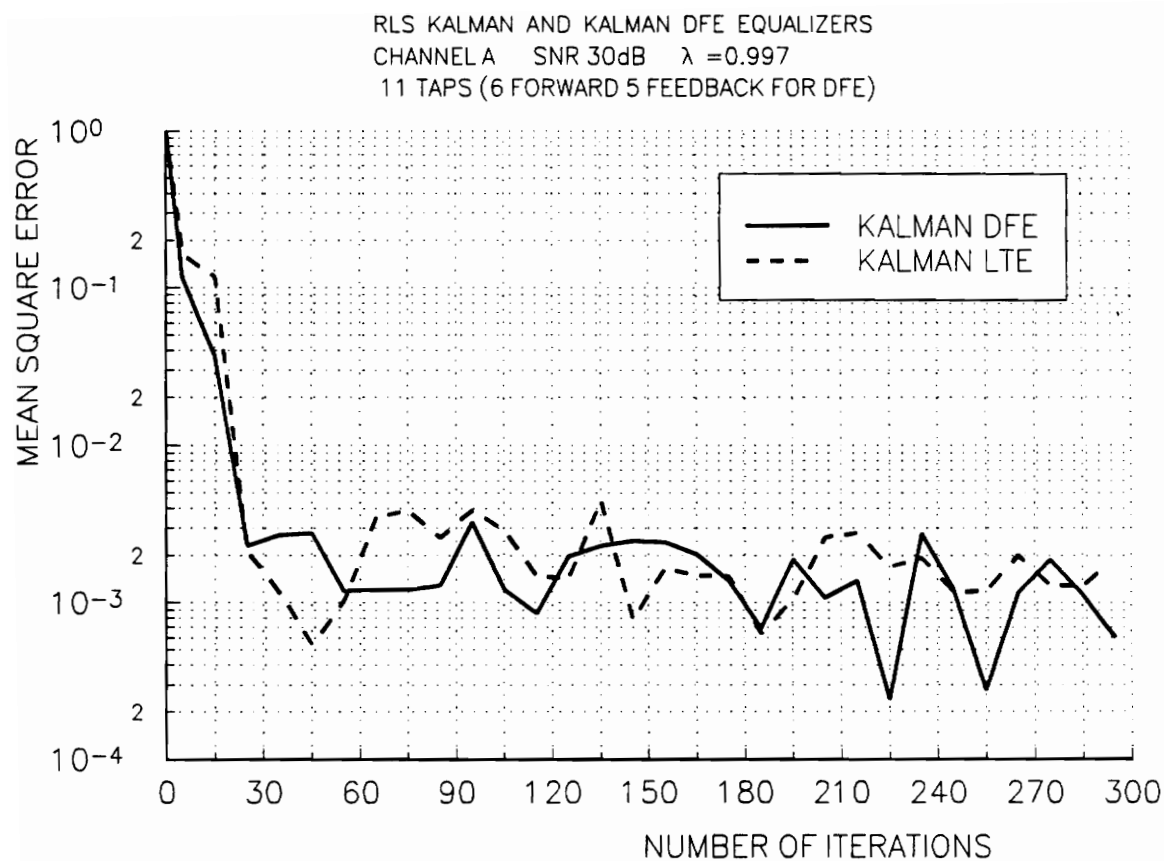


Figure 6.3-5. Convergence of Kalman DFE and Kalman LTE in channel A.
SNR=30dB, $\lambda=0.997$

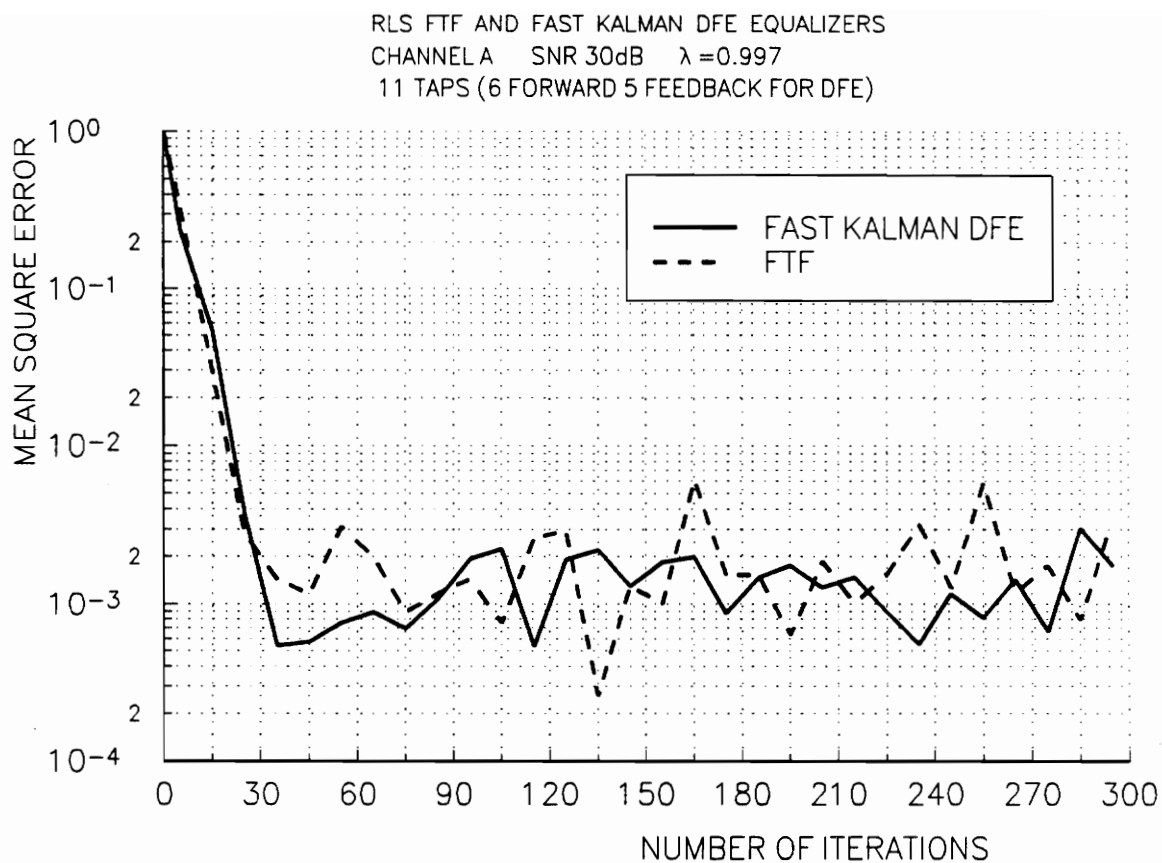


Figure 6.3-6. Convergence of fast Kalman DFE and FTF in channel A.
SNR=30dB, $\lambda=0.997$

In Figures 6.3-7 and 6.3-8, the four RLS equalizers are examined using channel *B*. The results in the two plots are very similar, which further indicates that fast versions of RLS algorithms can work as well as Kalman algorithms. The Kalman DFE has much less SS-MSE than Kalman LTE and FTF, and it converges as fast as the latter. The SS-MSE of the Kalman equalizer in channel *B* are similar to that of LSALE.

From the results shown here, it can be concluded that equalizers with an RLS algorithm converge much faster than those with an LMS algorithm; the DFE structures have much better SS-MSE than the LTE structures when the channel is severely distorted. The channel spectrum disparity affects the performances of both LTE and DFE structures, but the effects on a DFE are not as severe as it is on an LTE.

Now, the effects of step-size, α , in an LMS algorithm and weighting factor, λ , in an RLS algorithm are shown. Figure 6.3-9 illustrates the way α of an LMS algorithm affects the convergence process in channel *A*. A DFE(6,5) was used and SNR=30dB. Four α 's are chosen: 0.005, 0.02, 0.10, 0.15. When $\alpha=0.02$, the DFE converges after about 600 iterations and the SS-MSE is about 2×10^{-3} ; when $\alpha=0.1$, the DFE converges at about 150, and the SS-MSE is 6×10^{-3} . If α is larger than 0.15, the equalizer will not be stable. When $\alpha=0.005$, the DFE does not converge until about 3000 iterations. This convergence comparison method may not be very accurate, because it takes less time to reach larger SS-MSE, so the convergence time should be small if SS-MSE is large. Thus, the trends of convergence are compared, and it is apparent that a larger α causes the convergence curve to become steeper, which means faster convergence, but the SS-MSE after convergence is larger. Therefore, the larger the α , the faster the convergence rate and the larger the SS-MSE.

In Figure 6.3-10, the effect of the λ in RLS algorithms is shown. RLS-DFE was used in channel *A*, and the SNR is 30 dB. In order to get a clear comparison, the same pseudo-random seed was used for different simulations. The results show that the convergence curves of different λ 's are the same at the beginning, which means that λ has no effect on the convergence rate. However, a smaller λ does cause larger SS-MSE, although the difference is not significant.

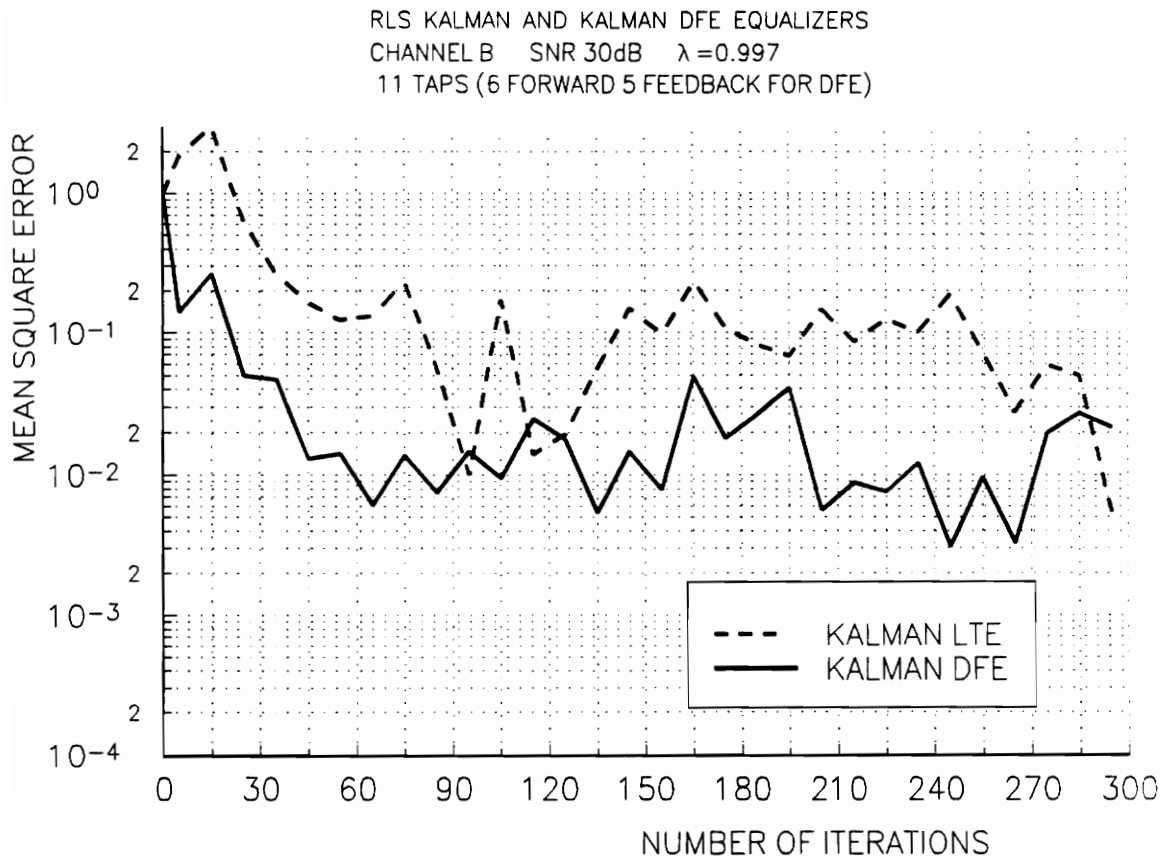


Figure 6.3-7. Convergence of the Kalman LTE and the Kalman DFE in channel B. SNR=30dB, $\lambda=0.997$

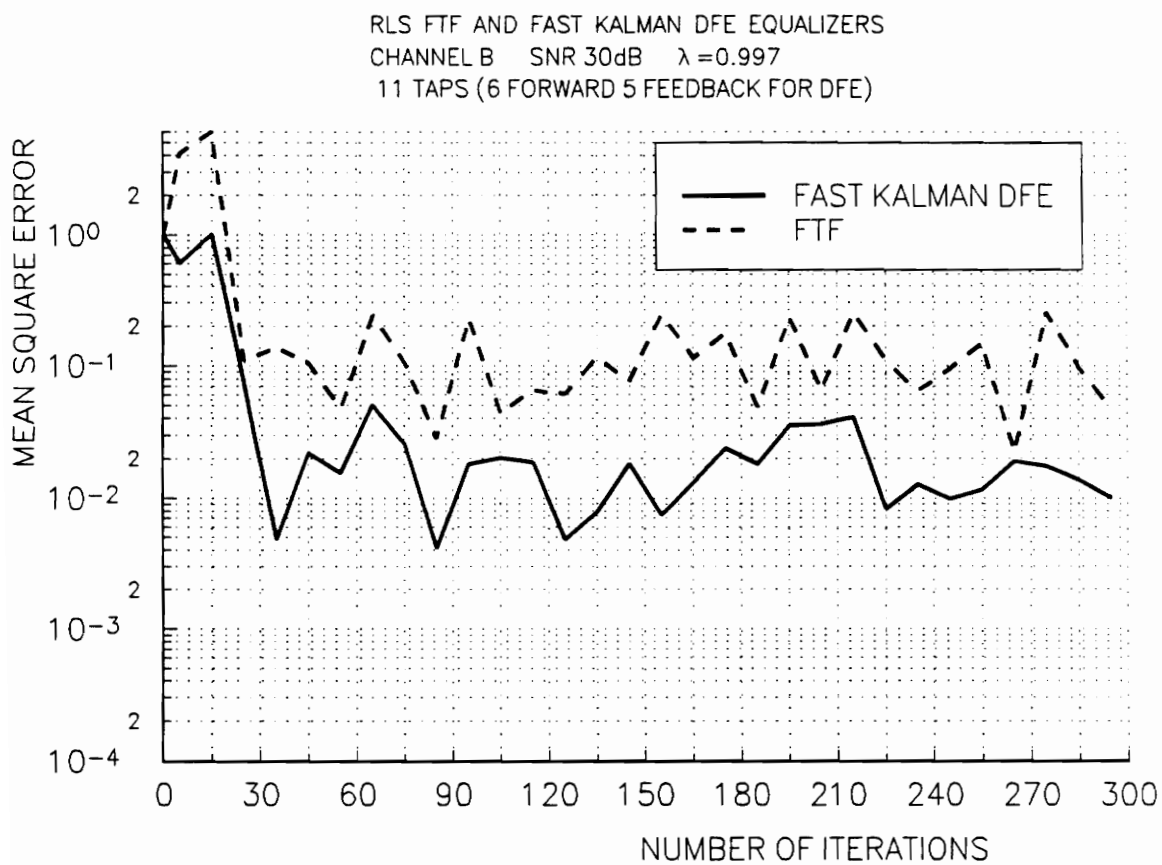


Figure 6.3-8. Convergence of fast Kalman DFE and FTF in channel B.
SNR=30dB, $\lambda=0.997$

2

Convergence of different step-sizes
LMS-DFE(6,5), Channel A

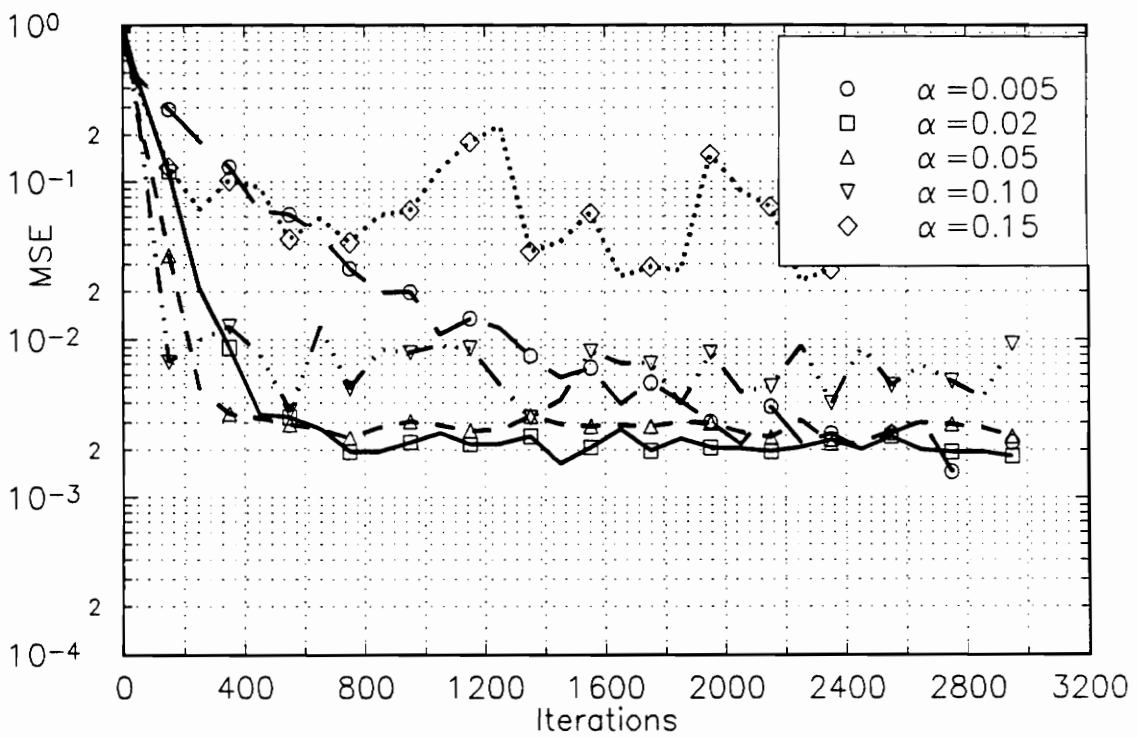


Figure 6.3-9. Convergence of LMS-DFE(6,5) for different α in channel A

Convergence of different forgetting factors
 RLS-DFE(6,5) Channel A, SNR=30 dB

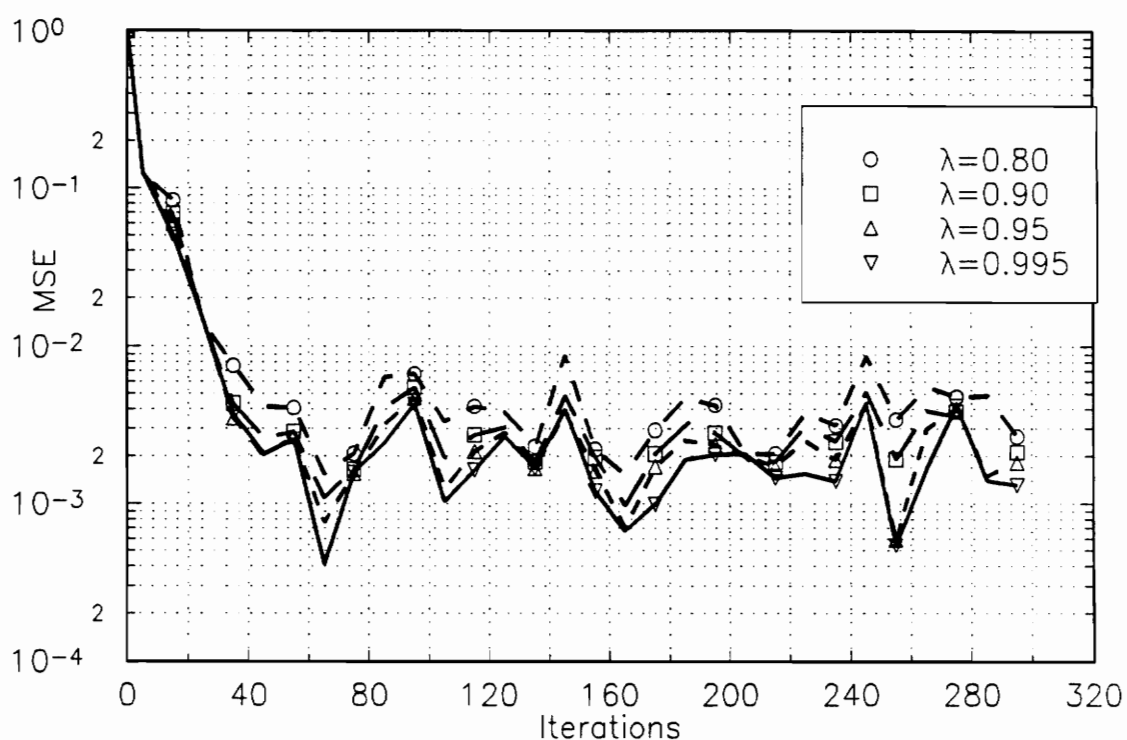


Figure 6.3-10. Convergence of RLS-DFE(6,5) for different λ in channel A.

Figures 6.3-11 and 6.3-12 show that the size of an equalizer does not affect the convergence of either the LMS or the RLS algorithm. In both figures, the convergence curves of DFE(2,2), DFE(4,4) and DFE(9,9) are the almost same, except DFE(9,9) has a little worse SS-MSE than DFE(4,4). For DFE(2,2), its SS-MSE is much larger than for the other two equalizers.

It is easy to understand why DFE(2,2) has a large SS-MSE because a small-sized equalizer cannot accommodate the delay larger than one symbol. Hence, the residual ISI affects the equalizer performance. The reason why DFE(9,9) is poorer than DFE(4,4) is that, if the delay spread is very small compared to the size of the equalizer, many of the extra tap-gains are expected to be zero or very small. However, the algorithm adjusts all the tap-gains so that it is impossible for them to be zero. This generates residual ISI in the equalizer that originally does not exist in the channel.

These results from Figure 6.3-9 to 6.3-12 are obtained from LMS and RLS DFE, but results for an LTE and lattice equalizer will be the similar because of their same mathematical basis.

The results show the importance of the equalizer's size. There is an optimal equalizer depending on the delay spread of the channel, and either smaller or larger than the optimal size will deteriorate the equalization performance. Although a larger sized equalizer does not cause as much deterioration as a smaller one, it wastes computational power.

Table 6.3-1 summarizes the convergence and SS-MSE performance of different equalizers and algorithms.

6.3.2 Tracking Capability

To evaluate the tracking capability of an adaptive equalizer in stationary channels, two stationary channels can be used. After the equalizer converges to the first channel, the channel is changed to another one; then one may observe how the equalizer converges to the new channel. In the following simulation, channel A was used as the first channel with SNR=30dB. The second channel was channel A also, but the amplitude of the channel impulse response was doubled and the SNR kept the same, so equalizers should have the same SS-MSE

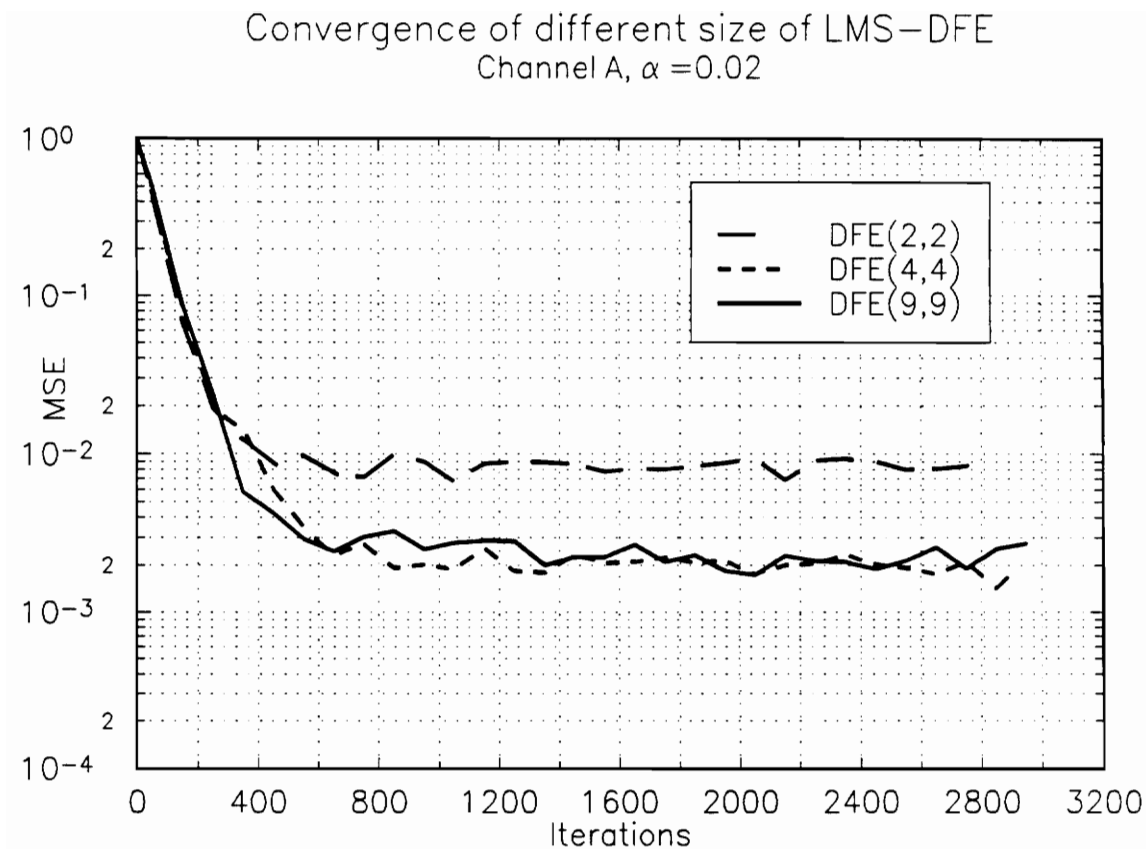


Figure 6.3-11. Convergence of different sized LMS-DFE in channel A, $\alpha=0.02$

Convergence of different sizes of DFE
 RLS-DFE $\lambda = 0.95$, Channel A, SNR 30dB

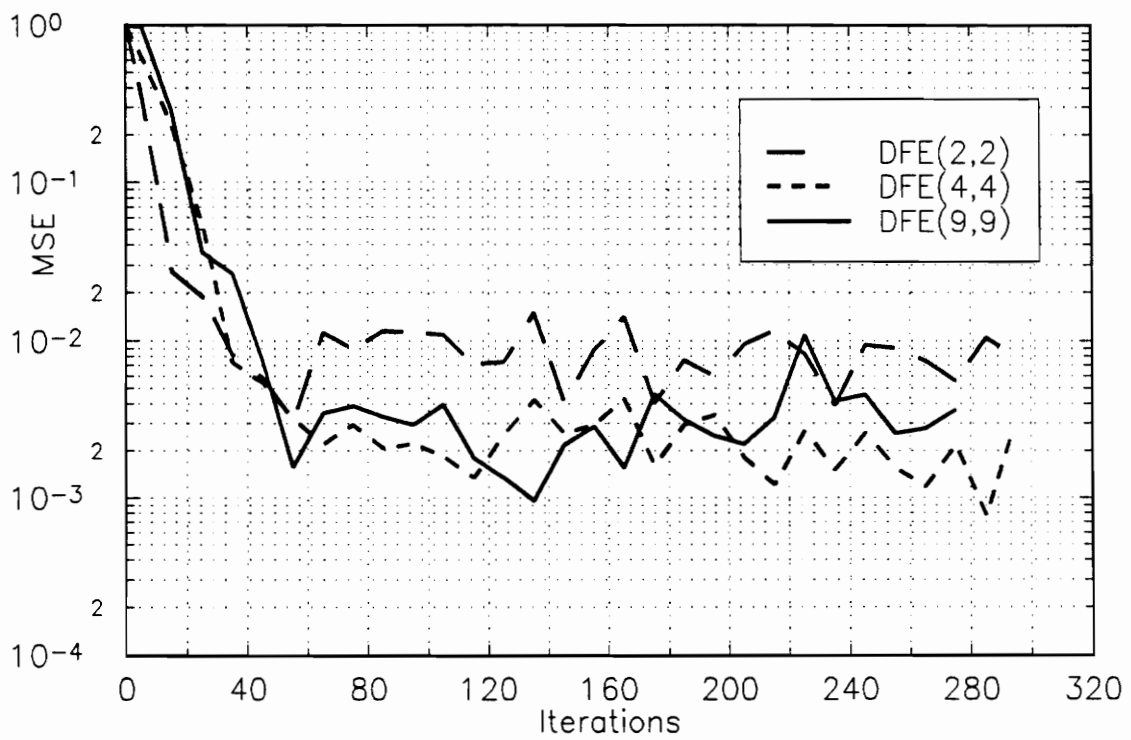


Figure 6.3-12. Convergence of different sized RLS-DFE in channel A, $\lambda=0.95$

Table 6.3-1. Summary of convergence rates and SS-MSE of different equalizers and structures.

Algorithms	Convergence Rate		Steady-State MSE	
	Channel A	Channel B	Channel A	Channel B
LMS LTE	600	600	2×10^{-3}	2×10^{-1}
LMS DFE	600	1800	2×10^{-3} ,	1×10^{-2}
Kalman, FTF	30	40	2×10^{-3}	8×10^{-2}
Fast Kalman DFE	30	40	2×10^{-3}	1×10^{-2}
LSALE	30	30	2×10^{-3}	1×10^{-1}
GALE	90	100	2.5×10^{-3}	1×10^{-1}
L-DFE	30	50	2×10^{-3}	8×10^{-3}

in both channels. This method of evaluating tracking capability of an equalizer was proposed by Ling and Proakis in [6].

Figure 6.3-13 shows how α influences the tracking performance of an LMS-DFE(4,4). Values of 0.02 and 0.001 are chosen for α to show the difference in performance. The MSE results of 100 iterations before the change of channels, and the MSE results of 1000 iterations after the change were recorded. Iteration number 0 represents the point at which the channel was changed. The results were the ensemble averages of 300 independent runs. When $\alpha=0.02$, the equalizer can come back to the normal MSE value in about 200 symbols, but if $\alpha=0.001$, it takes much longer to come back. Hence, α affects the tracking capability of a DFE. A larger α improves the tracking ability of an equalizer, but on the other hand, it causes larger SS-MSE. If checking the MSE before the time 0, one will find the MSE for $\alpha=0.001$ is a little smaller than when $\alpha=0.02$, although the difference is very trivial.

Figure 6.3-14 shows how the size of an equalizer affects the tracking. DFE(2,2), DFE(4,4), DFE(9,9) with an LMS algorithm were used, and the three curves have the same rate of decay, so they have the same tracking capability. The results indicate that the size does not have any effect on tracking capability of an LMS-DFE.

Figure 6.3-15 shows how λ of RLS algorithms affects the tracking. The MSE of 30 bits before and 300 bits after the channel was changed was recorded. From the Figure 6.3-15, it is clear that a small λ improves the tracking ability. When $\lambda=0.95$, it takes about 60 to 90 bits for the MSE to recover, and it takes much longer for $\lambda=0.995$. Compared to Figure 6.3-10, which shows that λ has no effect on convergence rate, λ has different effects on convergence and tracking.

Figure 6.3.16 shows how the size of an equalizer using a RLS algorithm affects the performance. DFE(2,2), DFE(4,4) and DFE(9,9) were used. The three curves have the same rate of decay, so the three equalizers have the same tracking capability. The result is the same as that of the LMS algorithm shown in Figure 6.3-11.

In Figure 6.3-16, the RLS return to the steady state in about 60 to 90 bits, while in Figure 6.3-14, the LMS took about 200 to 250 bits. Hence, an RLS

Tracking Performance of Different α
Channel A, LMS-DFE(4,4)

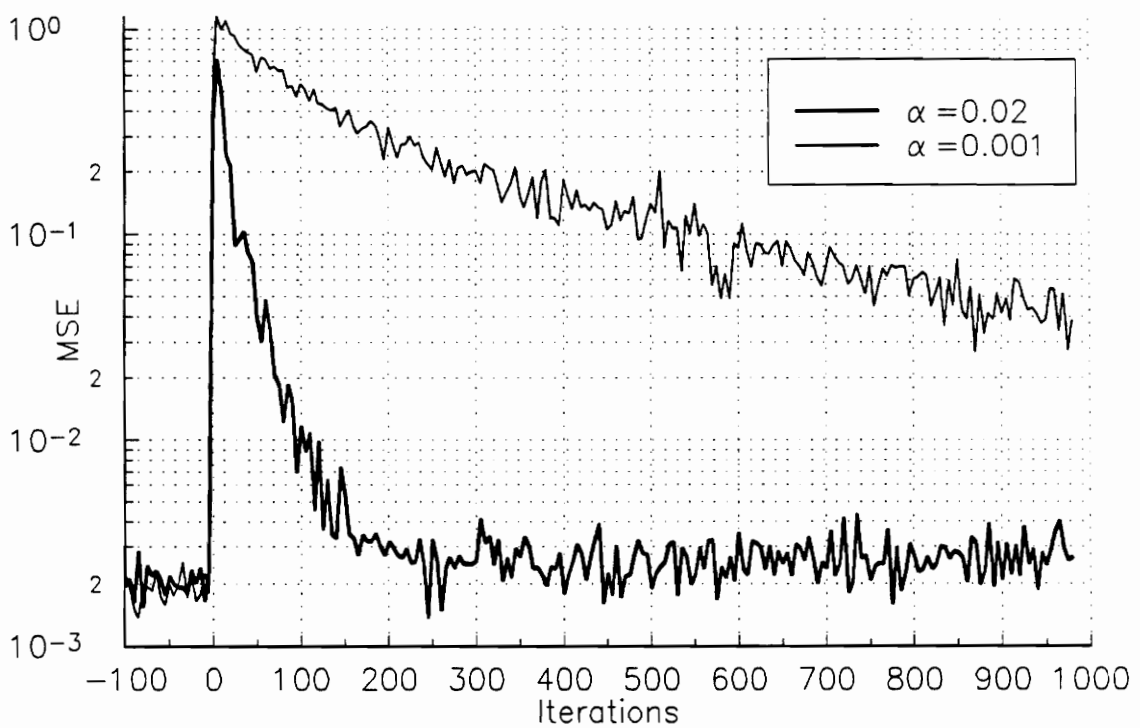


Figure 6.3-13. Tracking performance of LMS-DFE(4,4) with different α in channel A.

Tracking Performance of Different Size DFE
Channel A, $\alpha = 0.02$

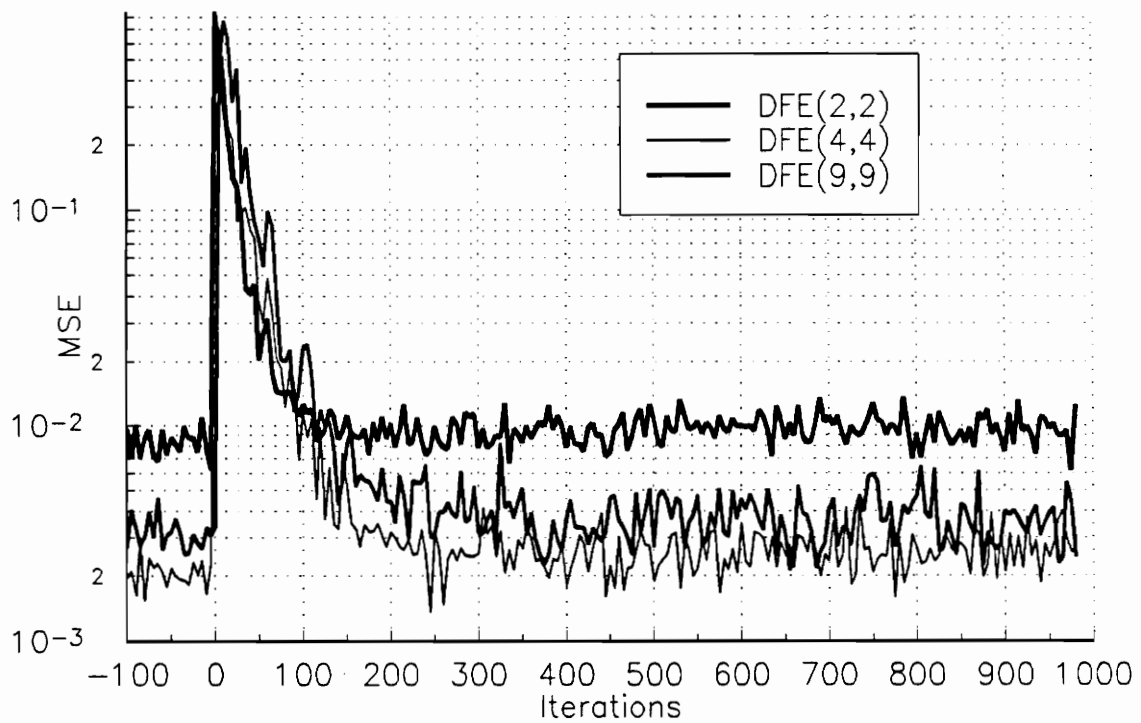


Figure 6.3-14. Tracking performance of different sized LMS-DFE in channel A. $\alpha=0.02$

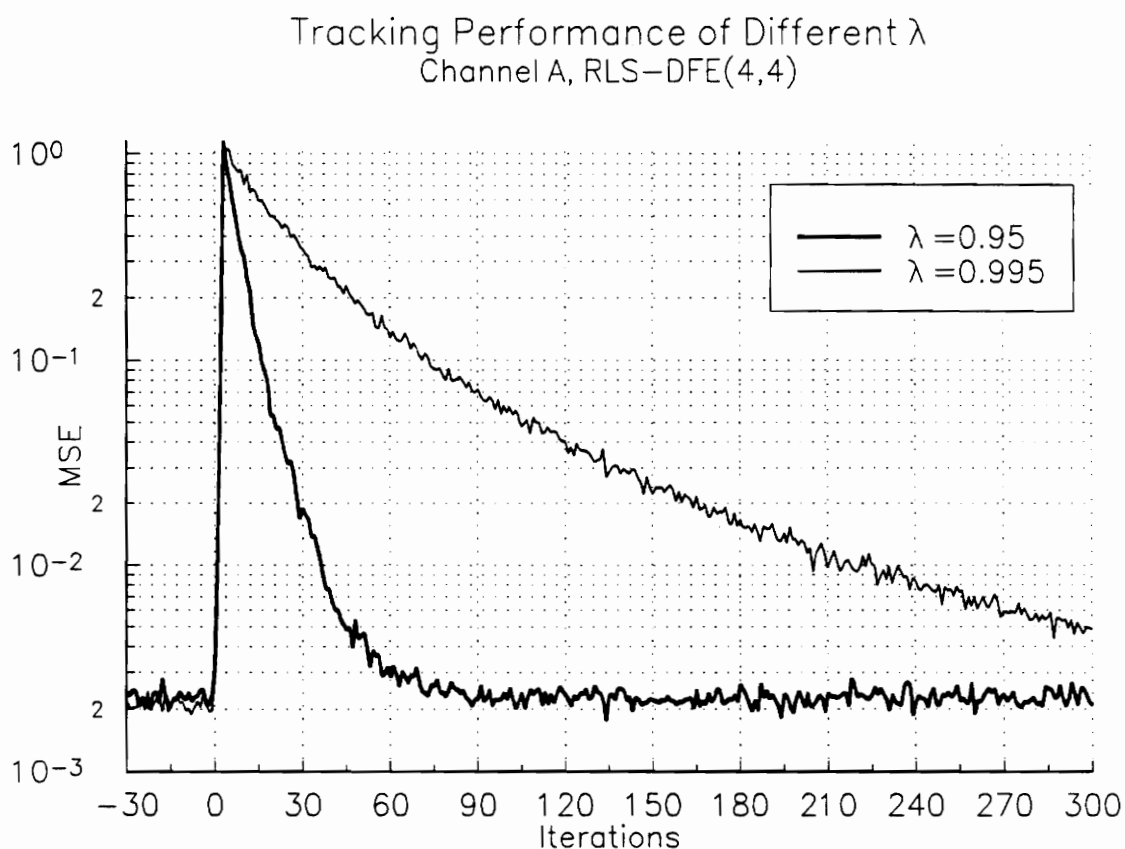


Figure 6.3-15. Tracking performance of RLS-DFE(4,4) with different λ in channel A.

Tracking Performance of Different Size RLS-DFE
Channel A, $\lambda = 0.95$

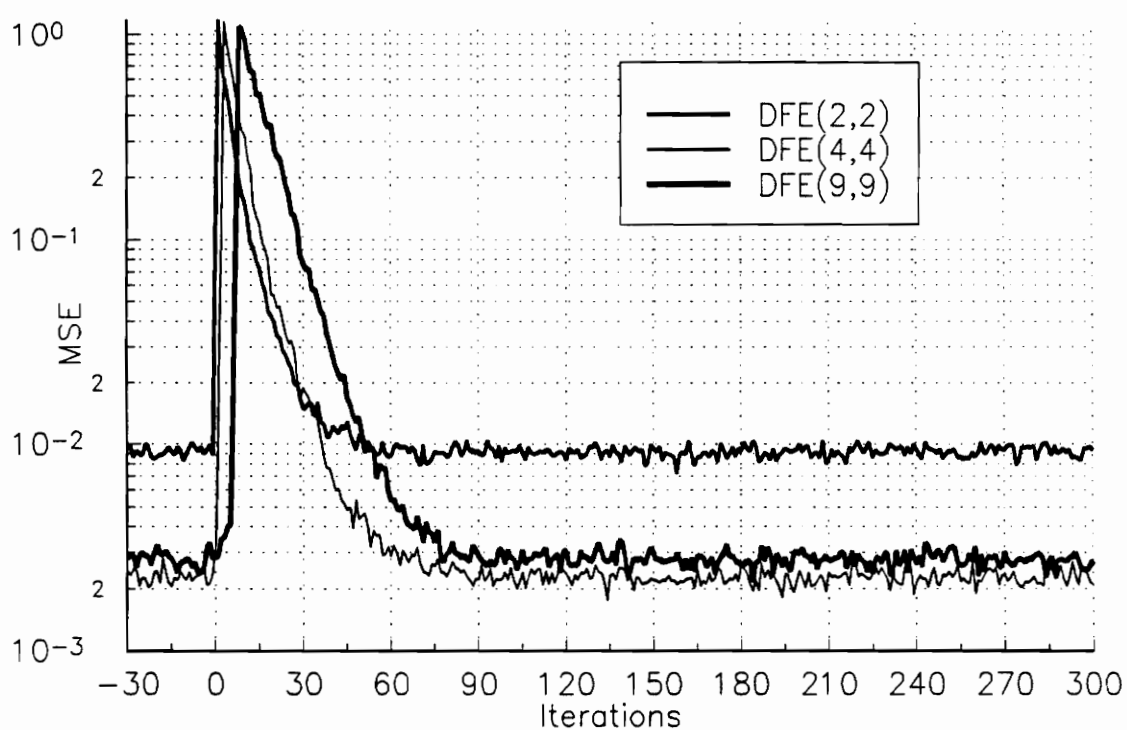


Figure 6.3-16. Tracking performance of different sized RLS-DFE in channel A $\lambda=0.95$.

algorithm has better tracking ability than an LMS algorithm. However, in Figures 6.3-1 and 6.3-6, which show the convergence performance of the RLS and the LMS algorithm, the RLS converges in 30 bits and the LMS converges in about 600 bits. The comparison between the tracking and convergence performance differences shows that the tracking superiority of the RLS algorithms over the LMS algorithms is not as good as the advantage in convergence.

6.4 Summary

This chapter presented convergence, SS-MSE and tracking performance of several adaptive equalizers and algorithms. Also, the effects of the size of equalizers, α and λ of the algorithms were shown on both the convergence and tracking capability. The simulation results are summarized below.

SS-MSE

Several equalizer structures are simulated. The decision feedback equalizer structures are superior to other choices, because they have small SS-MSE even when the channel is severely distorted or when there are nulls in the channel spectrum. The DFEs have the same complexity in the computational requirement as the LTEs.

Lattice equalizers have advantages because of their recursive order that allows the dynamic assignment of lengths of the lattice equalizer, and also because of their stability. Lattice DFEs can utilize the advantages of both lattice equalizers and DFE.

As for the effects of the parameters of the equalizers and algorithms, a smaller α in an LMS algorithm or a larger λ in a RLS algorithm causes smaller SS-MSE. There is an optimum size of equalizer for a channel. A smaller or larger size will cause extra ISI, and result in excess MSE.

Convergence Comparison

It was shown that RLS algorithms converge much faster than LMS algorithms. The step-size, α , controls the convergence rate and excess MSE of the equalizers using LMS algorithms. A bigger α decreases the converging time but

increases the SS-MSE. However, λ of the RLS algorithm has no effect on convergence, but only on the tracking. The size of an equalizer does not affect the convergence rate. Table 1 summarizes the convergence performance of different equalizers and different algorithms.

Tracking Comparison

An RLS algorithm has better tracking capability than an LMS algorithm. The size of an equalizer does not affect its tracking, but α and λ do. A larger α in LMS improves the equalizer's tracking capability, as does a smaller λ in RLS.

The trade-off between ISI immunity and tracking is a characteristic of all adaptive algorithms. A larger α and smaller λ cause larger SS-MSE, but they speed up the tracking. Therefore, greater ISI immunity results in less tracking speed, and vice-versa. The values of the parameters α and λ control the balance between reducing SS-MSE and tracking for LMS and RLS respectively.

7. Simulation of Mobile Channels

In the previous chapter, only stationary channels were discussed. In mobile communication, the channels change constantly. Although two stationary channels can be used to evaluate the tracking capability of an adaptive equalizer, they cannot be used to simulate mobile channels. Therefore, other mobile fading channel models should be used, such as two-ray Rayleigh fading channel models, to evaluate equalization performance.

7.1 Two-Ray Rayleigh Fading Channel

To test adaptive equalizers, one usually performs actual radio measurements or uses channel impulse response hardware simulators. However, it is difficult to control the delay spread in real-world channels, and hardware simulators are generally expensive. For simple testing and analysis, multiray impulse response models in which each ray has independent Rayleigh fading are widely used to evaluate the performance of adaptive equalizers and communication systems [33].

7.1.1 Rayleigh Fading Channel

The probability density function $p(\cdot)$ of the amplitude r of the Rayleigh fading channel is

$$p_r(r) = \frac{r}{b_0} \exp\left(-\frac{r^2}{2b_0}\right) \quad (7.1.1)$$

where b_0 is the mean power of the envelope.

✓ In propagation environments where a dominant component does not exist, multipath components with similar amplitudes come from random directions. In this situation, a Rayleigh fading distribution profile provides a good fit for measured CW data [33]. However, in the case where a component is dominant, a Rician or log-Gaussian distribution may be more accurate in the description of the measured data, because one component contains most of the power and does not

change very fast [33]. Despite this case, the Rayleigh fading channel is the most widely used model to simulate mobile channels.

If omnidirectional vertically polarized monopole antennas are used at the transmitter and receiver, and a CW signal is sent, the received RF signal frequency spectrum $S(f)$ of a Rayleigh flat fading channel is [32]

$$S(f) = \frac{1.5}{\pi f_D} \left[1 - \left(\frac{f - f_c}{f_D} \right) \right]^{-1/2} \quad (7.1.2)$$

where f_c is the carrier frequency and f_D is the Doppler frequency. The Doppler frequency is

$$f_D = \frac{v}{\lambda} \quad (7.1.3)$$

where v is the receiver or transmitter speed, and λ is the RF wavelength. Figure 7.1-1 shows the pass-band power spectrum of the receiving signal.

The variation of a channel is dependent upon the Doppler frequency. When a channel has a large Doppler frequency, its baseband and pass-band bandwidths are wide, so the channel has a fast variation. If the receiver speed is 100 km/h (27.78 m/s) and the carrier frequency is 850 MHz ($\lambda = 0.353$ m), the Doppler frequency, f_D , is 78.7 Hz. When the speed is 10 km/h, the Doppler frequency is only 7.87 Hz. Therefore, as a receiver moves faster, the channel changes faster. Also, from (7.1.3), the higher the carrier frequency, the faster the channel changes.

There are several methods to generate a Rayleigh fading channel. Jakes [33] recommends using two Doppler low-pass filters with white Gaussian noise to generate the real and imaginary parts of a flat Rayleigh fading channel. The structure of a Rayleigh fading channel generator is shown in Figure 7.1-2.

Another method of generating a Rayleigh fading channel is to multiply white Gaussian noise with the Doppler spectrum and take the inverse fast Fourier transform (IFFT) to get the real and imaginary parts of a flat Rayleigh fading channel.

Fung [23] used this method to generate Rayleigh fading channels in the bit

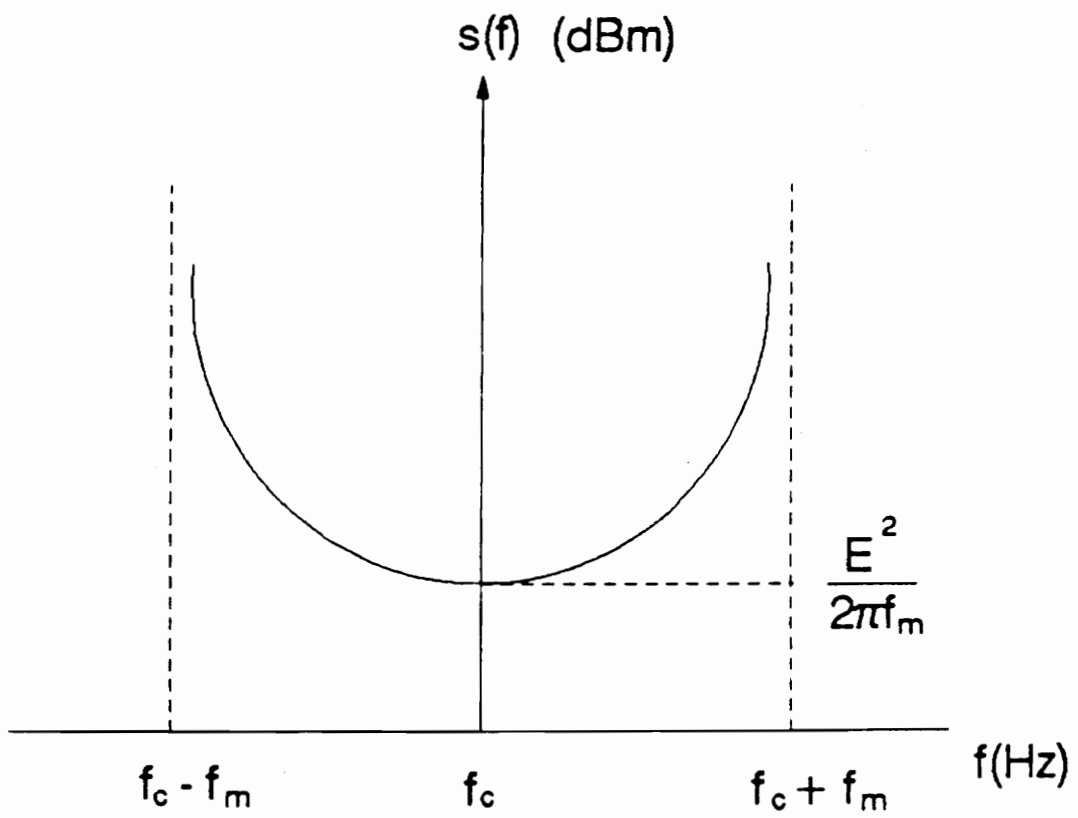


Figure 7.1-1. Theoretical power spectrum of a Rayleigh fading channel (From Fung's M.S. thesis [23])

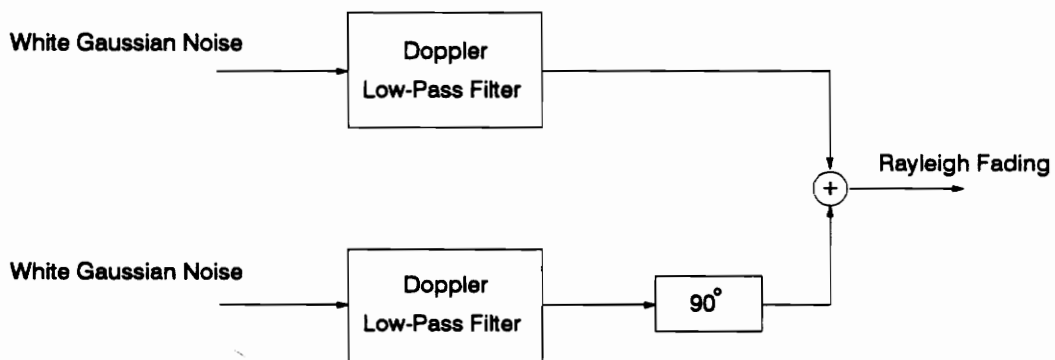


Figure 7.1-2. Structure of a Rayleigh fading channel generator.

error simulator called BERSIM. BERSIM simulated channels were used in the equalization simulation presented in this thesis. Two of the profiles generated by BERSIM are shown below in Figures 7.1-3(a) and (b), which are Rayleigh fading amplitude and phase envelopes for a vehicle speed of 75 km/h and those in Figures 7.1-4(a) and (b) are for a vehicle speed of 20 km/h. Radio wave has inphase (I) and quadrature (Q) components, and the amplitude of the radio wave is

$$A = \sqrt{I^2 + Q^2} \quad (7.1.4a)$$

The phase is

$$\theta = \arctan\left(\frac{I}{Q}\right) \quad (7.1.4b)$$

When the receiver moves faster, both amplitude and phase will change faster.

7.1.2 Two-Ray Channel

The one-ray Rayleigh fading channel is a flat fading channel, which has no delay spread. To obtain a frequency-selective fading channel, multipath components with delays must be employed. In this work, a two-ray model is used, and each ray has an independent Rayleigh fading characteristic that can be generated using the methods described in Section 7.1-1. The second ray of the channel has some delay which may cause intersymbol interference. The impulse response of a two-ray channel is:

$$h(t) = \sum_{i=1}^2 [c_{re}(i) + j c_{im}(i)] \delta(t - \tau_i) \quad (7.1.5)$$

where $c_{re}(i)$ and $c_{im}(i)$ are the inphase and quadrature parts of the i th ray and τ_i is the time delay of the i th ray. The mean power ratio of the two rays and the delay of the second ray are variable. Since the simulated receiver synchronizes to the first ray, τ_1 is equal to 0. From the definitions of $\bar{\tau}$ and σ_τ in (2.1.2, 2.1.3), it can be shown [3] that

$$\bar{\tau} = \frac{|c_2|^2 \tau_2}{|c_1|^2 + |c_2|^2} \quad (7.1.6a)$$

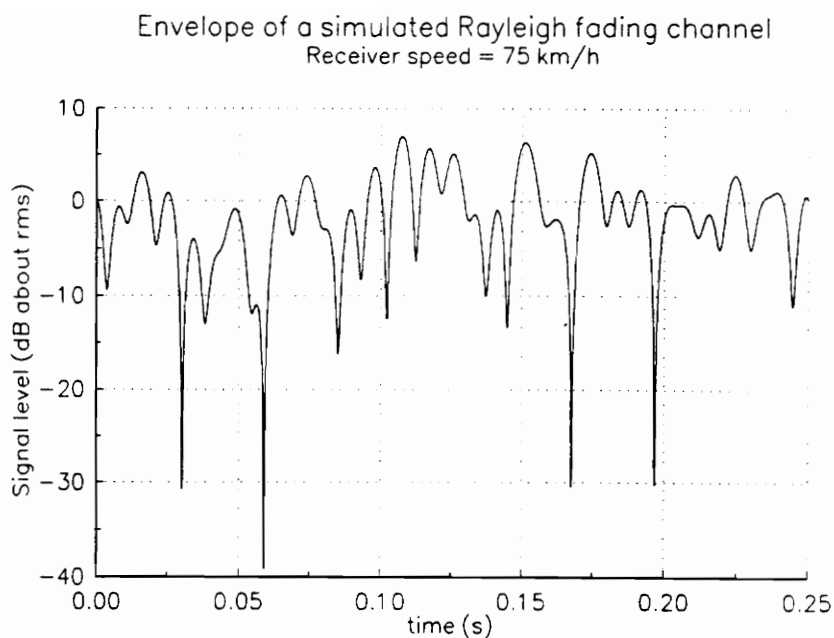


Figure 7.1-3(a). Envelope of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and the receiver speed is 75 km/h.

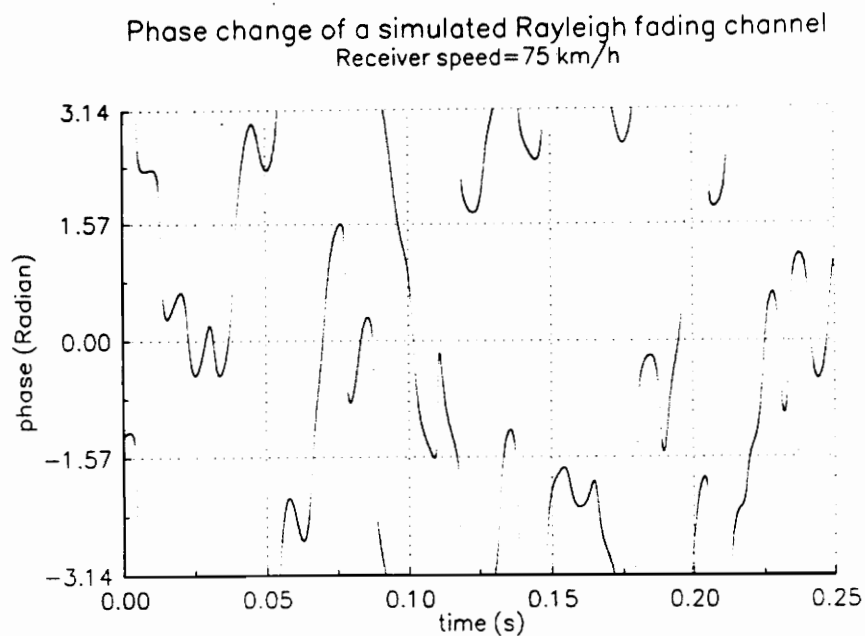


Figure 7.1-3(b). Phase of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and the receiver speed is 75 km/h.

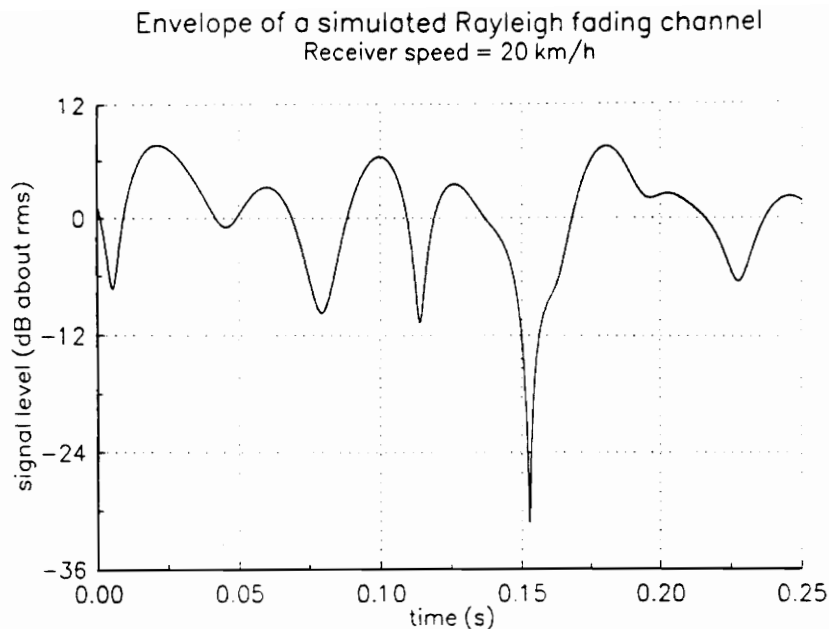


Figure 7.1-4(a). Envelope of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and receiver speed is 20 km/h.

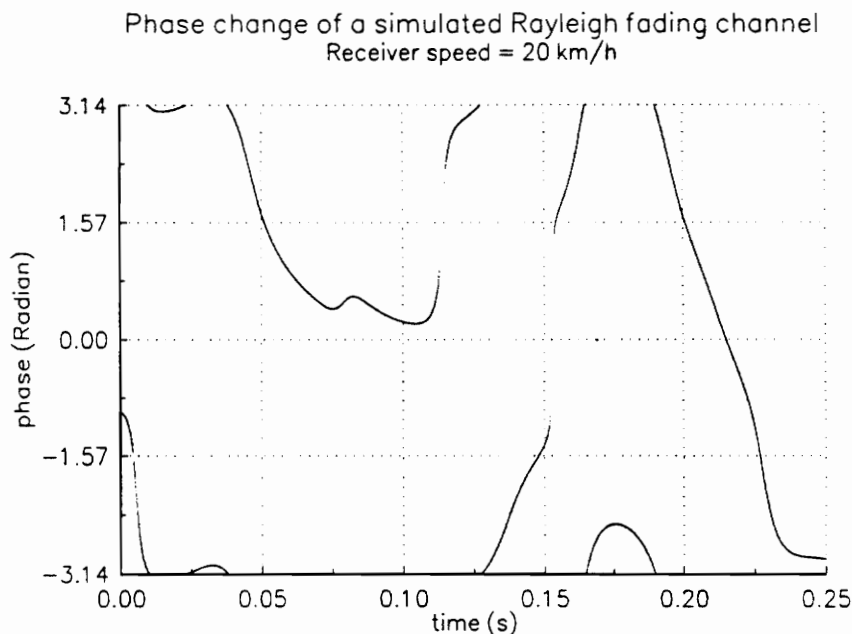


Figure 7.1-4(b). Phase of a simulated Rayleigh fading channel. The carrier frequency is 850 MHz, and the receiver speed is 20 km/h.

$$\bar{\tau}^2 = \frac{|c_2|^2 \tau_2^2}{|c_1|^2 + |c_2|^2} \quad (7.1.6b)$$

$$\begin{aligned} \sigma_\tau &= \sqrt{\bar{\tau}^2 - \bar{\tau}^2} \\ &= \sqrt{\frac{|c_2|^2 \tau_2^2}{|c_1|^2 + |c_2|^2} - \left(\frac{|c_2|^2 \tau_2}{|c_1|^2 + |c_2|^2} \right)^2} \\ &= \frac{|c_2| |c_1| \tau_2}{|c_1|^2 + |c_2|^2} \end{aligned} \quad (7.1.7)$$

If the two rays have the same mean power, the rms delay spread σ_τ of a two-ray model is

$$\sigma_\tau = \frac{\tau_2}{2} = \frac{\tau}{2} \quad (7.1.8)$$

Since $\tau_1=0$, τ_2 is expressed as τ in (7.1-8).

7.1.3 Power Loss in Wideband Channels

As explained in Chapter 1, an equalizer uses wideband information which means that the signal spectrum is wider than the channel coherent bandwidth, and the wideband fading is computed using (2.1-6) which sums the power of all the multipath components in the channel impulse response. Wideband fading is much smaller than CW fading computed using (2.1-7) which does the phasor summation of all the multipath components. CW and wideband fading of a two-ray independent Rayleigh fading channel such as the one in (7.1-4) are presented in Figures 7.1-5 and 7.1-6 which show that the amplitude fluctuation is smaller for a wideband channel. In wideband channels, nulls only appear when all multipath components are small.

In actual communication systems, the bandwidth is neither CW nor wideband, so different multipath components closely separated in time may partially cancel each other. Hence, the energy fluctuation received within the signal bandwidth will be smaller than the CW fading, but larger than the fading

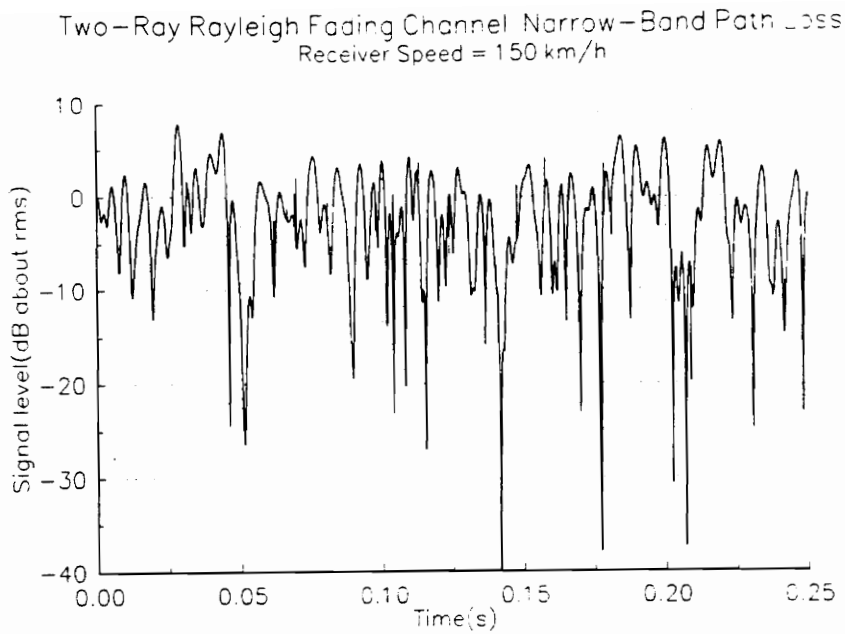


Figure 7.1-5. Narrow band path loss of a two-ray Rayleigh fading channel.

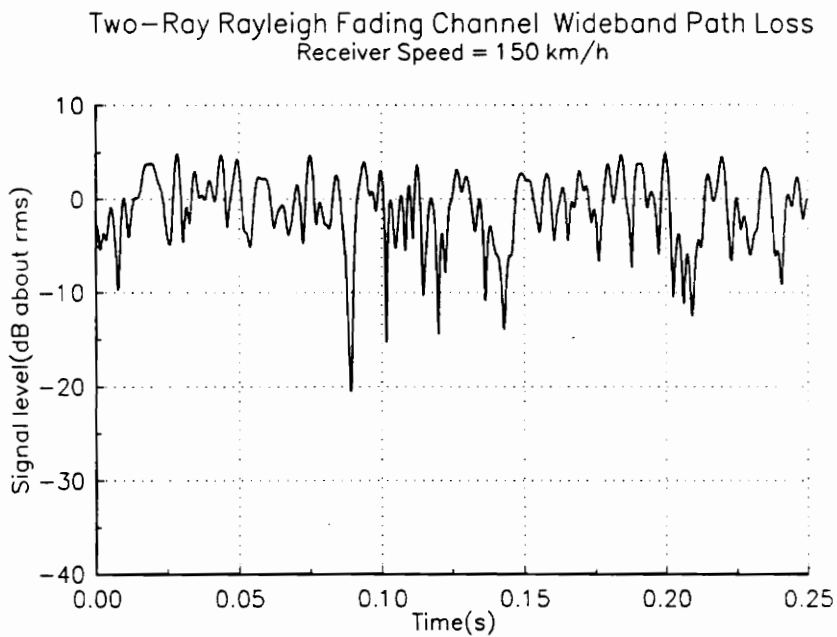


Figure 7.1-6. Wide band path loss of a two-ray Rayleigh fading channel.

over an infinite bandwidth.

7.2 Evaluation of E_b/N_o

Often E_b/N_o is used as a measure of SNR in digital systems, because it is a normalized quantity that is independent of bit-rate or bandwidth [34].

E_b is the energy per bit at the transmitter minus the channel power loss. Since both the transmitted power P_T and the channel gain G_c are normalized to unity, for QPSK and $\pi/4$ DQPSK, which transmit two bits per symbol

$$E_b = \frac{P_T T_s G_c}{2} = \frac{T_s}{2} \quad (7.2.1)$$

In the simulation presented in this thesis, white Gaussian noise is added at the receiver. According to Couch [34], the SNR after the matched filter of a receiver is

$$\left(\frac{S}{N}\right)_{out} = \int_{-\infty}^{+\infty} \frac{|S(f)|^2}{N_o/2} df = \frac{2}{N_o} \int_{-\infty}^{+\infty} s^2(t) dt = \frac{2E_s}{N_o} \quad (7.2.2)$$

Since signal power is normalized and white Gaussian noise with the power of σ^2 is added after the matched filter,

$$\left(\frac{S}{N}\right)_{out} = \frac{1}{\sigma^2} = \frac{2E_s}{N_o} \quad (7.2.3)$$

For QPSK modulation, $E_s = 2E_b$. Then

$$\frac{E_b}{N_o} = \frac{1}{4\sigma^2} \quad (7.2.4)$$

Using (7.2.4), σ can be chosen for a specific E_b/N_o .

7.3 Simulation of Equalization in Two-Ray Channels

This equalization simulation was done for the USDC standard. The standard uses a TDMA system and $\pi/4$ DQPSK modulation with a 48.6 kbps data rate. The symbol duration is 41.15 μ s. Each data slot contains 162 symbols, in

which the first 14 symbols are used for synchronization and training [22].

In the simulation of equalization in two-ray channels, the channels were independently generated by BERSIM. The delay of the second ray, τ was set to one symbol duration, $41.15 \mu\text{s}$. Since both rays of the channel are independently Rayleigh distributed with equal mean power, the mean rms delay spread is $\frac{\tau}{2}$. The carrier frequency used was 850MHz. QPSK modulation was used here. Perfect timing and frequency synchronization were assumed. No coherent detector is used before the equalizer, although it can be implemented. Therefore, the adaptive equalizer also serves as a coherent detector. It is possible that using an independent coherent detector can improve the BER performance of the system, but no simulation has been done to prove that. The equalizers were implemented at baseband by using I and Q signal demodulation, and then the baseband signal was equalized. Perfect decision feedback was assumed.

The multipath fading channels were generated beforehand for different speeds using BERSIM. About 6,000 consecutive channel impulse responses were generated for each channel with a spacing of a symbol duration $T_s=41.15\mu\text{s}$. Different equalizers used the same channels so that a fair comparison could be obtained. When each symbol was sent, a new channel impulse response was used. Hence, about 6,000 symbols or 12,000 bits could be transmitted through each channel generated.

Three forward and two feedback taps DFE and L-DFE (DFE(3,2) and L-DFE(3,2)) were used. The MLSE used a two tap CIRE. The weighting factor λ used was 0.9. The data sequences were formatted into 210 symbol slots. The first 10 symbols were a preamble used to train the equalizers, and the following 200 symbols were used to evaluate BER performances with perfect detections assumed. A 210 symbol slot is more than the 162 symbol slot adopted by the USDC standard, but is on the same order. Due to computational requirements, only 10,000 bits were used to test the performance of each equalizer, so the results may lack accuracy at $\text{BER} < 10^{-3}$.

The sensitivities of the equalizers to different vehicle speeds were tested by setting the mobile speed to 0, 20, 50, 75, 100, 150 km/hour. Also, the BER as a function of E_b/N_o was evaluated. White Gaussian noise was applied to the input

of an equalizer. Pulse-shaping was not implemented, because if the delay of the second ray is equal to one symbol duration, the results in the two-ray model are the same with or without pulse shaping. In the real world, the line-of-sight ray often has a smaller path loss compared to a reflected ray, so the mean power ratio of the two rays was allowed to vary to determine its influence on the equalization performance.

7.4 Simulation Results

Figure 7.4-1 shows the irreducible ($E_b/N_o=\infty$) BER performance of DFE, L-DFE and MLSE at different vehicle speeds. The multipath delay of the two-ray model was set to $\tau=T_s$, the symbol duration equal to $41.15\mu s$. From the result, it is clear that the equalizers are very sensitive to the vehicle speed. The BER at 120 km/h is about 10^{-2} and 100 times the BER at 20 km/h, which has a BER about 10^{-4} . The reason is that a fast moving vehicle implies a fast changing channel, and the fast changing channel causes more bit errors due to the equalizer tracking error. The performances of the three equalizers are very close. All the equalizers were able to keep the BER less than 1% at 100 km/h. Compared with the 25% BER when an equalizer was not used (See Figure 7.4-1), vast improvement is evident with an equalizer.

Figure 7.4-2 illustrates BER performance versus E_b/N_o at vehicle speeds of 0, 20 and 75 km/h. The performances of the three equalizers remain close. This result further proves that adaptive equalizers have a better overall performance in slow changing channels, because the tracking error is smaller. At low E_b/N_o , the differences in BER at different speeds are small, and the larger the E_b/N_o , the more significant the BER differences. This is because when noise is large, it is the main source of errors compared to the channel tracking error. When the noise power decreases, the channel tracking error becomes more important.

If the two-ray model is used to represent the real world channel, it is not a good idea to use two rays with the same mean power. The two-ray channel with a different mean power ratio P_1/P_2 was tested, where P_1 is the mean power of first ray's Rayleigh distribution and P_2 is that of the second.

To simulate the two-ray channel with a different mean power ratio, a two-

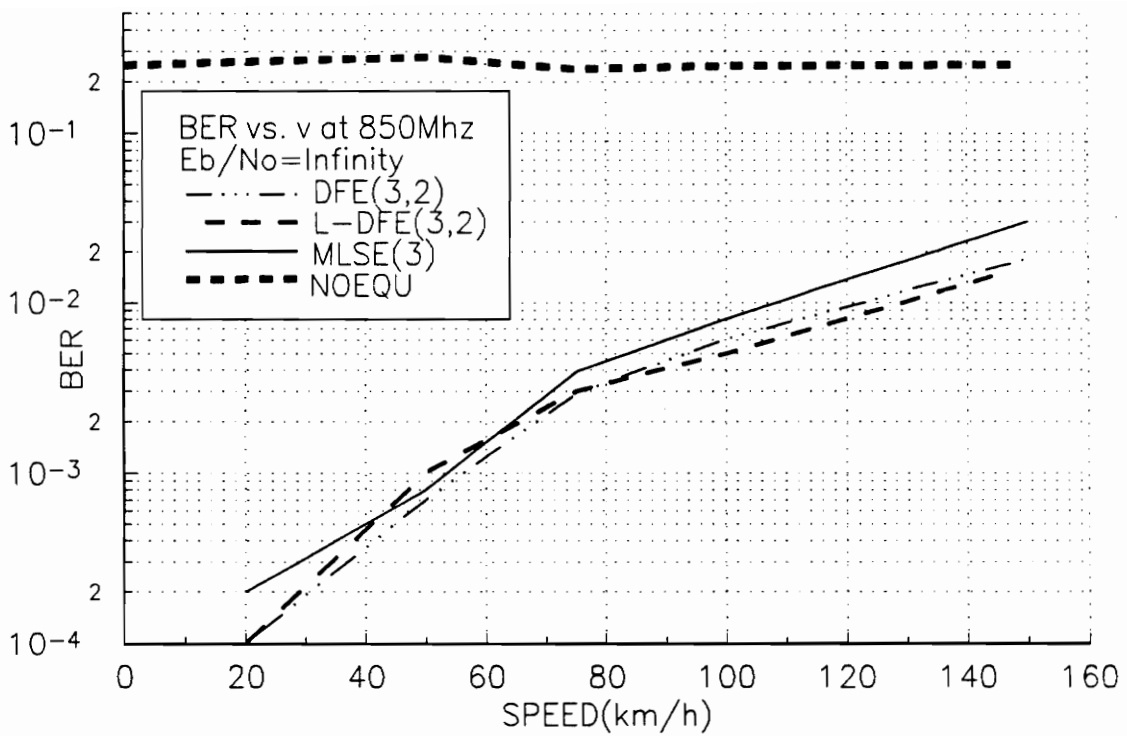


Figure 7.4-1. BER vs. speed of the mobile using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. The carrier frequency is 850 MHz, $E_b/N_o = \infty$. Two-ray Rayleigh fading channel, and the delay of the second ray is T_s .

BER vs. E_b/N_o on Two Ray Rayleigh Fading Channels

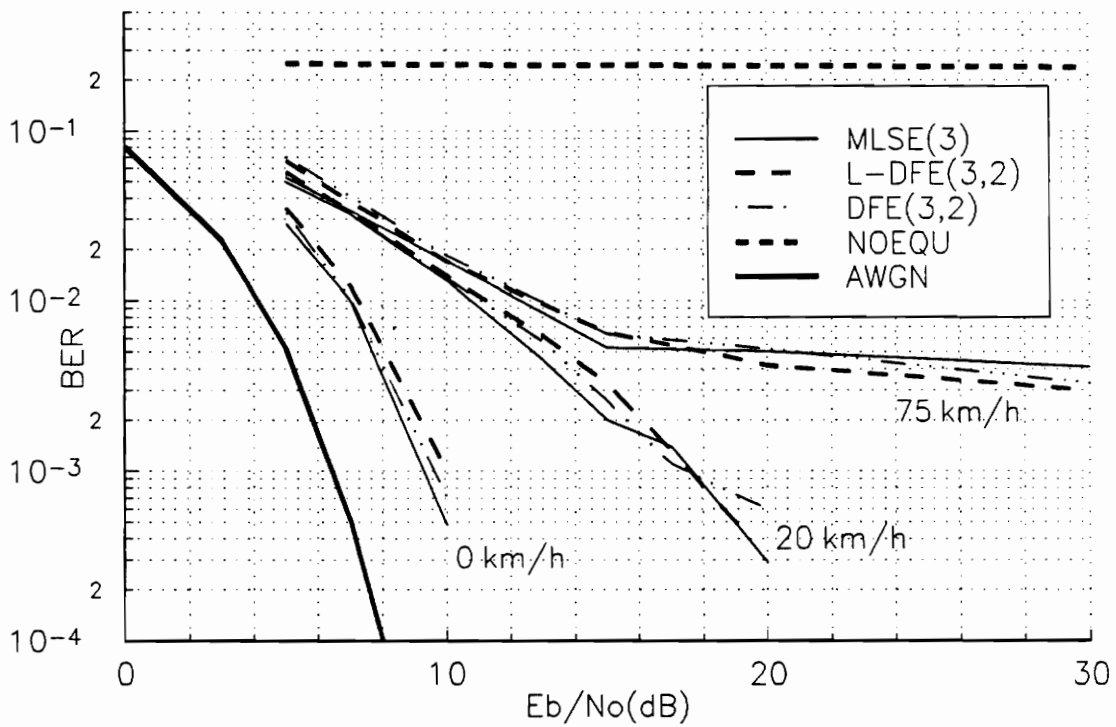


Figure 7.4-2. BER vs. E_b/N_o in two-ray Rayleigh fading channel using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. The delay of the second ray is T_s .

ray model with the same mean power was utilized, and the two rays were scaled by multiplying each ray amplitude with a constant to make the total power of the two rays equal to unity. Recall that the reason for setting the channel power to 0 dB is because it is easy to add noise and compute E_b/N_o .

The equalization results with different the values of P_1/P_2 are shown in figure 7.4-3. A speed of 75 km/h and $E_b/N_o=20$ dB were chosen. The graph shows that when $P_1/P_2=0$ dB, the BER is the smallest. Also, the values of BER are quite symmetric around $P_1/P_2=0$ dB. The larger the difference of the two rays' amplitudes, the worse the BER is. This indicates that information through both rays is helpful for an equalizer to reduce the impact of noise. An equalizer not only mitigates the ISI, but utilizes the multipath diversity to reduce bit-errors.

Figure 7.4-3 also shows that when $v=75$ km/h, the systems with and without an adaptive equalizer have a similar BER performance at $P_1/P_2=12$ dB. If P_1/P_2 increases, the result without adaptive equalization is better. This indicates when working in a flat fading channel, or a channel with a very small rms delay spread, an adaptive equalizer will not improve the BER, instead, it may harm the system.

All the non-equalization results were obtained assuming the receiver tracked the first ray. In a real world system, the receiver tracks the mean excess delay of a channel [1]; therefore the non-equalization results will be better than those shown in Figure 7.4-3, especially when the P_1/P_2 is smaller than 0 dB. One would expect the non-equalization results to be symmetric about $P_1/P_2=0$ dB.

From the results, one can see that the performances of the three adaptive equalizers are similar. With excellent convergence abilities, these equalizers usually can converge within 10 symbols. Also, they have good tracking capability. When the vehicle moves less than 20 km/h and the E_b/N_o is larger than 18 dB, the BER can be as low as 10^{-3} . Without an equalizer, the BER would be more than 20% according to the simulation results. The simulation indicates that equalizers work best when the mean powers of the two rays are equal.

To simulate a two-ray channel with a delay of the second ray not equal to T_s , pulse shaping has to be used in the system, and this will be discussed next.

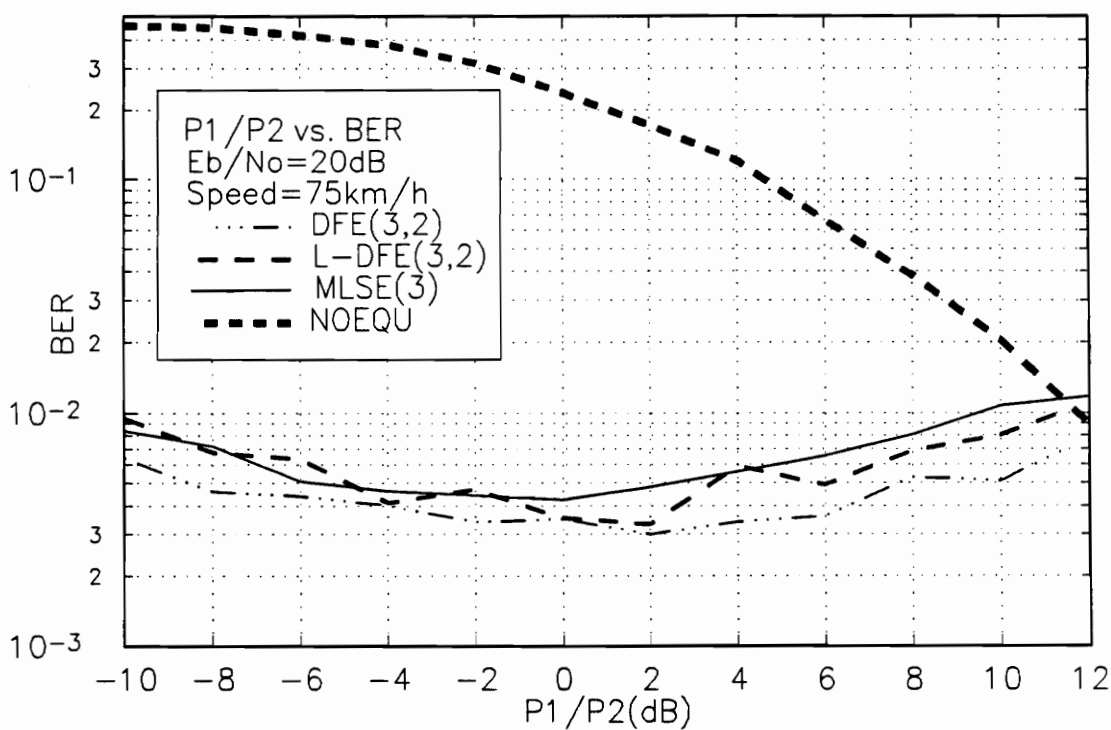


Figure 7.4-3. P_1/P_2 vs. BER in a two-ray Rayleigh fading channel using DFE(3,2), L-DFE(3,2), MLSE(3) and non-equalization. $E_b/N_0 = 20$ dB, and the carrier frequency is 850 MHz. Mobile speed is 75 km/h.

7.5 Band-limited channels

Pulse shaping was not used in the previous simulation. In a communication system, however, the spectrum has to be used efficiently, so Nyquist pulse shaping is used in order to send digital data in a small bandwidth without causing ISI. The raised-cosine filter, is one class of filters that satisfy Nyquist's first criterion: [34]

$$h(kT_s) = \begin{cases} C, & k=0 \\ 0, & k \neq 0 \end{cases} \quad (7.5.1)$$

where $h(t)$ is the impulse response of a pulse shaping filter, k is an integer, T_s is the symbol duration, kT_s is the sampling time and C is a non zero constant.

The transfer function of a raised cosine filter is [34]

$$H_e(f) = \begin{cases} 1, & |f| < f_1 \\ \frac{1}{2} \{1 + \cos[\frac{\pi(|f| - f_1)}{2f_\delta}]\}, & f_1 < |f| < B \\ 0, & |f| > B \end{cases} \quad (7.5.2)$$

where $f_\delta = B - f_0$, $f_1 = f_0 - f_\delta$, B is the absolute bandwidth and f_0 is the 6 dB bandwidth. The impulse response of a raised-cosine filter is

$$h_e(t) = 2f_0 \left[\frac{\sin(2\pi f_0 t)}{2\pi f_0 t} \right] \left[\frac{\cos(2\pi f_\delta t)}{1 - (4f_\delta t)^2} \right] \quad (7.5.3)$$

where r is the rolloff factor

$$r = \frac{f_\delta}{f_0} \quad (7.5.4)$$

In the time domain $h_e(kT_s) = 0$ (k is an integer) when $k \neq 0$, so that it satisfies Nyquist's first criterion (7.5.1). The signal bandwidth increases with the rolloff factor, which can be between 0 to 1. The relation between the baseband bandwidth and r is [34]

$$B = (1+r)W \quad (7.5.5)$$

where W is the symbol rate of the system. When $r=0$, the baseband bandwidth of the signal equals its minimum value W without causing ISI. However, systems using a small bandwidth need extremely accurate sample-timing requirements, or comparatively large ISI will appear as soon as timing jitter occurs. If the rolloff factor is chosen large, the signal bandwidth increases, but the sample-timing requirement is eased. Hence, the choice of r is actually the choice between sample-timing error and spectrum conservation. Figures 7.5-1(a) and (b) show the impulse response and transfer function of raised cosine filters with different rolloff factors. It is clear that a larger r requires a larger bandwidth, but in the time domain, the slope of the impulse response at kT_s ($k \neq 0$) is smaller; hence, if timing jitter occurs, ISI will be smaller.

Using raised-cosine filters in flat fading channels limits the bandwidth without causing ISI. However, for frequency-selective channels, raised-cosine filters cannot avoid ISI, and instead they spread small delays to several symbols. Therefore, a larger-sized equalizer may be needed. Figure 7.5-2(a) shows the impulse response of two individual rays with the same power, where the delay of the second ray is $\frac{1}{3}T_s$. Figure 7.5-2(b) is the impulse response of the two-ray channel. Figure 7.5-2(b) shows that although the excess delay is only one third of the symbol duration, it is spread out to many symbols due to the pulse shaping, and thus becomes a multiray channel. However, it can be seen from Figure 7.5-2(b) that these rays become very small when the time delay is large; therefore, the equalizers need not to be very large, since the residual ISI can eventually be considered background noise.

7.6 Simulation of Raised-Cosine Filters

The raised-cosine filters used in the simulations presented here are 6 symbols long. Because a raised-cosine filter is a non-causal filter, it has to be shifted 3 symbols to be causal. The shift causes a delay of 3 symbol durations. The rolloff factor was set to 0.35, which is adopted by the USDC standard [22]. Figure 7.6-1 is a simplified system block diagram that shows how to implement the raised-cosine filters. Two raised-cosine filters are used in each system, one for the inphase and the other for the quadrature signals. Both filters are used at the

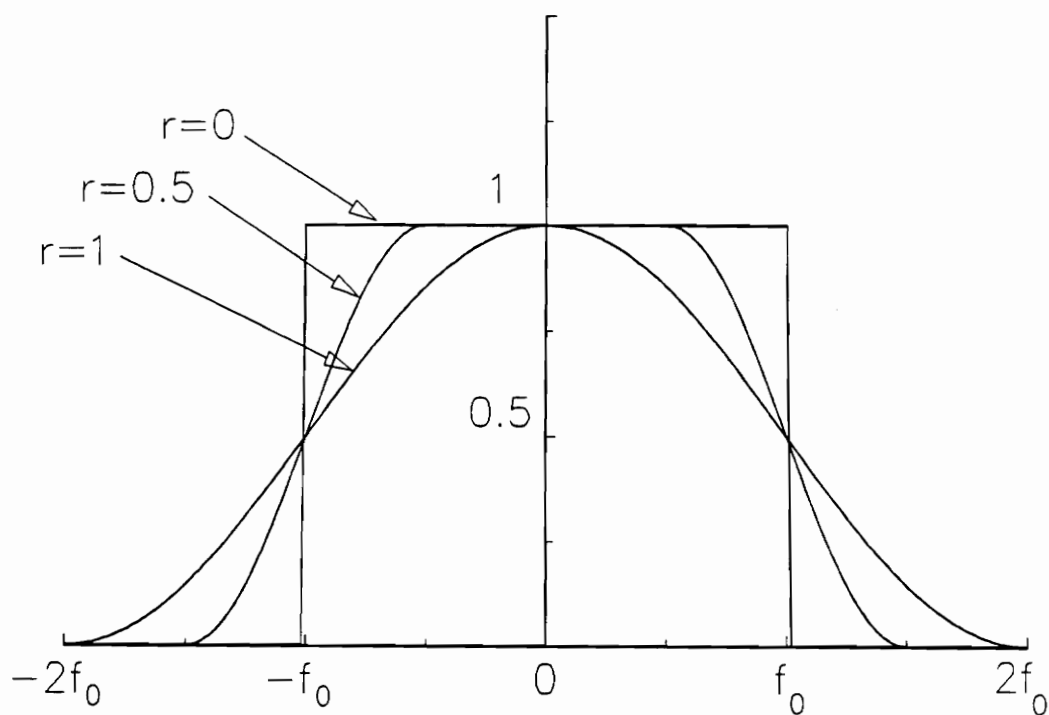


Figure 7.5-1(a). Magnitude frequency responses of raised-cosine rolloff filters with different rolloff factors.

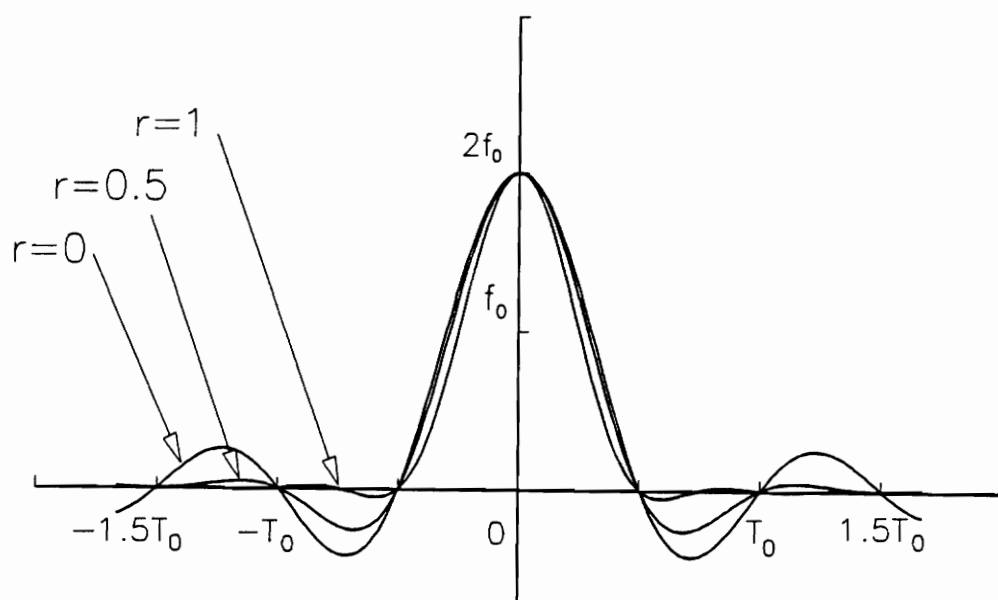


Figure 7.5-1(b). Impulse response (time) of raised-cosine filters with different rolloff factors.

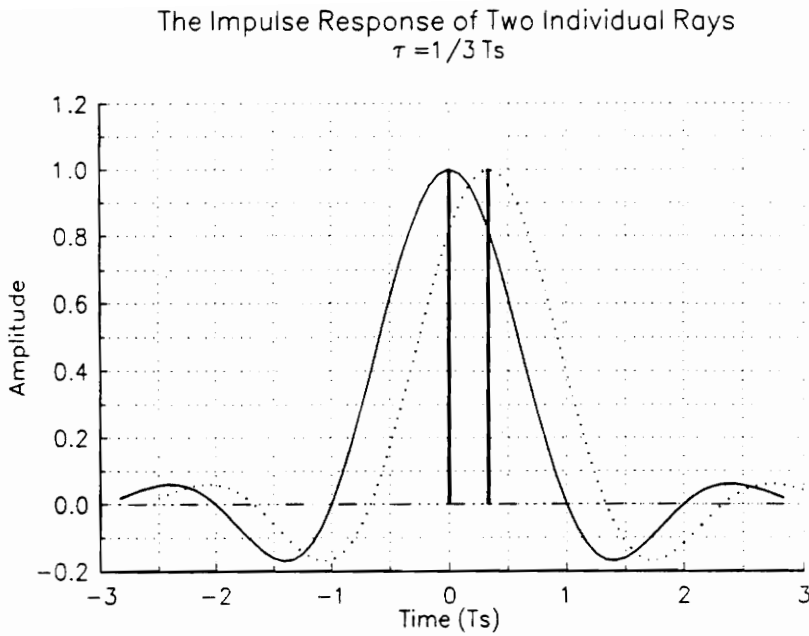


Figure 7.5-2(a). The impulse response of two individual rays using a raised-cosine filter. The delay of the second ray is $\frac{1}{3} T_s$.

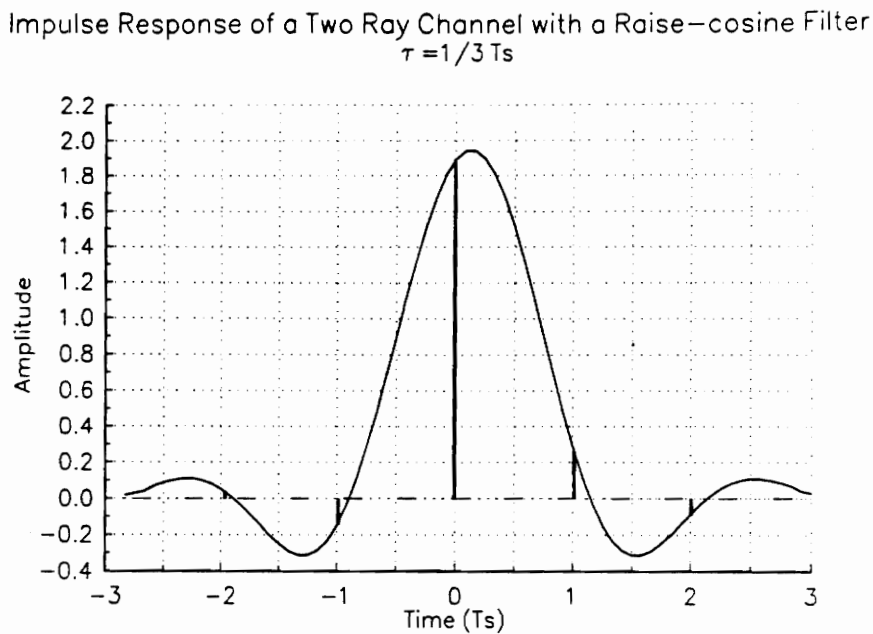


Figure 7.5-2(b). The combined impulse response of a two-ray channel using a raised cosine filter. $\tau = \frac{1}{3} T_s$.

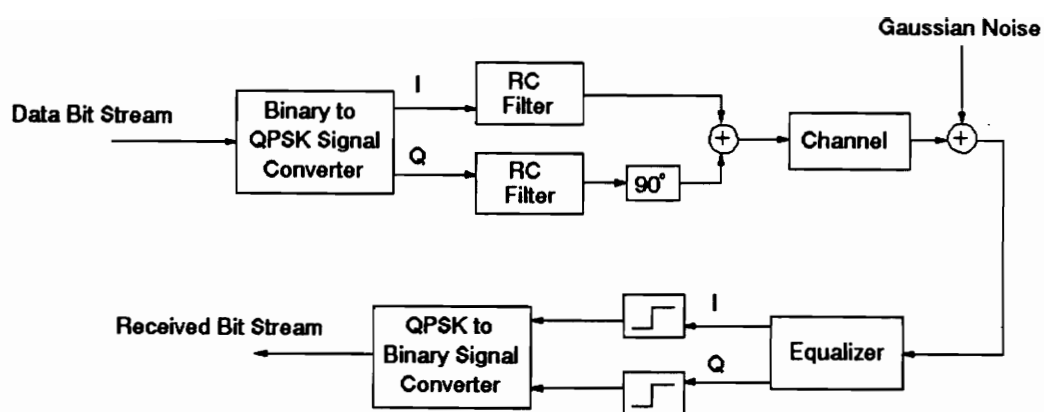


Figure 7.6-1. A baseband model of a QPSK system using raised-cosine filters and equalization.

transmitter to preserve the frequency spectrum. In an actual system, four *square root* raised-cosine filters are used. Two are in the transmitter and the other two serve as the matched filters in the receiver to maximize the SNR [23]. Using more filters in the simulation, however, requires more convolutions and more computation time. Raised-cosine filters have a little worse BER performance compared to *square-root* raised-cosine filters, because using raised-cosine filters does not produce the effect of matched filters.

In the simulation, each symbol duration was divided into 12 time bins. A two-ray Rayleigh fading channel model in which the two rays have the same mean power was used to simulate mobile channels. DFE equalizers with an RLS algorithm were used because their low SS-MSE and good convergence and tracking capabilities compared to other options as shown in Chapter 6. The performance of the equalizers with a different τ of the second ray was tested. Also, results at different vehicle speeds and E_b/N_o were shown. The effect of the rolloff factor was tested.

7.7 Results of Equalization in Systems with Raised-cosine Filters

Two-ray Rayleigh fading channels at 850 MHz are implemented here. Coherent QPSK is used as the modulation, and the data rate is 48.6 kbps. The data structure is same as described in Section 7.3. No coherent detector is used before the equalizer.

Figure 7.7-1 shows the irreducible BER using adaptive equalization in mobile channels with different τ and different speeds. 10000 bits were tested for each point. One can find that the vehicle speed has a large influence on the BER, and the BER of 75 km/h is better than that of 100 km/h. The result of a speed of 20 km/h is superb: the BER is less than 2×10^{-3} at all τ .

The value of τ is also a factor of the BER performance, although not as important as for an unequalized receiver. The equalizers work better when the τ is larger until τ is about T_s . Compared with the performance of a system without an equalizer, an equalizer does not help a system when τ is very small. At speeds from 75 to 100 km/h, if τ is smaller than about $\frac{1}{6}$ to $\frac{1}{12}T_s$, the performance without an equalizer is actually better. When the delay of the second ray is larger

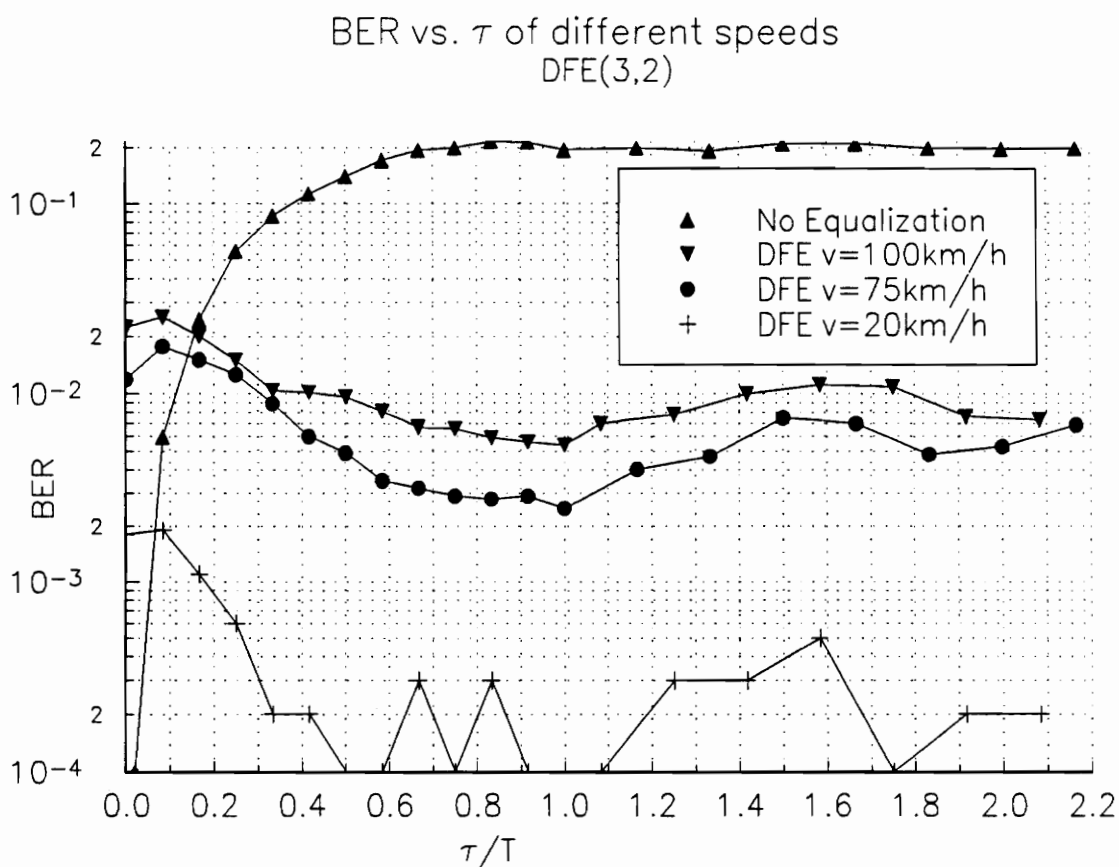


Figure 7.7-1. BER vs. τ of two-ray Rayleigh fading channel using DFE(3,2) at different speeds. Carrier frequency is 850MHz.

than that, the equalizer can improve the system performance as much as 100 times. If the data rate is 48.6 kbps, then the critical delay of the second ray is about 4 to 7 μs . Related to the rms delay spread σ_τ , the critical point at which an equalizer is needed is about 2 to 3.5 μs . Rappaport et al [2] indicate that 20%-30% of the locations they measured in urban areas have rms delay spreads larger than 3.5 μs .

If the vehicle speed is 20 km/h, an equalizer can improve the BER performance even when τ is about $\frac{1}{12}T_s$, which is same as $\frac{1}{24}T_s$ rms delay spread. If the data rate is 48.6 kbps and the symbol duration is 41.15 μs , the rms delay of the changing point $\frac{1}{24}T_s$ is lower than 2 μs . More than 50 % of locations measured in [2] have an rms delay spread larger than this.

Although the rms delay spread in [2] is not directly applicable to the rms delay spread of a two-ray model for adaptive equalization, this result shows that adaptive equalization can improve the mobile system over a large range of rms delay spreads. The simulation shows clearly that the rate a mobile channel changes is extremely important in terms of the degree to which an adaptive equalizer can improve a system. When a mobile channel does not change very fast, adaptive equalization can improve the system significantly. In this situation, maybe only an LMS algorithm or even a non-adaptive equalizer is needed since the change within a data slot of 200 symbols is very small and the equalizer will be retrained after each data slot.

The irreducible BER to different rolloff factors was tested with τ set to $\frac{T_s}{2}$. Figure 7.7-2 shows that the rolloff factor has very limited influence on the equalization performance. It seems an equalizer works best when rolloff factor is about 0.4 to 0.5.

The noise influence on the BER is also shown in Figure 7.7-3. Two specific τ values were chosen: $\frac{1}{6}T_s$ and $\frac{7}{12}T_s$. It is shown that when the E_b/N_o is small, the noise is more important in causing errors, so the BER of the two τ 's are similar. While the E_b/N_o is large, the number of bit errors is determined by the value of τ . The simulated BER of a receiver without equalization in QPSK in an additive white Gaussian (AWGN) channel is also shown in the Figure 7.7-3, and these results are the same as the theoretical results [34,23]. An AWGN channel is

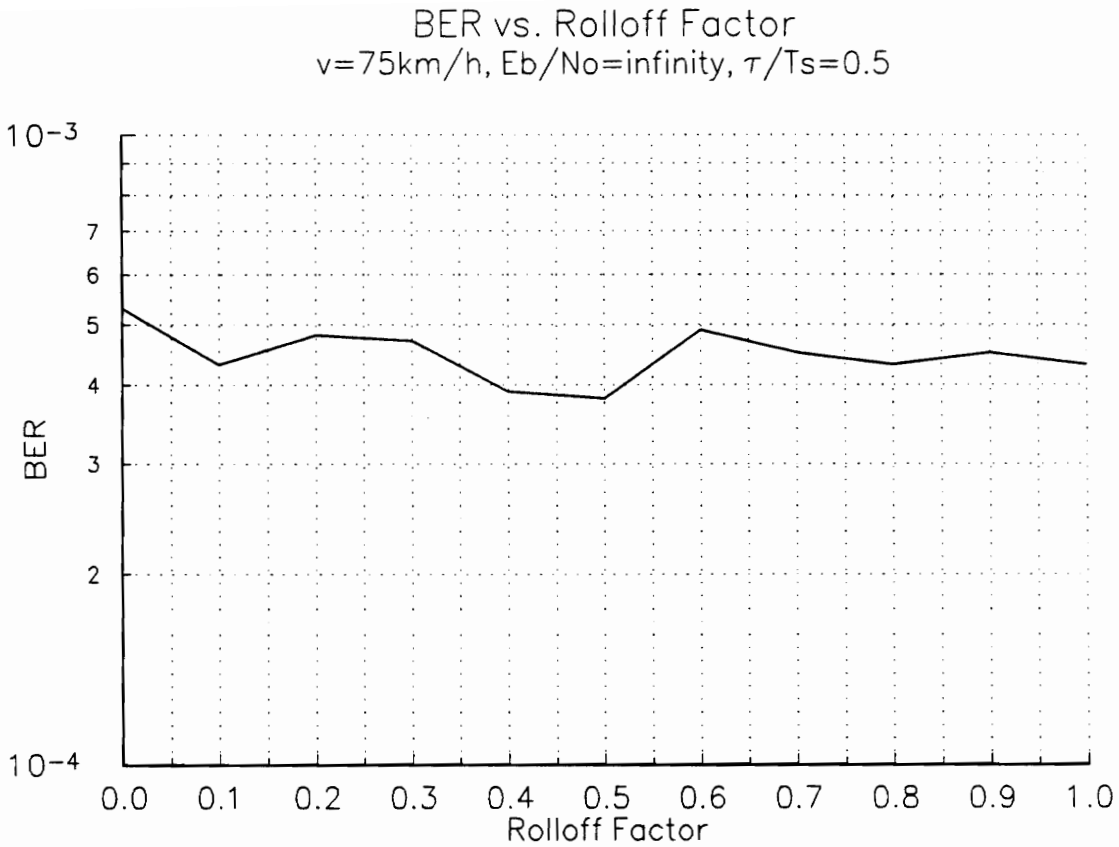


Figure 7.7-2. BER vs. the roll off factor of the raised-cosine filter using RLS-DFE(3,2) in a two-ray Rayleigh fading channel. $\frac{\tau}{T_s}=0.5$. Carrier frequency is 850 MHz, $v=75\text{km/h}$, $E_b/N_o=\infty$.

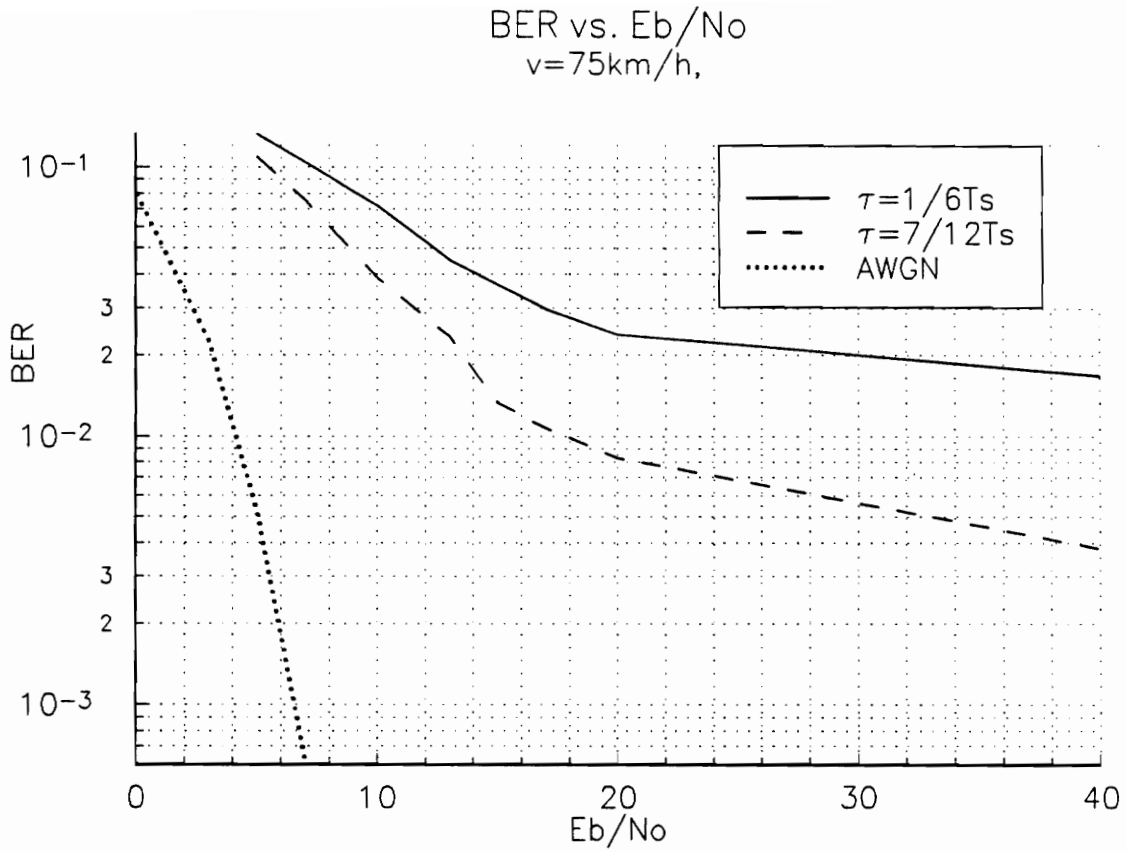


Figure 7.7-3 BER vs. E_b/N_o using RLS-DFE(3,2) in a two-ray Rayleigh fading channel, when $\frac{\tau}{T_s}=\frac{1}{6}$ and $\frac{\tau}{T_s}=\frac{7}{12}$. Carrier frequency is 850 MHz, $v=75\text{km/h}$, $E_b/N_o=\infty$.

a flat non-fading channel, so it is simulated as a stationary impulse with unity power and white Gaussian noise is the only source of error.

To choose the optimal size for the DFE in a two-ray channel, a series of different DFEs were tested. The value of τ is from 0 to $2T_s$, the range most multipath components will be in for the USDC system [2,36]. First of all, the number of forward taps was fixed in order to evaluate the effects of different numbers of feedback taps. The simulation results are in Figure 7.7-4. Although DFE(5,1) has a good performance when $\tau < T_s$, the BER for $\tau > T_s$ deteriorates. Hence, once a significant long-delay multipath component appears, the error increases significantly since only one feedback tap is available. Among the other sizes, DFE(5,5) seems to be the poorest, because the size is much larger than the delay spread of the channel; thus it results in equalizer-induced ISI. The DFE(5,2) and DFE(5,3) are the best equalizers, but since smaller size needs less computation, DFE(5,2) is the proper choice. Therefore, the best number of feedback taps is 2.

Then, the optimum number of forward taps was chosen in the same way by selecting 2 feedback taps, and the results are shown in Figure 7.7-5. Although DFE(2,2) has good performance when $\tau < T_s$, the BER increases for $\tau > T_s$. The other three choices have similar performance, so 3 is chosen as the number of forward taps because of its smaller computational requirement. Finally, DFE(3,2) is selected as the optimal size of the DFE; however, DFE(2,1) may be a better choice if all the multipath components are restricted within one symbol duration. This example shows how the worst-case multipath delay spread impacts equalizer design.

7.8 Summary

In the above simulations, the raised-cosine filters were used as pulse shaping filters. The adaptive equalization results for the USDC system with different delay spreads were obtained. It was found that adaptive equalizer performance depends highly on the speed of channel variation. A faster changing channel causes larger tracking error. Also, an equalizer may work better with some channel ISI than with very small ISI. If the delay spread is too small, no adaptive equalization is actually better. Using different sized equalizers, it is

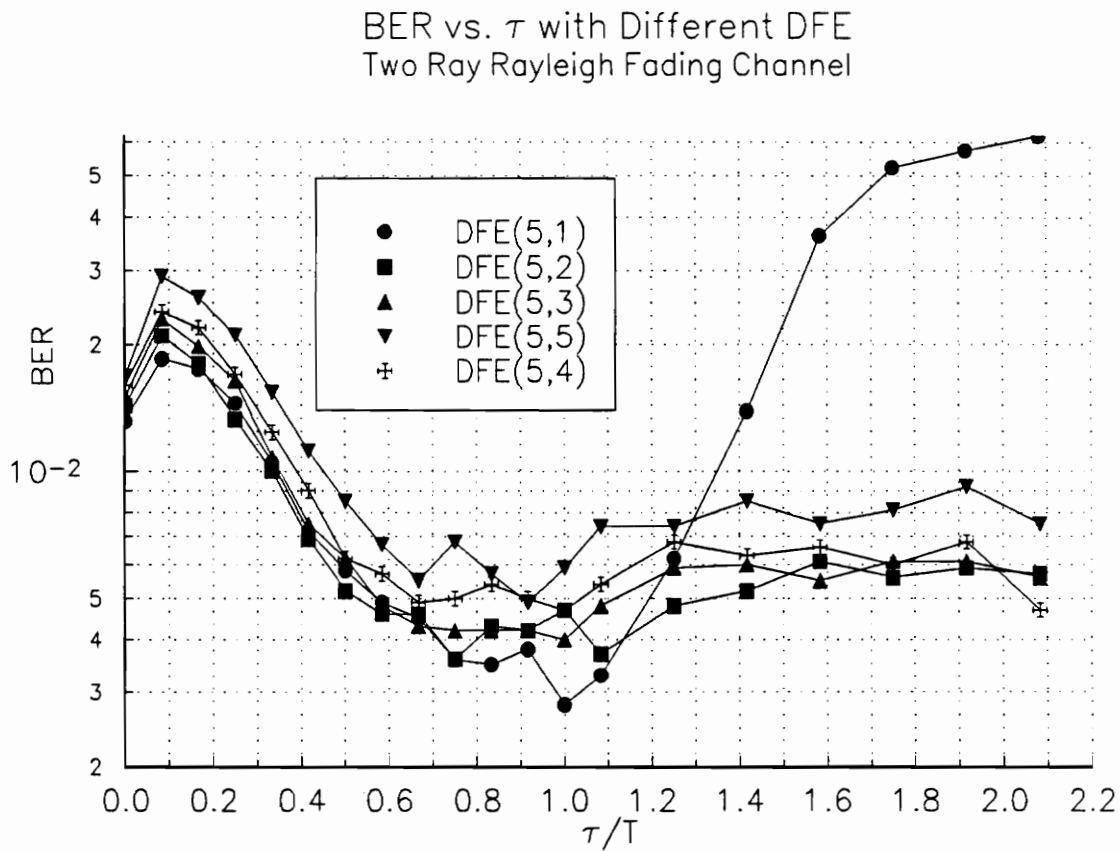


Figure 7.7-4. BER vs. τ for a two-ray Rayleigh fading channel with 5 different numbers of feedback taps of RLS-DFE and the number of forward taps is 5. Carrier frequency is 850 MHz, $v=75$ km/h, $E_b/N_o=\infty$.

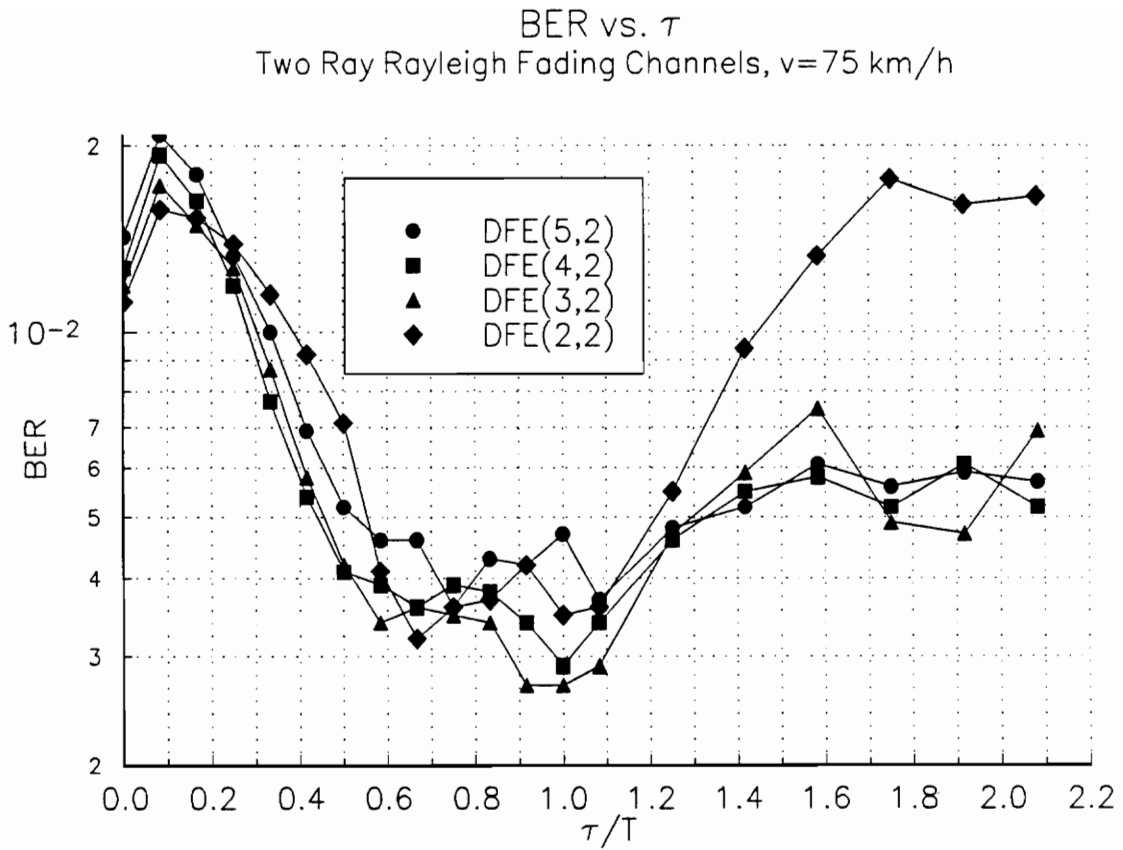


Figure 7.7-5. BER vs. τ for a two-ray Rayleigh fading channel with 5 different forward tap numbers of RLS-DFE and the number of feedback taps is 2. Carrier frequency is 850 MHz, $v=75$ km/h, $E_b/N_o=\infty$.

possible to get drastically different results. Larger sized equalizers do not necessarily have better performance, because they may add their own ISI.

8. SIRCIM Indoor Channel

8.1. SIRCIM

The Simulation of Indoor Radio Channel Impulse response Model (SIRCIM) software package developed by MPRG is based on extensive wideband channel measurements of different indoor environments at different frequency bands. A large measurement data base was collected in several different types of buildings. SIRCIM generates multipath delay profiles with the same ensemble statistics as measured data.

SIRCIM generates 19 impulse response profiles equally distributed ($\lambda/4$ spaced) over a 4.5λ track, and the $\lambda/4$ spacing between two profiles provides an adequate spatial resolution to repeat the fading characteristics of multipath components. The profiles have a maximum time delay of 500 ns, which is chosen according to the characteristics of indoor channels. It divides the multipath components into 64 time-bins within the 500ns, for a multipath resolution of 7.8 ns. SIRCIM simulates both line-of-sight (LOS) and obstructed (OBS) topographies, for open-plan factory buildings, soft-partitioned and hard-partitioned office buildings. SIRCIM creates both the power and the phase information so that it can be used in both channel characterization and bit-error simulation [35].

The SIRCIM model quantifies the number of multipath components as locally Gaussian distributed over a uniform distributed mean value. The probability of receiving a component with a particular excess delay decreases as the excess delay increases. The power within an excess delay is log-normally distributed about a predicted path loss, and the amplitudes of different multipath components are correlated over excess delay and distance along a straight track. The phases of the multipath components in the first profile are randomly chosen, and then deterministic phase changes are added to other profiles as the receiver moves along the 4.5λ track. Hence, SIRCIM offers the phase information of individual multipath components and creates CW fading that has a distribution similar to the measured data [35].

Users can specify an additive white Gaussian noise level to be added to the

profiles. However, noise is added to the multipath profiles, so users cannot extract noise from the channel profiles.

SIRCIM has proved to be a highly accurate UHF multipath radio channel simulator. It recreates both the large-scale and small-scale statistics of measured wideband impulse responses. It can be implemented as a channel model in communication studies and adaptive equalization.

8.2 Implementation of SIRCIM

The SIRCIM data cannot be used directly in channel simulations until some modifications are made. A SIRCIM profile has 64 samples over a 500 ns range corresponding to a resolution or sample period of 7.8 ns. However, in the equalization simulation, 12 samples are used in each symbol duration, so the sample period is $\frac{T_s}{12}$, which is not necessarily equal to 7.8 ns. In order to make SIRCIM profiles have the same sample period as the equalization simulation, the 500 ns delay is divided by the simulation sample period $\frac{T_s}{12}$; then, all the multipath components located within a specific time slot are added together as one sample. Therefore, the modified SIRCIM profiles have the same sample period as the simulation.

Another problem that needs to be solved is SIRCIM's low spatial resolution. SIRCIM's 19 profiles per 4.5λ are adequate to characterize the channel, but not sufficient for simulation, due to the high data rate of the simulated systems. In indoor environments where delay spreads are small, it is possible to use systems with a data rate up to megabits per second. Also, indoor channels vary very slowly. Therefore, millions of channel samples may be needed in a 4.5λ distance, and a huge number of impulse responses have to be added between two profiles of SIRCIM.

An effective method to increase the resolution is interpolation. By using a straight line to represent the values between the same time slots of two spatially-separated profiles, linear spline interpolation is the simplest to implement. However, it simplifies the channel variation, because the slopes of multipath components and CW variations will be constant between the two spatially-separated profiles. The simplification may have effects on the performance of

adaptive equalizers, which track the variation of a channel. Cubic spline interpolation is an effective way to smooth the variation [24].

Cubic spline interpolation utilizes a cubic polynomial to represent the variation between two profiles separated by $\lambda/4$. The polynomial may be represented as [24]

$$f_j(x) = a_j(x_{j+1} - x)^3 + b_j(x - x_j)^3 + c_j(x - x_j) + d_j(x_{j+1} - x) \quad (8.2.1)$$

where x_{j+1} and x_j are the locations of the two profiles surrounding x , and $f_j(x)$ is the amplitude at x . Coefficients a_j , b_j , c_j and d_j are obtained according to the sample values of the 19 profiles along a 4.5λ track at the same time slot.

For $j=1$ to 18,

$$a_j = \frac{f_j''(x_j)}{6h_j} \quad (8.2.2)$$

$$b_j = \frac{f_{j+1}''(x_{j+1})}{6h_{j+1}} \quad (8.2.3)$$

where $h_j = x_{j+1} - x_j$ and $f_j''(x_j)$ is the second derivative of f at x_j .

$$c_j = \frac{f_{j+1}}{h_j} - b_j h_j^2 \quad (8.2.4)$$

$$d_j = \frac{f_j}{h_j} - a_j h_j^2 \quad (8.2.5)$$

To get coefficients a_j , b_j , c_j and d_j , $f_j''(x_j)$'s in the following linear equation

$$\begin{bmatrix} 2h_0 & h_0 & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & h_j & 2(h_j+h_{j+1})h_{j+1} & & \\ & \cdot & \cdot & \cdot & & \\ & & & & h_{17} & 2h_{17} \end{bmatrix} \begin{bmatrix} f_0'' \\ f_1'' \\ \vdots \\ f_{j+1}'' \\ \vdots \\ f_{18}'' \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{j+1} \\ \vdots \\ \beta_{18} \end{bmatrix} \quad (8.2.6)$$

where

$$\beta_0=6 \left[\frac{\Delta f_0}{h_0} - f'(x_0) \right] \quad (8.2.7)$$

$$\beta_j=6 \left[\frac{\Delta f_j}{h_j} - \frac{\Delta f_{j-1}}{h_{j-1}} \right], \quad \text{for } j=1 \text{ to } 17 \quad (8.2.8)$$

$$\beta_{18}=6 \left[f'(x_{18}) - \frac{\Delta f_{17}}{h_{17}} \right] \quad (8.2.9)$$

$f'(x_0)$, $f'(x_{18})$ are the slopes of the start and end points that can be decided randomly.

To implement the cubic spline interpolation, the channel profiles are converted from the power and the phase components to the real and the imaginary components. Interpolation of real and imaginary components of the channel are done separately. The profiles are divided into time slots of $\frac{T_s}{12}$, and equations similar to (8.2.6) are set for the real and imaginary values of every time slot. After solving these equations for $f''_j(x_j)$ for every slot, coefficients a_j , b_j , c_j and d_j can be obtained using Equations (8.2.2) to (8.2.5) and saved in a data file. Each time a channel profile at a specific location is needed, the two profiles the location next to the location are determined. Then the a_j , b_j , c_j and d_j for all time slots in this area are applied into Equation (8.2.1) to get the real and imaginary values of the entire profile.

SIRCIM puts noise directly into the channel impulse response because it is based on the real world data which measure the signal and noise in the same profiles. However, when the channel is used in simulation, the impulse response and noise cannot be separated; hence, the noise is treated as a part of the channel and will be convolved with the signal. In other words, there is actually no noise effect in SIRCIM channels. Therefore, to generate SIRCIM channels for the adaptive equalization simulation, the noise level is always set as small as possible to avoid its influence on channel's characteristics. Then white Gaussian noise is added at the equalization simulation.

Whether to use the wideband (2.1.6) or CW channel (2.1.7) path loss is a question needed to be addressed before E_b/N_o is computed. Since an equalizer uses the wideband energy, it may make sense to use the wideband path loss. However, after some experiments, the mean wideband path loss and mean CW

path loss of SIRCIM channels were found very close. The CW channel path loss was used. The channel profiles are normalized by multiplying them by the mean channel path loss, so the normalized mean channel path loss is 0 dB. Hence, a specified E_b/N_0 can be obtained by adding white Gaussian noise at the receiver using (7.2.4).

8.3 Characteristics of Indoor Channels and Equalization Simulation

An indoor channel is different from an urban channel in that it usually has a very small rms delay spread and small temporal variation. Because of its small rms delay spread, indoor systems can use very high data rates without causing high BER. For instance, in an open-plan factory building, a typical σ_τ is about 100 ns and more than 90% profiles measured by Rappaport [3] were smaller than 150 ns. Then, if setting a criterion of $\frac{\sigma_\tau}{T_s} < 0.1$ at 90%, T_s can be as small as 1500 ns and the symbol rate can be as high as 0.67 million symbol-per-second (Msps). The symbol rate can be even higher for office buildings, which have smaller rms delay spreads. Hence, if equalizers are used, the indoor systems can use data rates up to several Mbps.

Indoor channels change very slowly, because of slow possible mobile velocities. As in (2.1.4)

$$f_D = \frac{v}{\lambda}$$

a small velocity causes a small Doppler frequency and slow variation in the channel. The normalized Doppler frequency (2.1.5) is further reduced because of the high symbol rates of indoor systems.

Both the high data rate and small Doppler frequency cause a very slow varying channel for indoor communication. This is favorable for an adaptive equalization, because the channel tracking error will be very small. However, a high data rate results in a high computational requirement. For example, assuming one iteration for an equalizer needs 100 operations and the symbol rate is 1 Msps, the processor must have the power of 100 MIPS. However, because of the slow channel variation, LMS algorithms or even stationary equalizers may be used which will lower the computational requirement.

The simulation in this work used DFEs with RLS algorithms. Several SIRCIM channels with different rms delay spreads were generated for equalization on different data rates (from 0.2 Mbps to 20 Mbps). DFEs with different sizes were tested to choose an optimal size that has the best BER performance.

SIRCIM offers indoor channels about 4.5λ . Because of the high data rate and slow variation of an indoor system, a huge number of bits can be sent over this distance. Suppose the symbol rate is 1 Msps and the receiver speed is 10 km/h (2.78 m/s), and the carrier frequency is 1.3 GHz, a total of 360,000 symbols can be sent along a 4.5λ track. If the symbol rate is chosen as 10 Msps, the number will be more than 3 million. However, both system and adaptive equalization simulations take a significant amount of computation time. Because of the limited computation power available, it takes too long to simulate so many bits. So, only a portion of the SIRCIM channel can be used. In order to obtain accurate results, a SIRCIM channel is evenly divided into 18 areas with a $\lambda/4$ length and the locations of the 19 profiles are the beginning and end points of these 18 areas. A small portion of each area is taken for the equalization simulation, and this portion is chosen as 1000 bits long, so 18,000 bits are used totally. Thus, the different parts of the channel with different rms delay spreads and path loss can be used to obtain a reasonably good result without excessive amount of computations.

8.4 Simulation Results

A SIRCIM channel named "HOBS2" for an open-plan building was generated. Since it is 100% OBS, which means there is no LOS path, the channel has a large rms delay spread $\sigma_\tau=138$ ns. Figure 8.4-1 shows the channel profiles, CW fading, rms delay spread distribution, and CW envelope distribution. The profiles show that there is a significant path at a delay of 500 ns. HOBS2 is chosen for equalization simulation because this channel is similar to the worst case impulse responses measured in [3].

In the simulation, a 10 Msps symbol rate was selected and the speed of the receiver was set to 10 km/h. QPSK modulation is used but no coherent detector is implemented before the equalizer. The symbol duration T_s is 100 ns and $\frac{\sigma_\tau}{T_s}=1.38$. The normalized rms delay spread, $\frac{\sigma_\tau}{T_s}$, which is the ratio of channel rms delay

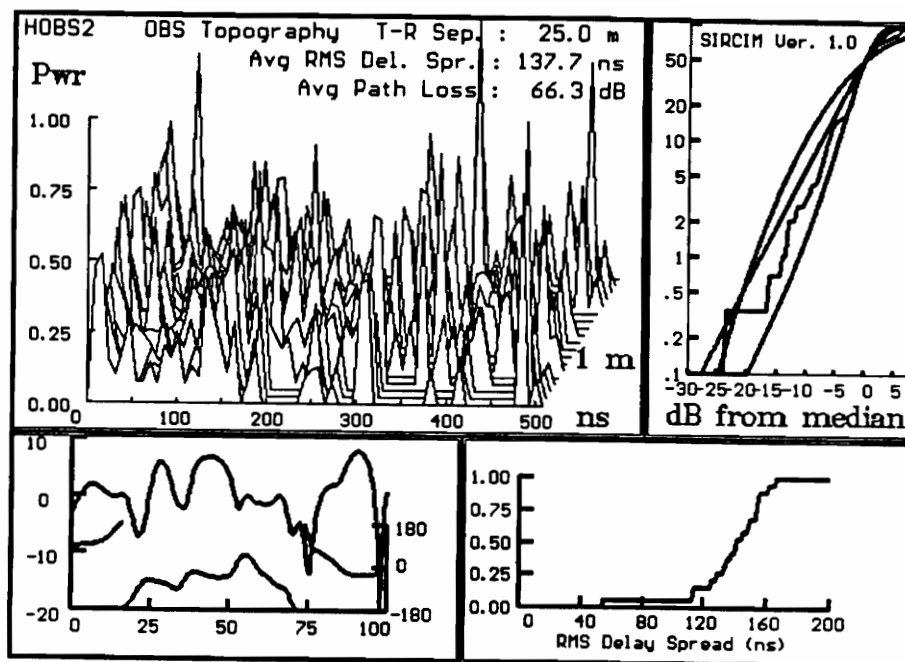


Figure 8.4-1. An indoor wide band impulse response named as HOBS2 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% Obstructed in an open-plan building, T-R separation is 25 m. The spatial mean rms delay spread of HOBS2 is 138 ns.

spread and the symbol duration, is excessively large, so adaptive equalization must be used to secure acceptable BER. Several equalizer sizes were evaluated in order to obtain the optimal size. A number of forward taps was set, then the number of feedback taps was changed to select the smallest number of feedback taps that generates the smallest BER. Figure 8.4-2 shows the results. Six was used as the number of forward taps and the number of feedback taps was tested from 3 to 6. From the graph, DFE(6,5) is clearly better than DFE(6,3) or DFE(6,4), but is similar to DFE(6,6). Therefore, 5 is the optimal number of feedback taps. Then, the number of feedback taps was set to 5 and the number of forward taps was selected using the same method, which looks for diminishing returns. From the results shown in Figure 8.4-3, 6 is the best number of forward taps. Therefore, DFE(6,5) is the optimal equalizer size for the channel HOBS2 at 10 Msps. Compared to the result without any equalization, which has a BER about 30%, using a DFE gives a significant improvement in the performance. The BER is kept below 10^{-3} at a E_b/N_o of 13 dB. The BER of a non-equalization QPSK receiver in an AWGN channel is also simulated and shown in the Figures 8.4-2 and 8.4-3. These results are the same as the theoretical results in [34]. The simulation shows that although an adaptive equalizer has a good performance in a frequency-selective fading indoor channel, it is still much worse than a receiver working in an ideal AWGN channel. The 3 to 6 dB difference is caused by the wideband fading of the indoor channel and the residual ISI in the equalizer. However, the performance of the equalizer is better than an unequalized receiver in a Rayleigh fading channel [23].

Different symbol rates were selected including 5 Msps ($\frac{\sigma_r}{T_s}=0.69$), 2 Msps ($\frac{\sigma_r}{T_s}=0.28$) and 1 Msps ($\frac{\sigma_r}{T_s}=0.14$). The method of first finding the point of diminishing returns for the number of feedback taps and then the point of diminishing returns for the number of forward taps was used to determine the best DFE sizes of these symbol rates for channel HOBS2. Figures 8.4-4 and 8.4-5 were used to choose the best sizes of the forward and feedback filters, and the results indicate that DFE(4,3) is the best size for 5 Msps. The results in Figures 8.4-6 and 8.4-7 show that DFE(3,2) is the optimal choice when the symbol rate is 2 Msps. Also, Figures 8.4-8 and 8.4-9 show the best size for a symbol rate of 1 Msps is DFE(2,2). The simulation results show that at these symbol rates a DFE

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=20Mbps

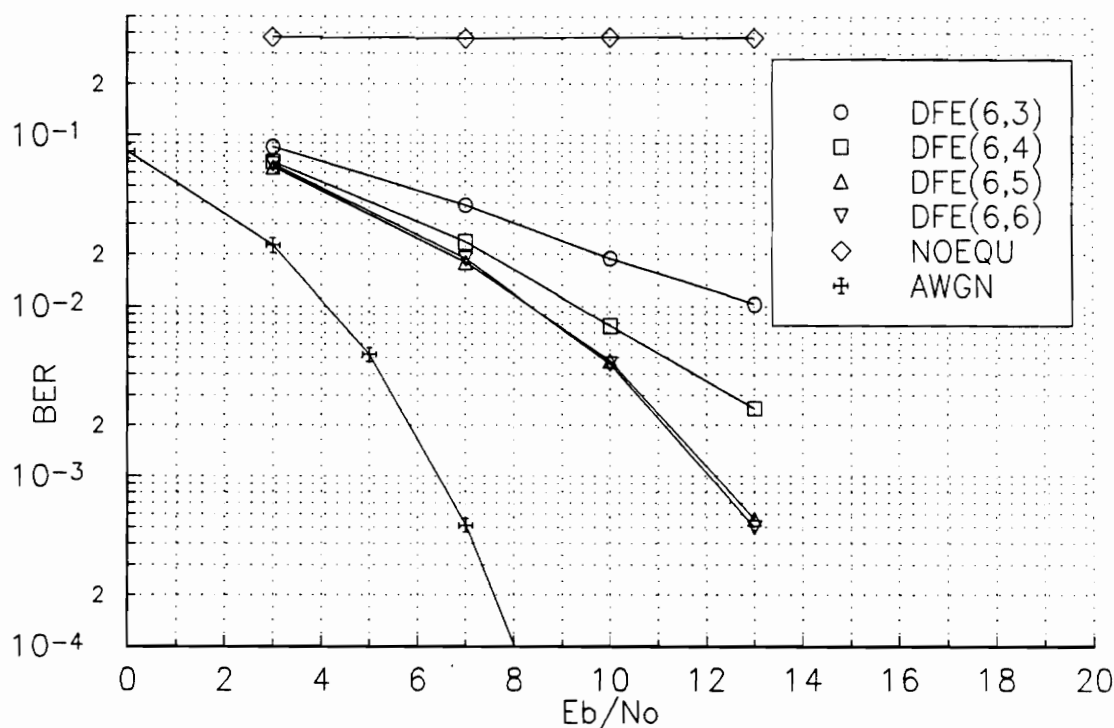


Figure 8.4-2. BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation and a 20 Mbps data rate.

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
Open-plan Building 100% OBS RMS=138ns, Data Rate=20Mbps

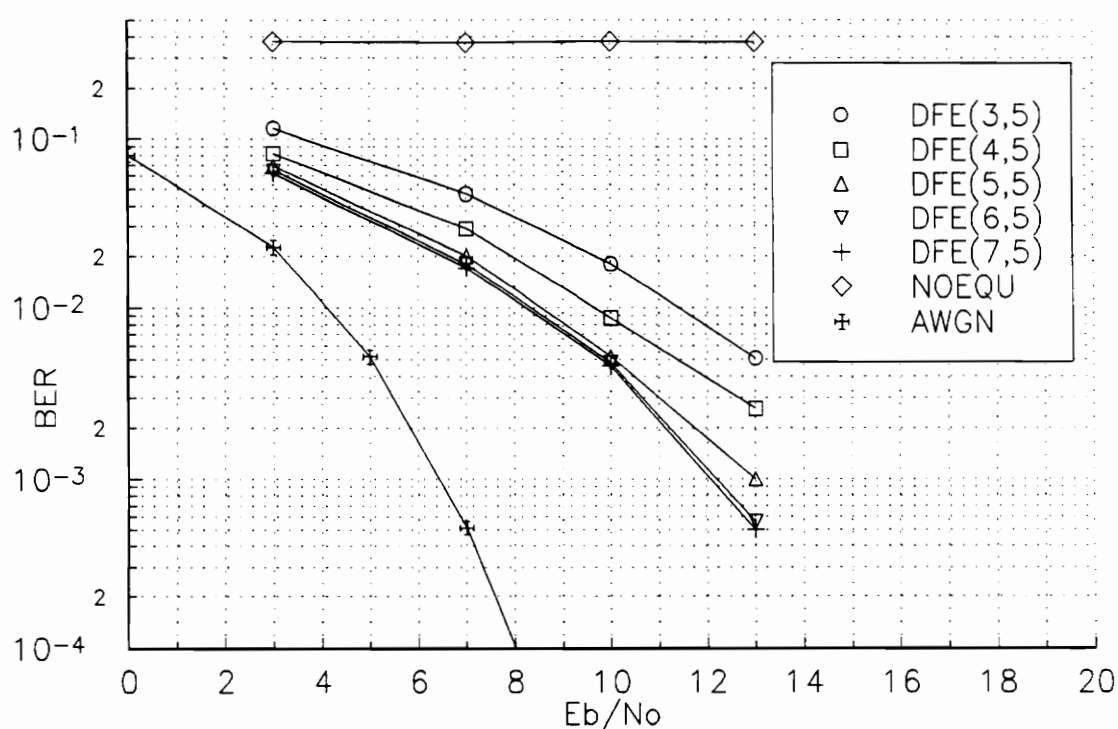


Figure 8.4-3. BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 20 Mbps data rate.

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=10Mbps

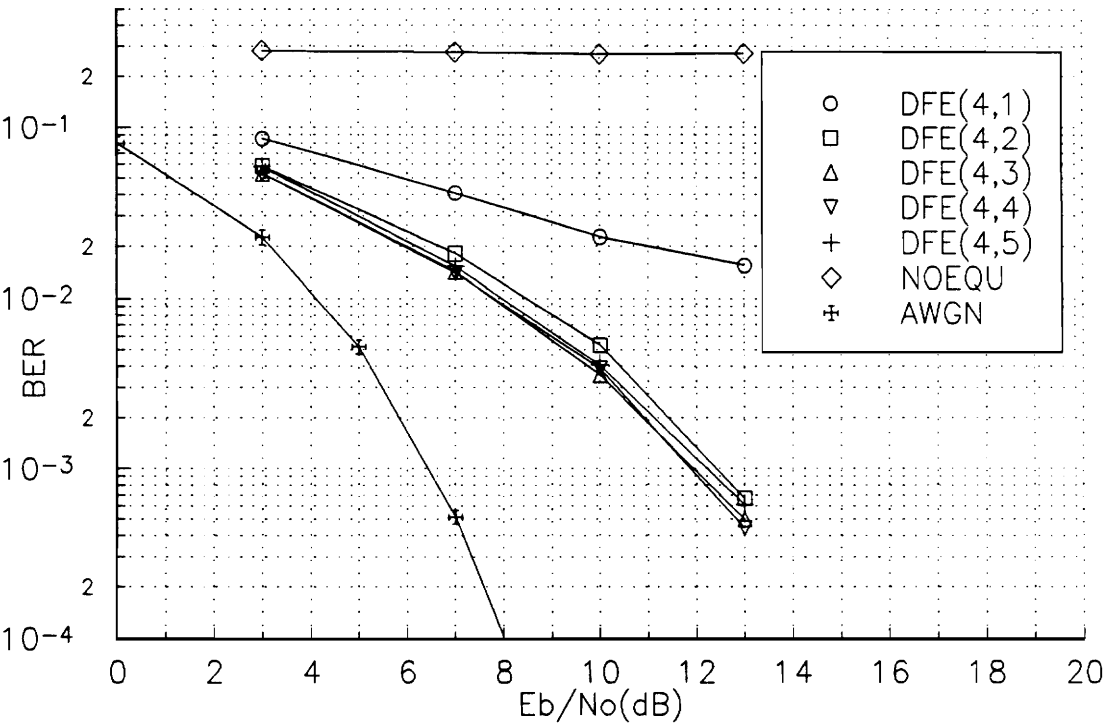


Figure 8.4-4. BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 10 Mbps data rate.

Ber vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=10Mbps

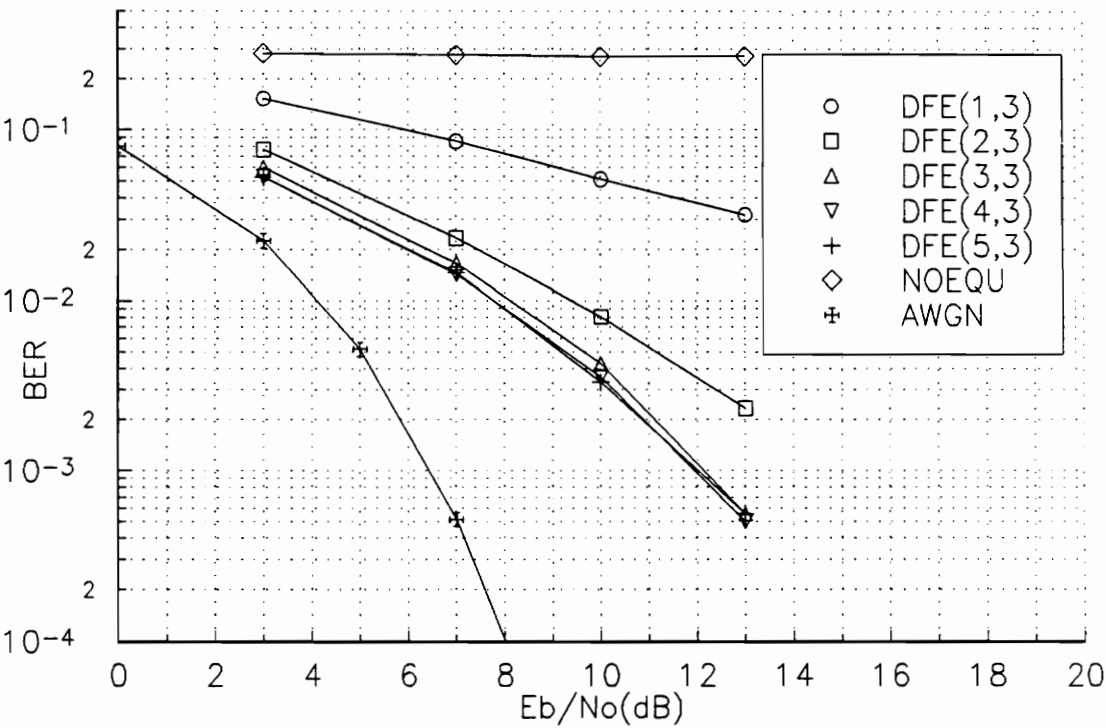


Figure 8.4-5. BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 10 Mbps data rate.

Ber vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=4Mbps

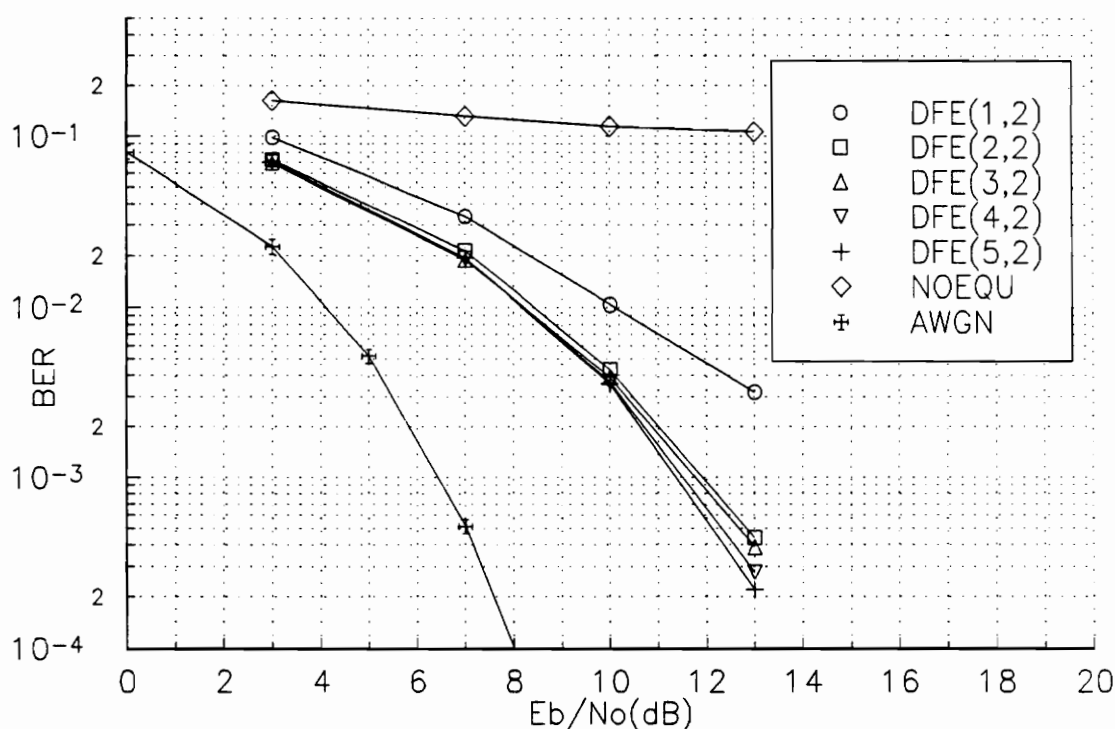


Figure 8.4-6. BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 4 Mbps data rate.

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
Open-plan Building 100% OBS RMS=138ns, Data Rate=4Mbps

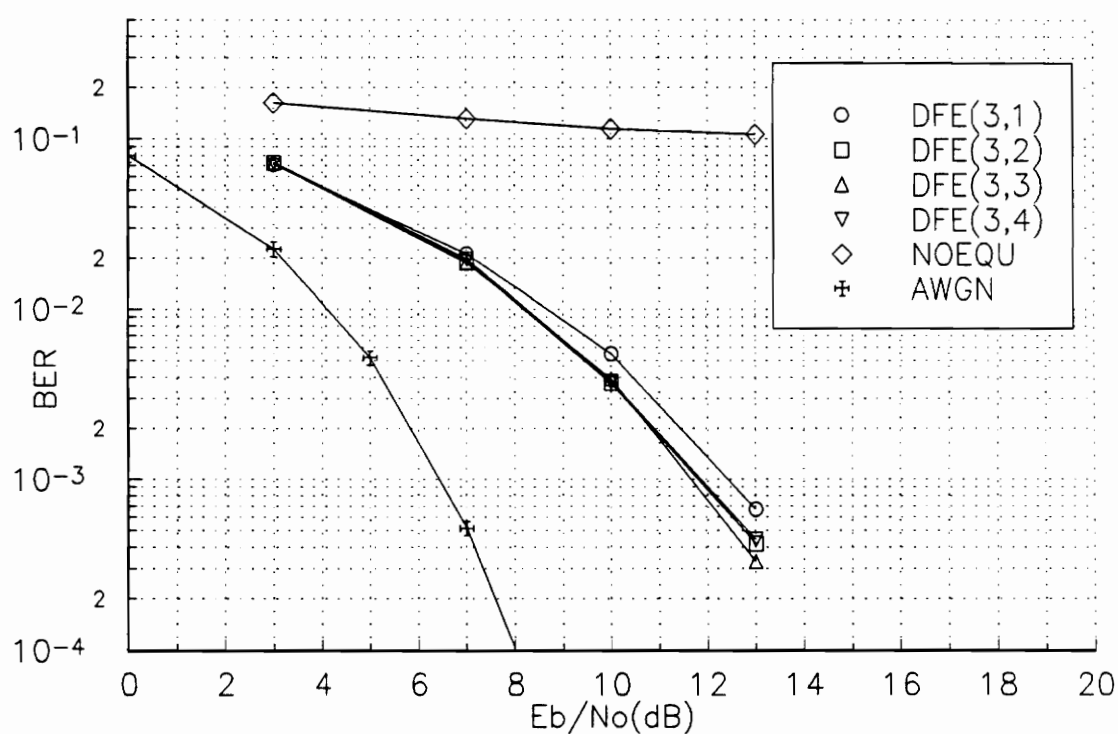


Figure 8.4-7. BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 4 Mbps data rate.

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=2Mbps

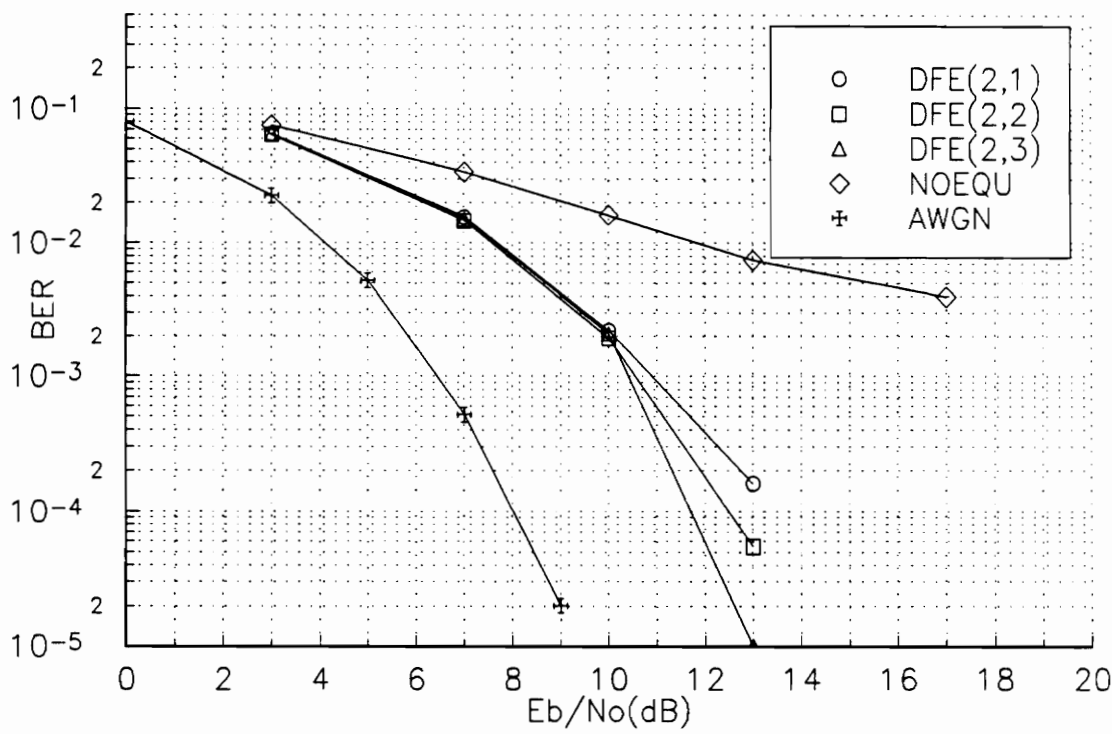


Figure 8.4-8. BER vs. E_b/N_o using different numbers of feedback taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 2 Mbps data rate.

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS2
 Open-plan Building 100% OBS RMS=138ns, Data Rate=2Mbps

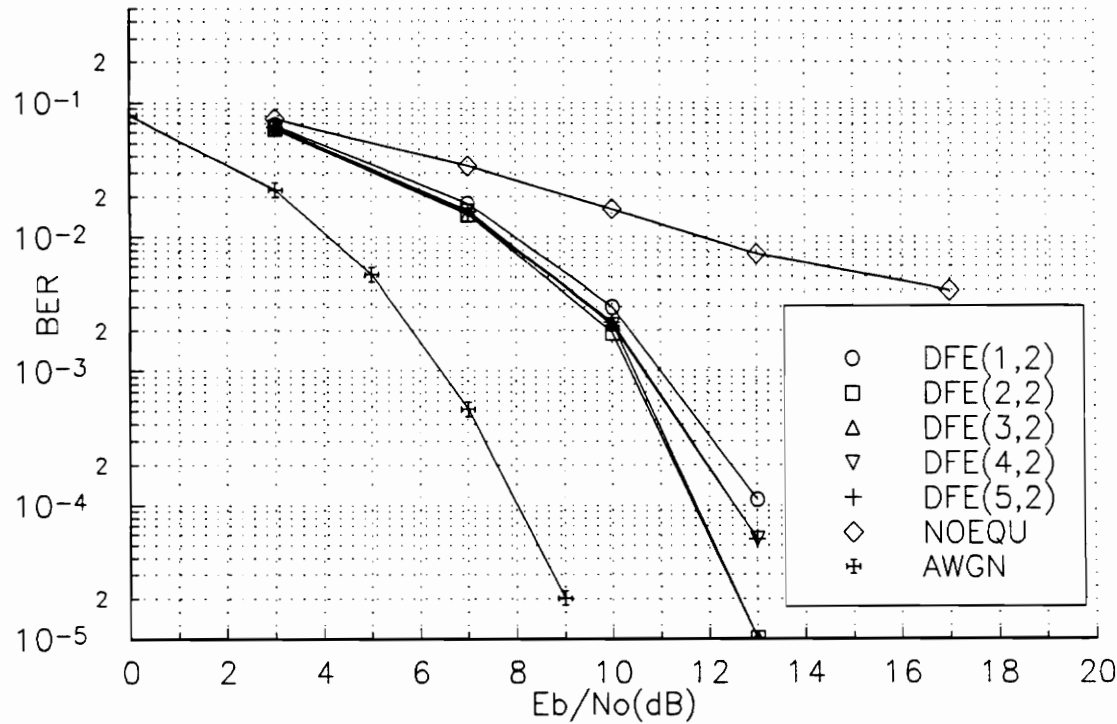


Figure 8.4-9. BER vs. E_b/N_o using different numbers of forward taps in an RLS-DFE in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation, and a 2 Mbps data rate.

works very well. The values of BER are below 10^{-3} at 13 dB E_b/N_o and will decrease with the increasing E_b/N_o . The curves for non-equalization and non-fading AWGN results are also displayed in the graphs. It is clear that adaptive equalization significantly increases the indoor system performance for these symbol rates, but it is still not close to the performance in an AWGN channel due to the wideband fading and residual ISI.

In order to determine when the DFE can no longer improve the BER, equalization at lower symbol rates was also tested. The results in Figure 8.4-10 indicate when the symbol rate is smaller than 0.5 Msps ($\frac{\sigma_\tau}{T_s}=0.07$), the adaptive equalization provides very limited improvement compared to non-equalization. When the symbol rate is 0.1 Msps, there is very small ISI ($\sigma_\tau=0.012$). In this case, a system without equalization is better than the one with an equalizer although the difference is very small.

Other SIRCIM channels with different rms delay spreads were also generated. LOS4 in Figure 8.4-11 is an open-plan building channel with 100% LOS and has a mean σ_τ of 95.6 ns. HARDLOS2 in Figure 8.4-12 is a hard-partition building 100% LOS channel and has a 33.2 ns mean σ_τ . Different symbol rates were used to evaluate the performance of the adaptive equalization on these channels. The optimal DFE sizes determined for the HOBS2 channel were also used for the equalization of both LOS4 and HARDLOS2 channels. Their own optimal sizes may be different due to the smaller σ_τ .

Figure 8.4-13 shows the equalizer performance at different symbol rates in channel LOS4 and the results without equalization and the optimal AWGN results. When the symbol rate is 1 Msps ($\frac{\sigma_\tau}{T_s}=0.1$), the BER without equalization remains larger than 0.01. But, if using a DFE(2,2), the BER is below 10^{-3} at 13 dB E_b/N_o . The symbol rate of the system has very limited effect on the equalizer's performance as long as the equalizer is designed for the worst case delay spread. It does, however, play a significant role in the BER without equalization.

Equalization performance in the channel HARDLOS2 is shown in Figure 8.4-14. Although the mean σ_τ for HARDLOS2 is only 33.2 ns, which is three times less than in LOS4, the DFE maintains good performance. The

BER vs. E_b/N_o in SIRCIM Indoor Channel HOBS 2
Performance of DFE with low data rate

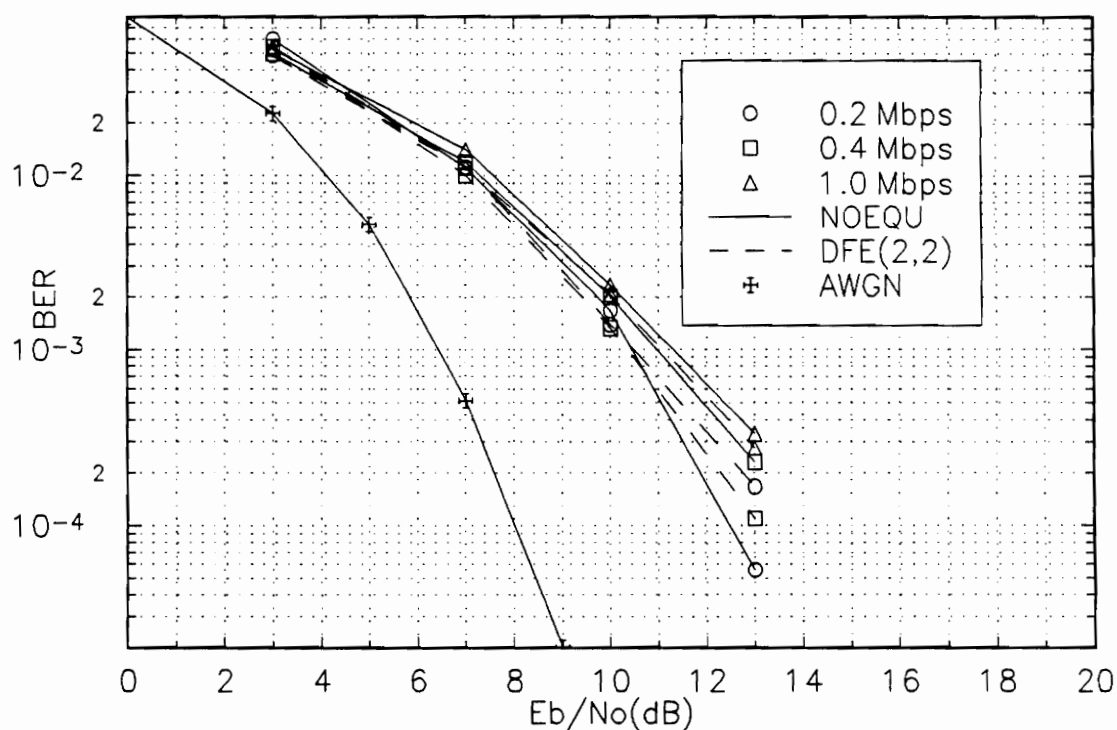


Figure 8.4-10. BER vs. E_b/N_o at different low data rates using RLS-DFE(2,2) in SIRCIM indoor channel HOBS2, which has a spatial mean RMS delay spread of 138 ns. The simulation uses QPSK modulation.

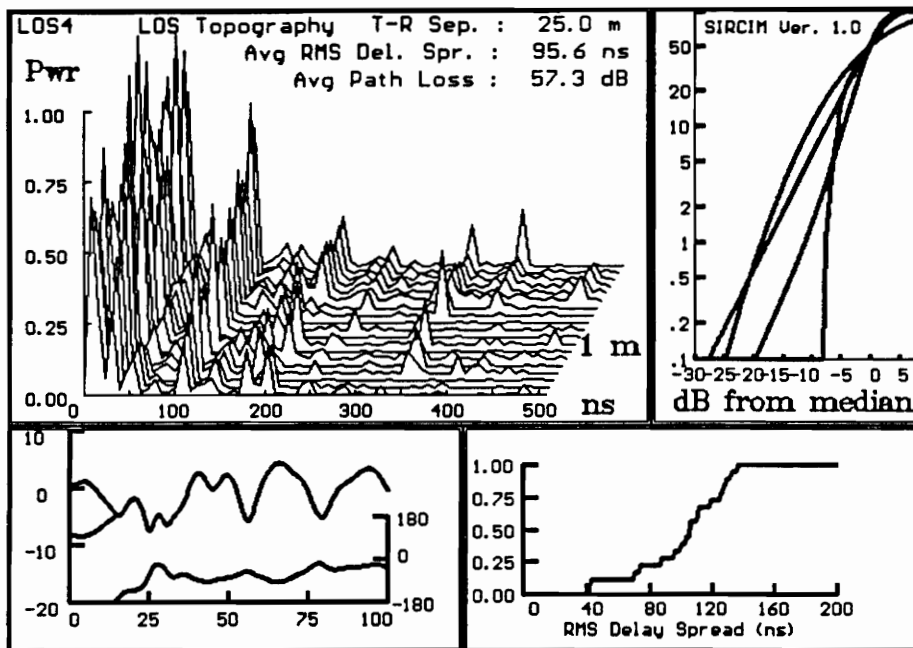


Figure 8.4-11. An indoor wide band impulse response named as LOS4 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% line-of-sight in an open-plan building, and the T-R separation is 25 m. The spatial mean rms delay spread of LOS4 is 95.6 ns.

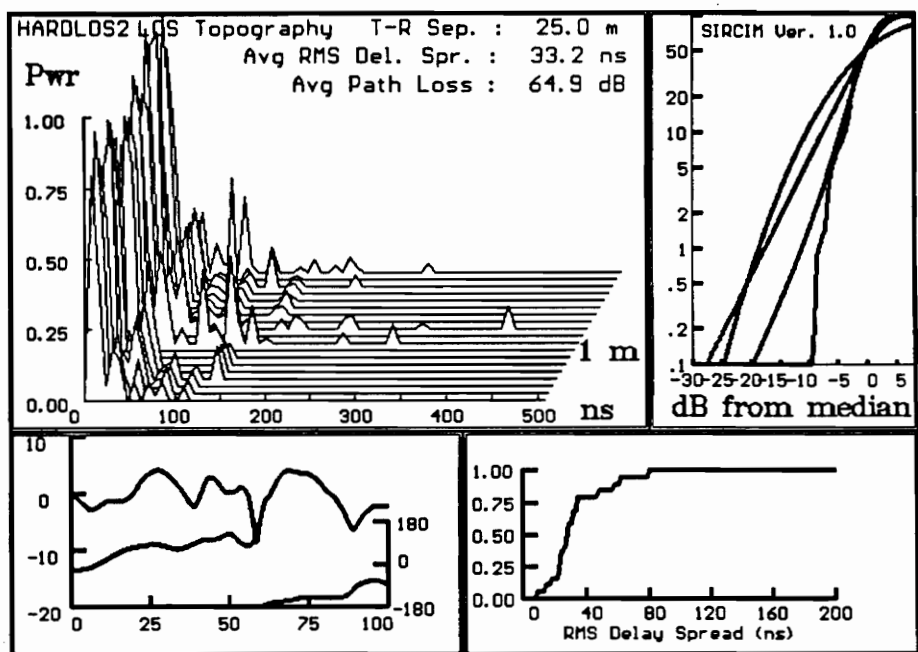


Figure 8.4-12. An indoor wide band impulse response named as HARDLOS2 simulated by SIRCIM at 1.3GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. The channel is simulated as 100% line-of-sight in a hard partitioned building, and the T-R separation is 25 m. The spatial mean rms delay spread of HARDLOS2 is 33.2 ns.

BER vs. E_b/N_0 at SIRCIM Indoor Channel LOS4
Open-plan Building 100% LOS RMS=95.6 ns

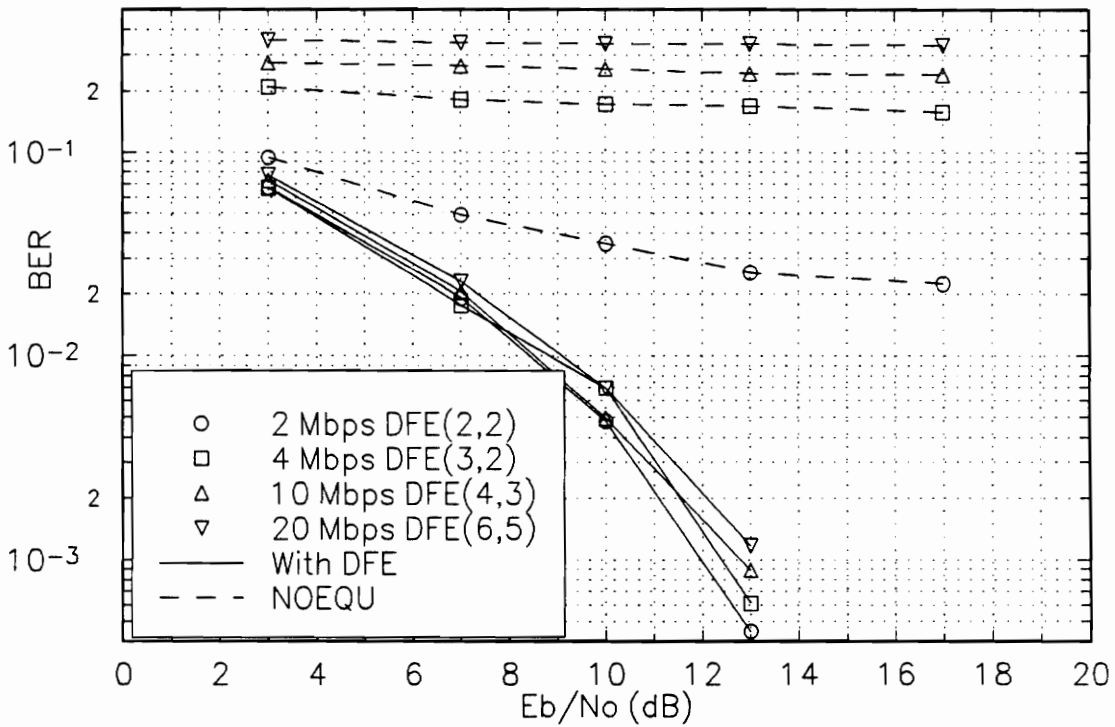


Figure 8.4-13. BER vs. E_b/N_0 at different data rates for SIRCIM indoor channel LOS4 using the optimal RLS-DFE sizes obtained from the optimization of channel HOBS2. LOS4 has a spatial mean RMS delay spread of 95.6 ns. The simulation uses QPSK modulation.

BER vs. E_b/N_o at SIRCIM Indoor Channel Hardlos2
Hard-partion Building 100% LOS RMS=33.2 ns

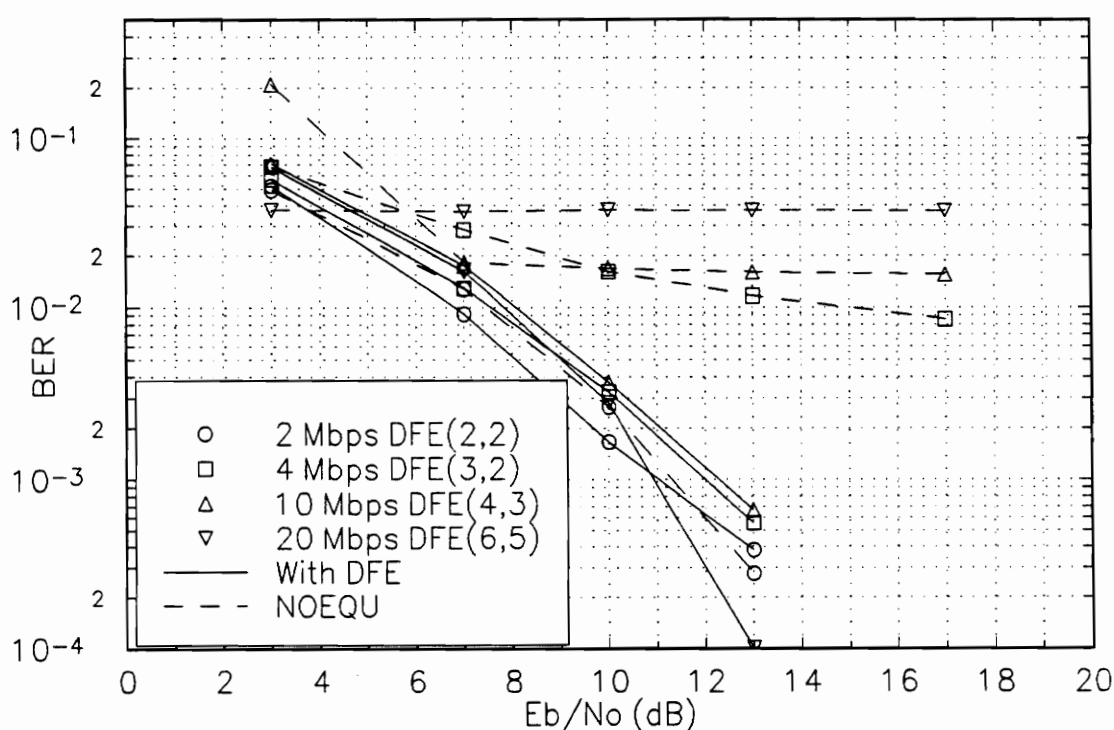


Figure 8.4-14. BER vs. E_b/N_o at different data rates for SIRCIM indoor channel HARDLOS2 using the optimal RLS-DFE sizes obtained from the optimization of channel HOBS2. HARDLOS2 has a spatial mean RMS delay spread of 33.2 ns. The system uses QPSK modulation.

improvement over non-equalization is not as significant as in the channel HOBS2 in Figures 8.4-9, 8.4-10 and LOS4 in Figure 8.4-13. At 1 Msps symbol rate ($\frac{\sigma_r}{T_s}=0.03$), the results with and without equalization are very close.

Comparing the equalization results for the three indoor channels, it is apparent that the equalizers have very similar performance in different channels and at different symbol rates. Therefore, indoor equalization systems have very consistent performance regardless of σ_r . Since the channel changes very slow, the channel tracking error is trivial.

The simulation results show that adaptive equalization can improve the BER by a large margin as long as there exists some ISI. Equalization can even improve the BER performance at $\frac{\sigma_r}{T_s}$ as small as 0.03. The reason of the superior performance of adaptive equalization for indoor channels is the slow channel variation and the high data rate.

9. SMRCIM Urban Channel

9.1 SMRCIM

To evaluate the performance of adaptive equalization in real world channels, the Simulation of Mobile Radio Channel Impulse response Models (SMRCIM) was developed. It is being disclosed to Virginia Tech Intellectual Properties, Inc. in December, 1991 and is based on measurements performed in several American and European cities [2,36]. The measurements emphasized the worst multipath propagation situations in urban area, so the statistics may be different from measurements in other urban environments.

According to the measured data, 5% of locations have rms delay spreads larger than $7.0 \mu\text{s}$, 10% have rms delay spreads larger than $6.0 \mu\text{s}$ and 50% have rms delay spreads larger than $2\mu\text{s}$. Because of the limitation of the information available, some parameters are unknown, so some assumptions have to be made. As more information becomes available, SMRCIM can be improved.

SMRCIM was developed from SIRCIM and uses SIRCIM's software structures and basic ideas [35]. Assuming a fixed transmitter, SMRCIM produces impulse response power and phase profiles equally spaced by $\frac{\lambda}{4}$. The total length of a consecutive channel run can be as long as 20 meters. The T-R separation can be selected from 100 meter to 20 km in urban environments. The maximum delay is $40 \mu\text{s}$. Since there is little possibility that a significant path exists after a delay of $40 \mu\text{s}$ based on measured data in [2,36], the maximum delay of $40 \mu\text{s}$ is believed large enough. SMRCIM divides the profiles into 64 time-bins within the $40\mu\text{s}$ time span to provide a multipath resolution of 625 ns. The carrier frequency range of SMRCIM is from 600MHz to 4 GHz.

SMRCIM uses the same methods as SIRCIM but different statistics to quantify the number of multipath components in a profile, the probability of receiving a component at a particular excess delay, and the path loss of a multipath component. SMRCIM calculates the phases of different multipath components deterministically using the same method as SIRCIM [35].

Figure 9.1-1 is an urban channel named "W4" generated by SMRCIM.

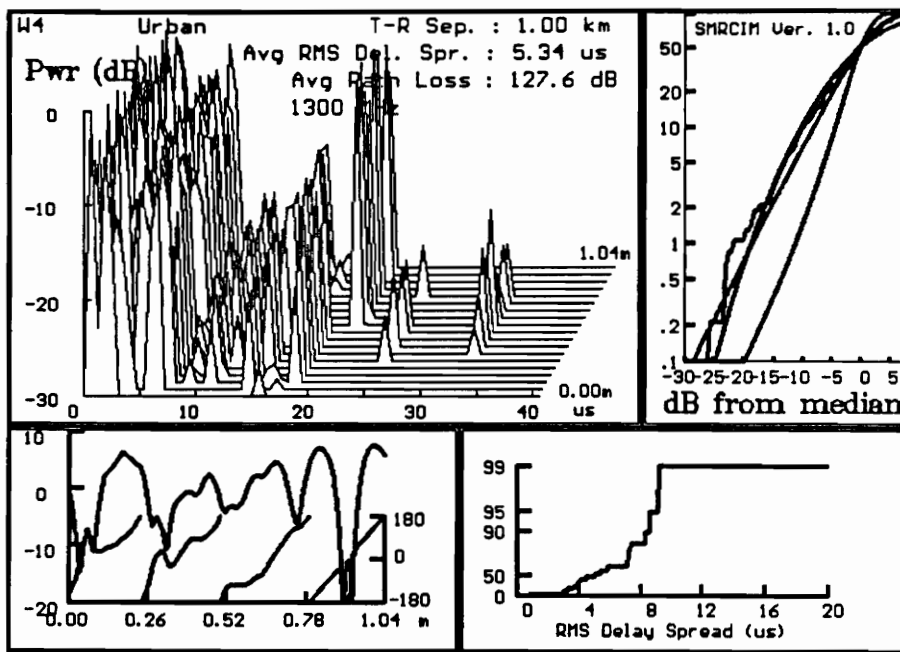


Figure 9.1-1. An urban wide band impulse response named as W1 simulated by SMRCIM at 1.3 GHz. Also shown are its distribution of the rms delay spread, narrow band fading, and narrow band signal power distribution. T-R separation is 1 km. The spatial mean rms delay spread is 3.09 ns.

The channel has a T-R separation of 1 km, and has a mean σ_r of 5.34 μs and an average path loss of 127.6 dB. The figure also shows the channel's narrowband fading and the amplitude distribution of the CW signal.

To evaluate the accuracy of SMRCIM, 80 SMRCIM channels were generated for the urban environment at 1.3 GHz and the length of each channel was 4.5λ or 1.04 meters. SMRCIM also computed the rms delay spread distribution and the displayed the distribution in Figure 9.1-2. The median σ_r value is 1.78 μs and the maximum σ_r is 17.16 μs ; also 10% of all points have σ_r larger than 6 μs . When compared to the data measured by Rappaport et al [2] in four American cities, the rms delay spread distribution of SMRCIM is close to the measured data.

The CW envelope distribution of a 10 meter SMRCIM channel is shown in Figure 9.1-3. Its amplitude distribution fits the Rayleigh fading curve very well except at 0.1 to 0.2%. Hence, SMRCIM has good agreement with measured data, and it is a valuable tool for the development of cellular and wireless systems.

9.2 Equalization of SMRCIM channels

To implement SMRCIM for adaptive equalization, cubic spline interpolation was also used to increase the spatial resolution of SMRCIM (see Section 8.2). The delay profiles are quantized into 12 samples per symbol duration, so they are compatible with equalization simulations. The channel path loss was normalized so that the mean channel path loss is 0 dB.

Coherent QPSK with 48.6 kbps was used as the modulation. DFE(2,1) with an RLS algorithm is used since the multipath components are always within one symbol duration, 41.15 μs . Compared to the choices for indoor systems, this data rate is very small. Also, the outdoor environment usually changes very fast, so urban channels are fast time-varying as opposed to indoor channels.

9.3 Results

Ten SMRCIM channels at 1.3 GHz with σ_r from 1 to 3 μs were selected for the evaluation of the DFE(2,1) at speeds of 75 km/h and 20 km/h. Results of four of these channels with and without equalization are shown in Figure 9.3-1 in

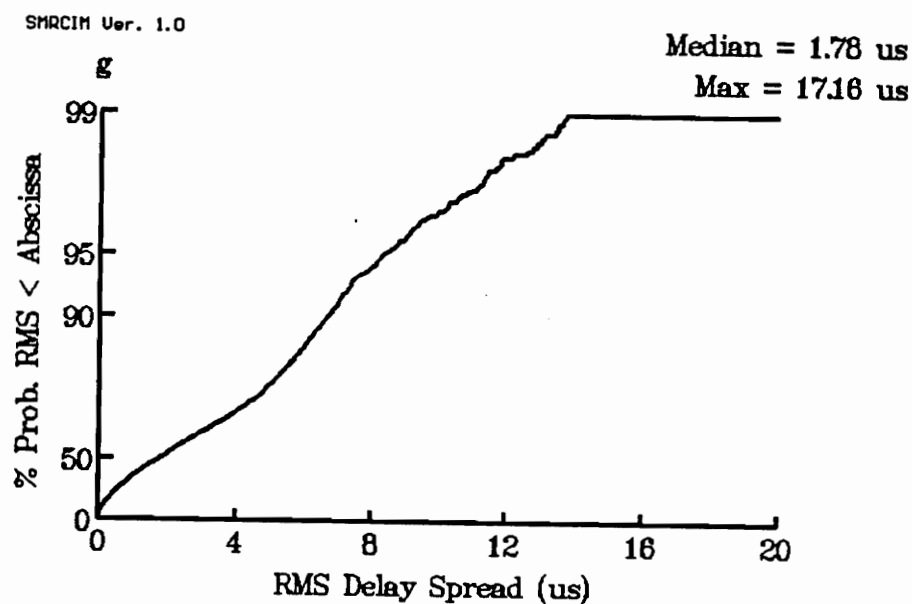


Figure 9.1-2. The rms delay spread distribution of 100 urban channels generated by SMRCIM. The median rms delay spread is 1.78 μ s, and the maximum rms delay spread is 17.16 μ s.

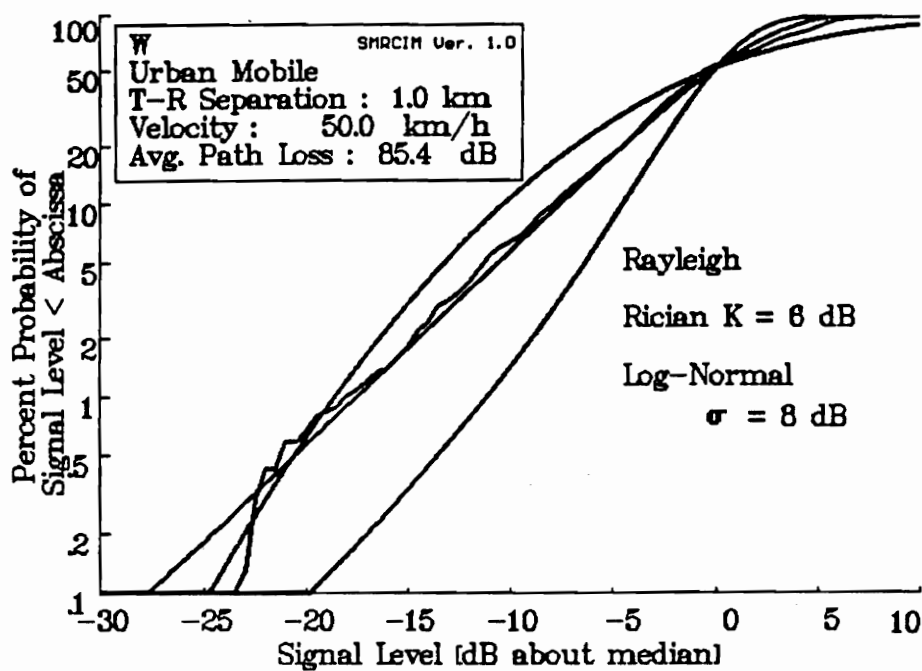


Figure 9.1-3. The narrow band signal power distribution of a 10 meter channel generated by SMRCIM.

Equalization for SMRCIM Channels rms delay spread from 1 to 3 μ s

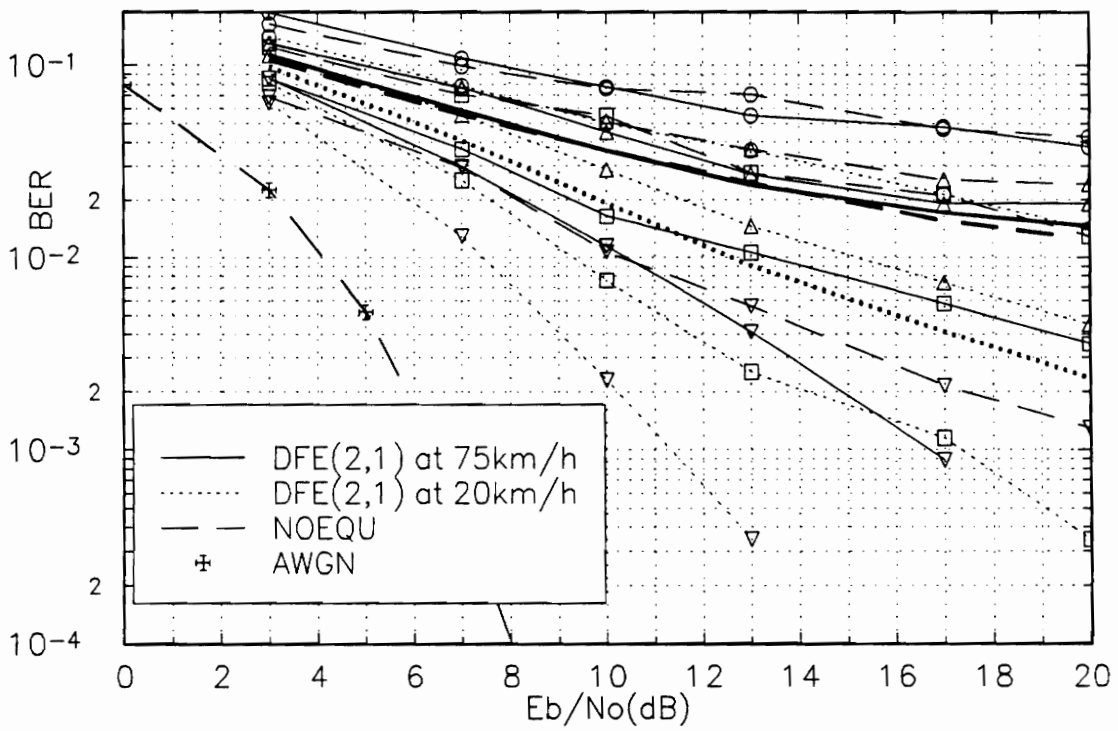


Figure 9.3-1. BER vs. E_b/N_o at several channels generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h. The spatial mean rms delay spreads of these channels are from 1 to 3 μ s. Bold curves are the mean BER.

the form of BER vs. E_b/N_o . The BER results of the 10 channels were averaged and plotted as bold curves in the graph. The results show that in channels with a small σ_τ , using equalization sometimes is better, and sometime is worse than non-equalization at the speed of 75 km/h. The graph also shows the average BER values are virtually the same with or without equalization at 75 km/h vehicle speeds. However, at 20 km/h, adaptive equalization shows its advantage, since the channels change much slower. For example, Figure 9.3-1 shows that at 75 km/h, neither a DFE(2,1) or unequalized system can do better than about 2×10^{-2} at $E_b/N_o=20$ dB. Yet at 20 km/h velocity, a DFE(2,1) can do better than 10^{-2} BER at 13 dB E_b/N_o .

Another 10 SMRCIM channels at 1.3 GHz with rms delay spreads from 6 to 10 μ s were also generated and evaluated in the same way. The results are shown in Figure 9.3-2. In this case, using adaptive equalization provides much better results even at 75 km/h. The average BER at 20 dB E_b/N_o using equalization is 4 times lower than the results without equalization. At 20 km/h, the BER results are even lower. It is clear that the BER performance of a system without equalization is highly dependent on σ_τ (a larger σ_τ causes higher BER). However, σ_τ does not affect a DFE that much. Compared to the results in Figures 9.3-1 and 9.3.2, the average performances of equalization in the two σ_τ regions are very close.

Ten random SMRCIM channels at 1.3 GHz which have σ_τ ranging from 0.5 μ s to 10 μ s were also used to determine if adaptive equalization improves the system performance. The mean BER performance for mobile speeds of 20 km/h and 75 km/h are shown in Figure 9.3-3. The results indicate that at 75 km/h, the adaptive equalization does not improve the BER for urban channels, and it has an overall error rate of more than 1% at 20 dB E_b/N_o . At a lower speed, however, an adaptive equalizer can significantly reduce the BER as shown in Figure 9.3-3.

Although σ_τ is very important to the system performance without equalization, it has very limited effect on an adaptive equalizer. The equalization performances are diversified in the channels with similar σ_τ values, so there are other factors that affect the equalization performance. However, the speed of a mobile plays a significant role in adaptive equalization in an urban channel. And the results show that at 1.3 GHz, adaptive equalization cannot improve a USDC

Equalization for SMRCIM Channels rms delay spread from 6 to 10 μ s

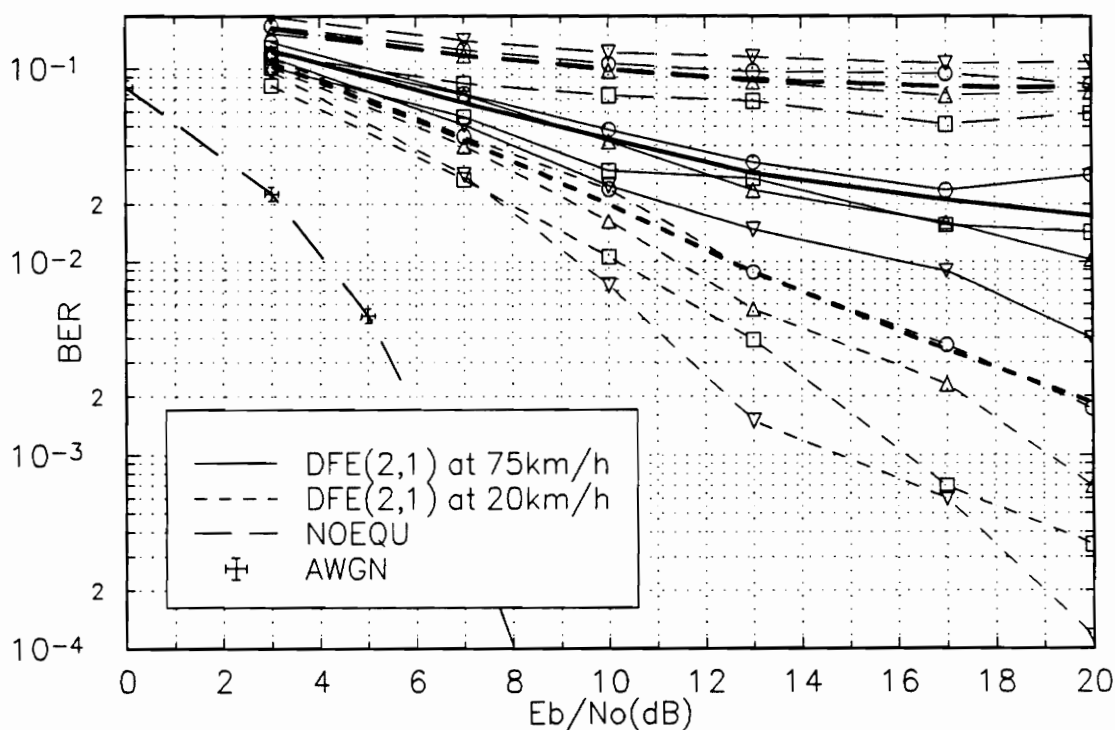


Figure 9.3-2. BER vs. E_b/N_o at several channels generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h. The spatial mean rms delay spreads of these channels are from 5 to 10 μ s. Bold curves are the mean BER.

Equalization on SMRCIM Channels Mean Performance of 10 Random Channels

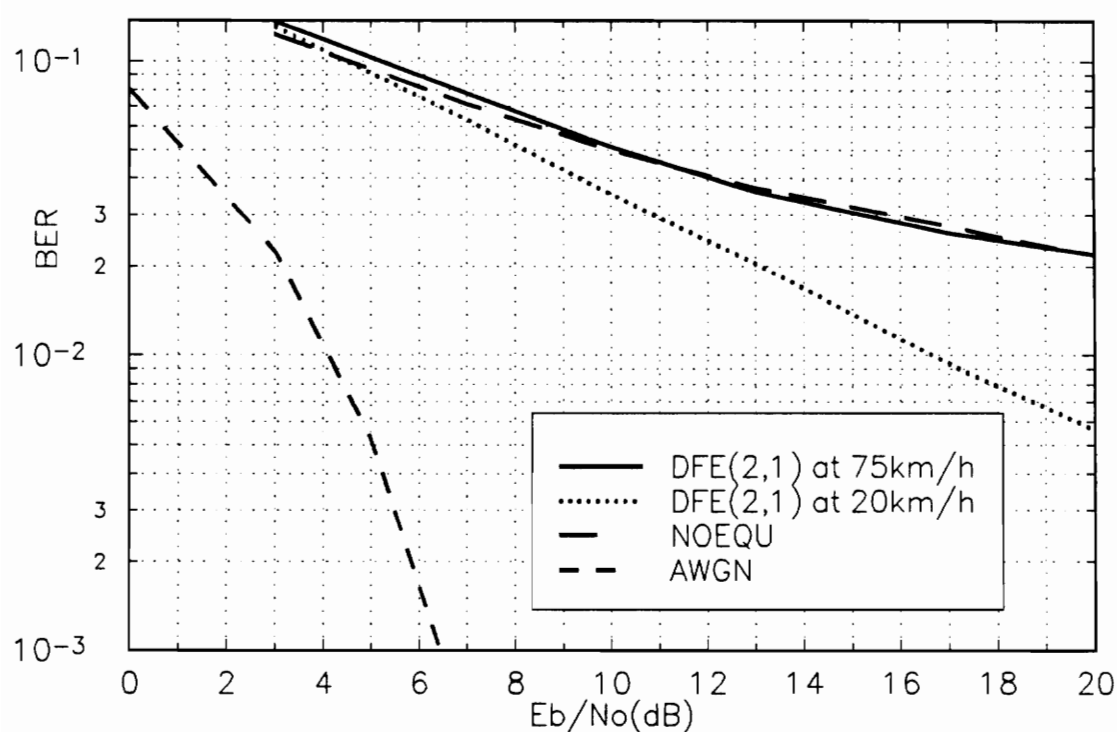


Figure 9.3-3. Mean BER vs. E_b/N_o at 10 channels randomly generated by SMRCIM using RLS-DFE(2,1) at 20 km/h and 75km/h.

system BER at a receiver velocity larger than 75 km/h.

To transform the results of 1.3 GHz to 850 MHz, which is the frequency band used for cellular telephones, let the Doppler frequency remain the same:

$$f_D = \frac{v_1}{\lambda_1} = \frac{v_2}{\lambda_2} \quad (9.3.1)$$

Then

$$\begin{aligned} v_2 &= \frac{\lambda_2 \times v_1}{\lambda_1} = \frac{f_1 \times v_1}{f_2} \\ &= \frac{1300}{850} \times 75 = 115 \text{ km/h} \end{aligned} \quad (9.3.2)$$

Therefore, at the 850 MHz band, adaptive equalization cannot improve a USDC system BER at a receiver velocity larger than 115 km/h.

10. A Novel Equalization Structure for Differential Modulations

10.1 Differential Modulation

All the equalization simulations done in this research were for coherent modulations; however, differential modulation techniques such as $\pi/4$ DQPSK are widely used because their receivers do not require a coherent detector, so the receivers are easy and inexpensive to build. For coherent modulation techniques such as QPSK, information is sent through the absolute phases of the signals:

$$\begin{aligned} f(n) &= \text{Re}[A e^{j\omega n T + \theta_n + \theta_0}] \\ \text{if } s=00, \theta_n &= -\frac{3\pi}{4} \\ \text{if } s=01, \theta_n &= \frac{3\pi}{4} \\ \text{if } s=10, \theta_n &= -\frac{\pi}{4} \\ \text{if } s=11, \theta_n &= \frac{\pi}{4} \end{aligned} \quad (10.1.1)$$

where s is a symbol consisting two bits, $f(n)$ is the channel input, A is the amplitude of the signal, ω is the carrier frequency, T is the symbol duration, θ_0 is the carrier's initial phase, and θ_n is the phase shift used to send the signal.

To detect the information, the receiver must be able to recover the exact phase of the carrier, θ_0 . Also, because θ_0 changes constantly due to the movement of the transmitter, the receiver and the medium, the receiver must be able to track its variation to receive the signal correctly. Therefore, coherent detection techniques such as the coherent phase locked carrier must be used.

Different from coherent modulation techniques, the differential phase shift keying technique sends information by the value of the phase shift between consecutive symbols. The instantaneous phase of $\pi/4$ DQPSK obeys the following rules: [24]

$$\begin{aligned} f(n) &= \text{Re}[A e^{j\omega n T + \theta_n}] \\ \text{if } s=00, \theta_n &= \theta_{n-1} - \frac{3\pi}{4} \\ \text{if } s=01, \theta_n &= \theta_{n-1} + \frac{3\pi}{4} \end{aligned}$$

$$\begin{aligned} \text{if } s=10, \theta_n &= \theta_{n-1} - \frac{\pi}{4} \\ \text{if } s=11, \theta_n &= \theta_{n-1} + \frac{\pi}{4} \end{aligned} \quad (10.1.2)$$

All the variables in (10.1.2) are the same as for QPSK in (10.1.1).

By detecting the phase difference of the two consecutive symbols $\theta_n - \theta_{n-1}$, the value of the second symbol can be identified using the rules in (10.1.2). Using noncoherent techniques, the original carrier phase θ_0 does not have to be recovered, and only the instantaneous phase difference is needed. Thus, the receiver is simple to build.

10.2 Equalization in $\pi/4$ DQPSK

Differential modulation techniques make the receivers simple, but they also make adaptive equalization harder to implement. Little work has been published on how to develop a method to implement adaptive equalization for differential modulations.

Generally, coherent techniques are used to implement adaptive equalization for differential modulations. An equalizer is used as a coherent detector to recover the exact phase, although a coherent detector can be used instead.

Figures 10.2-1 (a) and (b) show the block diagram for the implementation of adaptive equalization for $\pi/4$ DQPSK. The equalizer is located before the differential decoder. The differential decoder is located between the equalizer and the decision maker. The input of the forward filter is the output of the channel (after RF and IF stages), which is denoted by A in the figures. The output of the equalizer after the decision maker is denoted by D , and D is recoded using the same differential coder as the receiver. Therefore, B is the recoded signal, which is used as the input of the feedback filter. The error used to update the equalizer is between B and the output of the equalizer before the differential decoder, C .

The receiver detects the data by obtaining the phase difference between the equalizer's current and previous outputs. Using threshold detection, one of the four phase values in (10.1.2) is chosen, and the symbol output is detected according the demodulation code. The phase shift is added to the accumulated

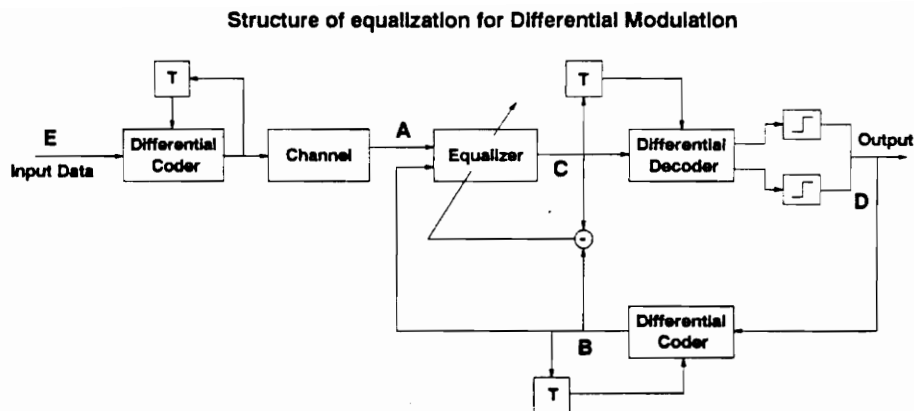


Figure 10.2-1(a). Structure of implementing adaptive equalization to differential modulations

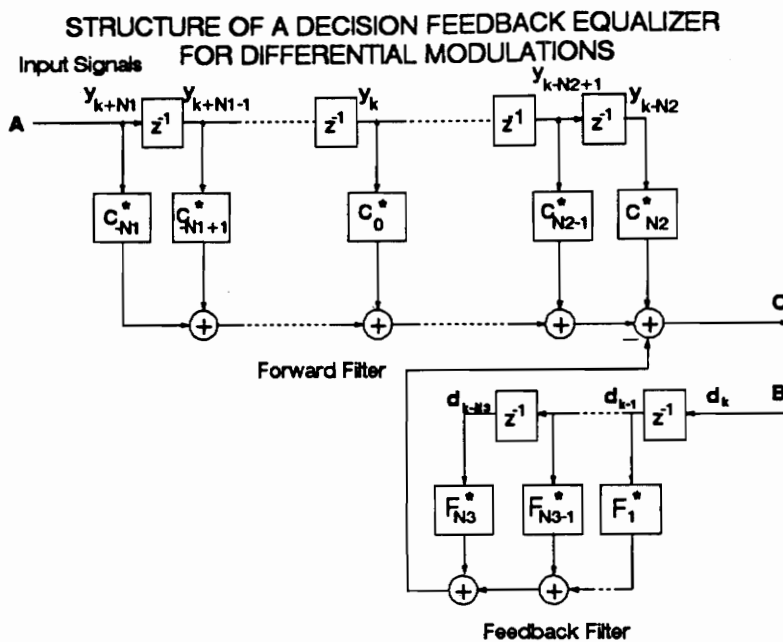


Figure 10.2-1(b). Structure of a decision feedback equalizer in a differential modulation system.

data being fed into the feedback filter. The accumulated data does not necessarily have the same phase as the channel input, because only differential phase shift is important. First a random initial phase is set for the accumulated data, then according to the training or detected signal, a phase shift is added to the initial phase using the coding rules for the modulation scheme.

At the beginning, the transmitter needs to send a training sequence, which is known to the receiver. During the training period, the equalizer converges to the channel. After the training, the receiver starts to receive data and at the same time updates itself to track the variation of the channel.

All adaptive equalizers such as LTE, DFE and lattice equalizers discussed earlier can be used in this structure and the LMS and the RLS algorithms can be applied to update the equalizer.

This structure works fine when perfect decision-making is assumed, which means using E instead of D as the input to the differential coder at the receiver (see Figure 10.2-1). However, if detected data D is fed in, once an error appears, the equalizer will lose tracking. The equalizer comes back to normal after the appearance of a huge block of errors. The reason is that once an error appears, there is a false phase shift in the phase of the decision. Since the differential decoder added the wrong phase shift to the previous symbol's phase, there is always an error in the phase after the error appears. Therefore, the equalizer has to adjust itself to the new phase which takes time, so it causes more errors and the equalizer collapses. This situation is called error propagation. One of the methods to solve the error propagation problem is retraining such as in TDMA systems where the equalizer is retrained at the beginning of each data block. Also, coding and diversity can be used to reduce number of the errors.

10.3 Simulation and Results

A DFE with an RLS algorithm using the structure in Figure 10.2-1 was used in the simulations. To test how well this structure works, several mobile channels were used in the evaluation of the adaptive equalization for $\pi/4$ DQPSK. The two-ray Rayleigh fading channel was the first to be implemented.

Figure 10.3-1 shows how a differential $\pi/4$ DQPSK DFE with an RLS

BER vs. τ of Different Speeds
At Two Ray Channel of $\pi/4$ DQPSK DFE(3,2)

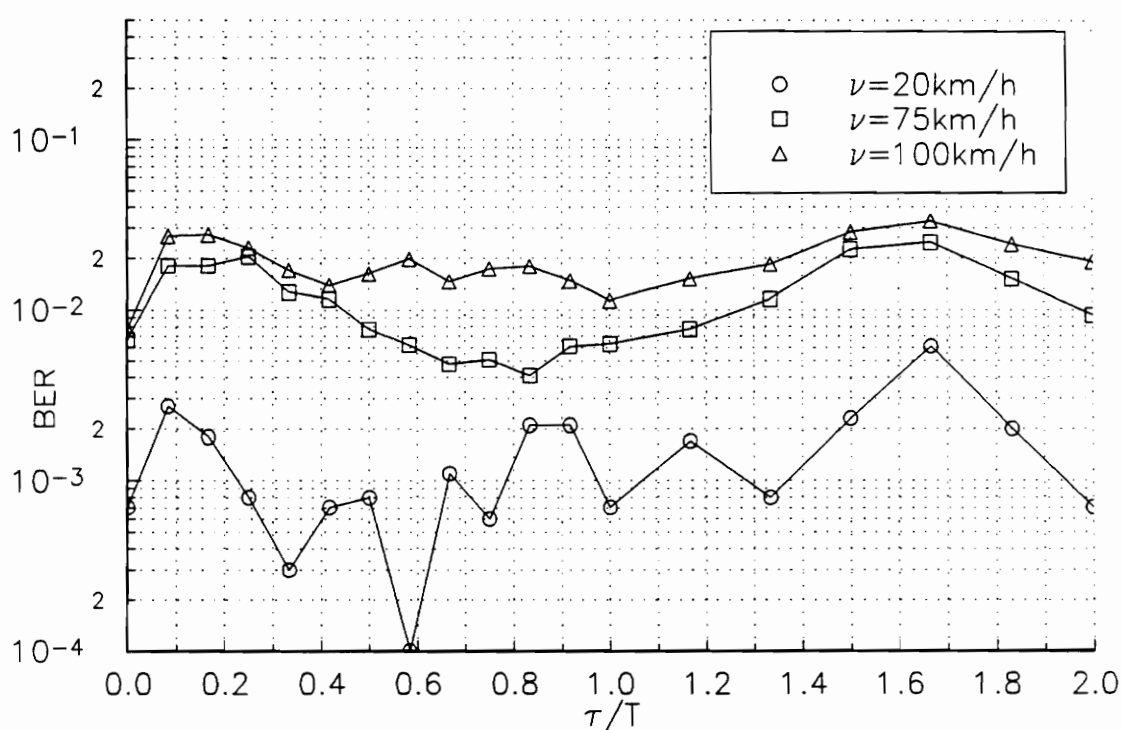


Figure 10.3-1. BER vs. τ of two-ray Rayleigh fading channel using RLS-DFE(3,2) and $\pi/4$ -DQPSK at different speeds. Carrier frequency is 850 MHz.

algorithm works in a two-ray Rayleigh fading channel. The performance pattern is similar to that of the QPSK in Figure 7.7-1. A DFE works better in a slower channel, and its performance is the worst with a τ from $0.1T$ to $0.3T$. However, compared to equalization performance in QPSK (see Figure 7.7-1), equalization results in $\pi/4$ DQPSK are not as good as for the QPSK except when $\tau=0$. When $v=100$ km/h, the equalizer for QPSK has a BER less than 10^{-2} at most values of τ , but for $\pi/4$ DQPSK, BER is always larger than 10^{-2} except at $\tau=0$. When $v=75$ km/h, the equalizer for QPSK generally has better performance, and its lowest BER is about 2.5×10^{-3} at $\tau=T$, while for $\pi/4$ DQPSK, the lowest BER is more than 4×10^{-3} at $\tau=0.83T$. The performance difference at 20 km/h is larger than at 75 km/h and 100 km/h. The simulated BER results for an unequalized $\pi/4$ DQPSK receiver in an AWGN channel is also presented. These results agree with the theoretical results [34].

The indoor SIRCIM channel HOBS2 is also used to evaluate the adaptive equalization for $\pi/4$ DQPSK. The optimal DFE sizes for QPSK were used here, and the simulation results are shown in Figure 10.3-2. The results are much better than those for the urban channel in Figure 10.3-1, and the BER can be maintained around 10^{-3} or less at 20 dB E_b/N_o . However, compared to the results for QPSK in Figures 8.4-2 to 8.4-9, the BER for $\pi/4$ DQPSK is still higher. For instance, at 20 Mbps, the BER of QPSK at 3 dB E_b/N_o is about 6.5×10^{-2} , and about 5×10^{-4} at 13 dB E_b/N_o , while for $\pi/4$ DQPSK, the BER is about 1.2×10^{-1} at 3 dB E_b/N_o , and about 3×10^{-3} at 13 dB E_b/N_o , which are significantly larger than for QPSK. Also, there are several dB loss compared to the results for an AWGN channel due to the wideband fading and residual ISI. The unequalized bit-error simulation results for SIRCIM indoor channels can be found in [23,42].

The BER difference between the two modulations is caused by the inherited weakness of differential modulation in order to achieve a simpler receiver [34]. Without adaptive equalization, the BER for $\pi/4$ DQPSK is poorer than QPSK too.

From the simulation results presented here, adaptive equalization for differential modulations using the structure proposed here basically has the same characteristics as the coherent modulation methods, but the performance is not as good.

Differential Equalization in SIRCIM Indoor Channel HOBS2 $\pi/4$ -DQPSK

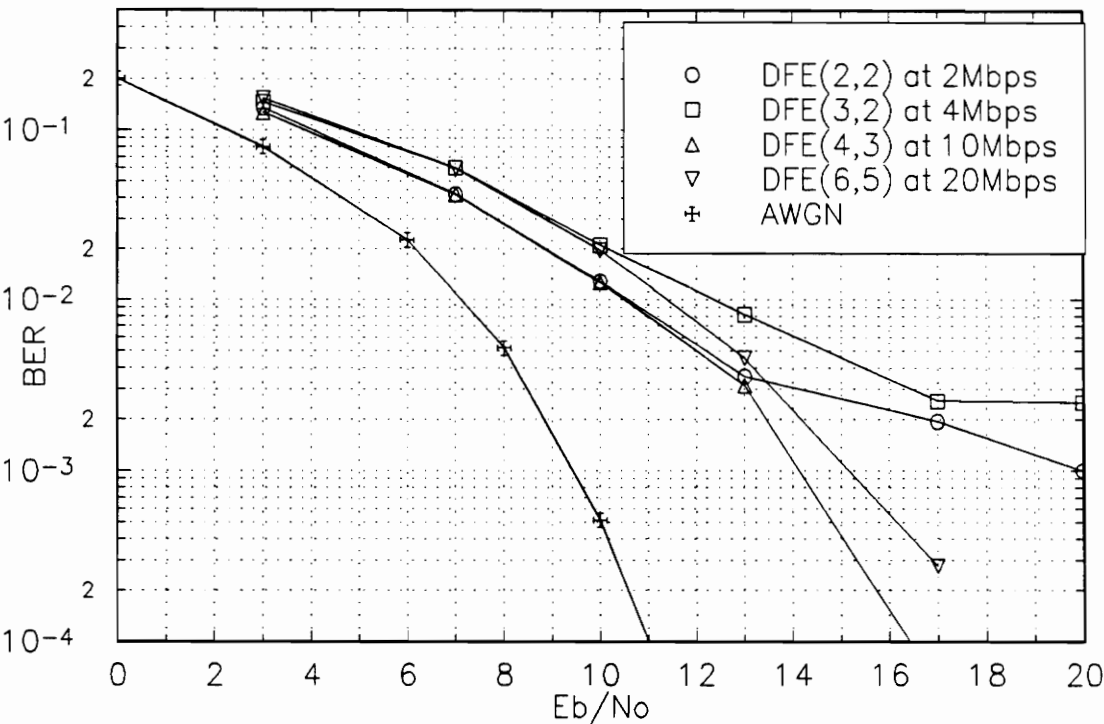


Figure 10.3-2. BER vs. E_b/N_0 of SIRCIM channel HOBS2 using RLS-DFE using $\pi/4$ -DQPSK at different speeds, and the sizes of the DFE are obtained from QPSK modulation. Carrier frequency is 1.3 GHz.

11 Conclusion

Adaptive equalization techniques to mitigate multipath induced intersymbol interference, and methods of implementing adaptive equalization in mobile communication systems have been studied. Among the different adaptive equalizer structures presented and compared in this thesis, the decision feedback equalizer, lattice decision feedback equalizer, and maximum likelihood sequence estimator perform the best, and therefore are concluded to be the best equalizer structures for mobile systems. This is similar to the conclusions reached in [19,20,21].

Least mean square algorithms and recursive least square algorithms, the two most widely used algorithms, were studied to determine their properties. RLS algorithms are theoretically and experimentally proved to converge much faster and also have better tracking ability than LMS algorithms. However, LMS algorithms need much less computation to update the equalization. Therefore, an LMS algorithm is more appropriate for slowly varying channels. On the other hand, if the channel changes very fast, RLS algorithms are needed, or the equalizers will not be able to track the channel variation.

The performance of adaptive equalizers in urban mobile channels was evaluated. The urban channels are simulated by two-ray Rayleigh fading models. Also, a real world channel impulse response simulator, SMRCIM was developed for the equalization simulation. The simulation results show that, in urban mobile channels, the performance of an equalizer is highly dependent on the speed of the channel variation. Faster channel variation results in a higher bit-error-rate.

From the results presented, an adaptive equalizer does not necessarily improve the performance of a system. The conditions for an adaptive equalizer to improve BER are that the channel has some delay spread and the channel has a slow variation. Therefore, a switch that turns the equalizer on when the delay spread is large should be built at the receiver. The σ_r value at which the equalizer should be turned on is dependent on the variation of the channel; the

slower the variation, the smaller the σ_r can be. Without a switch, an adaptive equalizer for a USDC system will not improve the BER at 1.3 GHz, unless the receiver speed is smaller than 75 km/h, and for 850 MHz, the receiver speed should be less than 115 km/h.

The optimal size of an equalizer depends on the rms delay spread of the channel. A channel with large σ_r channel needs a large equalizer. For USDC channels, DFE(3,2) is a good choice. An equalizer that is either too large or too small will affect the performance of the equalizer.

SIRCIM indoor channel models are used to simulate indoor mobile channels. Indoor channels have small rms delay spreads, so the data rates for indoor systems can be very high compared to urban systems, and the variation of an indoor channel is much slower than an urban channel. The slow channel variation and high data rate result in good performance of the adaptive equalization. However, a high data rate needs a high computational requirement.

Finally, an equalizer structure for the differential modulations was proposed. It coherently detects the differential signals. The simulation results shows that, although it has the similar properties of an equalizer in a coherent modulation, the performance is not as good as a coherent modulation.

Future work

Although adaptive equalization has been available for about 3 decades, many problems need to be solved. The performance in low data rate and high speed mobile channels is dismal because of the channel tracking error. Therefore, algorithms with a better tracking ability is needed. The high computational requirements, especially for the high data rate indoor systems, limits the applications of adaptive equalization, and designers usually avoid using equalizers because of their high cost and also power consumption.

To avoid the problem, LMS algorithms may be utilized for indoor systems in the future. Although an LMS algorithm converges very slowly and has poor tracking ability, it is possible to implement it for indoor systems which have very slow changing channels. Other faster convergence methods introduced in Chapter 3 can improve the convergence, and after the convergence, the LMS algorithm can

be used instead.

One other suggestion for future research is using the interpolation to deal with the high computational requirements. In TDMA systems, there is usually a training sequence at the beginning of a block, so optimal equalizer coefficients can be obtained using the training sequence. Also, another set of equalizer coefficients can be obtained using the training sequence of the next block. Then, instead of using an adaptive algorithm, the interpolation is used to get the equalizer coefficients for each symbol received between the two training sequences.

In a real world system, a coherent detector should be implemented before the equalizer for the QPSK system. Also, the receiver will not synchronize to the first pulse but to the mean excess delay of the channel or the point the receiver detected the maximum power [34]. Therefore, future simulation should use a coherent detector before the equalizer, and also the receiver should detect the maximum power as a real world receiver instead of the first pulse.

References:

- [1] J.G. Proakis, *Digital Communications*, McGraw-Hill 1983.
- [2] T.S. Rappaport, S.Y. Seidel, R. Singh, "900 MHz Multipath Propagation for U.S. Digital Cellular Radiotelephone." *IEEE Transactions on Vehicular Technology*, vol.39, No.2, pp. 132-139, May 1990.
- [3] T.S. Rappaport, "Characterization of UHF Multipath Radio Channels in Factory Building" *IEEE Antennas and Propagation Magazine*, vol.37, No.8, pp.1058-1069, August 1980
- [4] R.W. Lucky, "Automatic Equalization for Digital Communication" *Bell System Technology Journal*, vol. 44, pp.547-588, April 1965.
- [5] D.A. George, R.R. Bowen, J.R. Storey, "An Adaptive Decision Feedback Equalizer", *IEEE Transactions on Communications*, vol.COM-19, No.3, pp.281-293, June 1971.
- [6] F. Ling, J.G. Proakis, "Nonstationary Learning Characteristics of Least Squares Adaptive Estimation Algorithms," *Proceedings ICASSP84*, San Diego, California, pp.3.7.1-3.7.4, 1984.
- [7] D. Godard, "Channel Equalization Using a Kalman Filter for Fast Data Transmission," *IBM Journal of Research and Development* pp.267-273, May 1974.
- [8] D.D. Falconer, L. Ljung, "Application of Fast Kalman Estimation to Adaptive Equalization," *IEEE Transactions on Communications*, vol.COM26, pp.1439-1446, October 1978.
- [9] J.M. Cioffi, T. Kailath, "Fast, Recursive-Least-Squares Transversal Filters for Adaptive Filtering" *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol.ASSP-32, pp.304-337, 1984.
- [10] E.H. Satorius, S.T. Alexander, "Channel Equalization Using Adaptive Lattice Algorithms" *IEEE Transactions on Communications*, vol.COM.27, pp.899-905, June 1979.
- [11] E.H. Satorius, J.P. Pack, "Application of Least Squares Lattice Algorithms to Adaptive Equalization" *IEEE Transactions on Communications*, vol.COM.29, pp.136-142, February 1981.
- [12] F. Ling, J.G. Proakis, "A Generalized Least Square Lattice Algorithm and

Its Application to Decision-Feedback Equalization," *Proceedings of ICASSP*, Paris, France, pp.1439-1466, May 1982.

- [13] F. Ling, J.G. Proakis, "Lattice Decision-Feedback Equalizers and Their Application to Fading Dispersive Channels," *Proceeding of International Conference on Communications*, Boston, MA, pp.C8.2.1-C8.2.5, June 1983.
- [14] F. Ling, J.G. Proakis, "Adaptive Lattice Decision-Feedback Equalizers Their Performance and Application to Time-Variant Multipath Channels," *IEEE Transactions on Communications* vol.COM.33, No.4, pp.348-356, April 1985.
- [15] G.D. Forney, Jr., "Maximum-likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Transactions on Information Theory*, vol. IT-18, pp.363-378, May 1972.
- [16] G. Ungerboeck, "Adaptive Maximum-likelihood Receiver for Carrier-Modulated Data-Transmission Systems," *IEEE Transactions on Communications*, vol.COM-22, No.5, pp.624-635, May 1974.
- [17] F.R. Magee, Jr., J.G. Proakis, "Adaptive Maximum-Likelihood Sequence Estimation for Digital Signaling in the Presence of Intersymbol Interference," *IEEE Transactions on Information Theory*, pp.120-125, January 1973.
- [18] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, 1986.
- [19] A. Narasimhan, S. Chennakeshu, J.B. Anderson, "An Adaptive Lattice Decision Feedback-Equalizer for Digital Cellular Radio," *Proceeding of IEEE VTC 90*, pp.662-667, 1990.
- [20] S. Chennakeshy, A. Narasimhan, J.B. Anderson, "Decision Feedback Equalization for Digital Cellular Radio," *Proceeding IEEE International Conference on Communications*, Atlanta, pp.1492-1496, April 1990.
- [21] G. D'Aria, V. Zingarelli, "Results on Fast-Kalman and Viterbi Adaptive Equalizers for Mobile Radio with CEPT/GSM System Characteristics," *Proceedings of IEEE GLOBECOM*, pp.26.3.1-26.3.5, 1988.
- [22] Electronics Industries Association, "EIA/TIA INTERIM STANDARD: Cellular System Dual-Mode Mobile Station-Base Station Compatibility IS-54" Electronic Industries Association, May 1990.
- [23] V. Fung, "Bit Error Simulation of FSK, BPSK, and $\pi/4$ DQPSK in Flat and Frequency-Selective Fading Mobile Radio Channels Using Two-Ray and Measurement-Based Impulse Response Models," *Thesis for Masters of*

Science in Electrical Engineering at VPI&SU, 1991.

- [24] L.W. Johnson R.D. Riess, *Numerical Analysis*, Addison-Wesley, 1982.
- [25] G.J. Bierman, *Factorization Method for Discrete Sequential Estimation*, Academic, New York, 1977.
- [26] S. Thomas Alexander, *Adaptive Signal Processing*, Springer-Verlag, 1986.
- [27] I. Korn, *Digital Communications*, Van Nostrand Reinhold, 1985.
- [28] B. Widrow, et. al., "Stationary and Non-Stationary Learning Characteristics of the LMS Adaptive Filter" *Proceedings of the IEEE*, vol.64, No.8, pp.1151-1162, August 1976.
- [29] S.U.H. Qureshi, "Adaptive Equalization", *Proceedings of IEEE*. vol.73, No.9, pp.1349-1387, September 1985.
- [30] P. Monsen, "Adaptive Equalization of the Slow Fading Channel" *IEEE Transactions on Communications*, vol.22, pp.1064-1074, August, 1974.
- [31] C.L. Liu, K. Feher, "Performance of Non-Coherent $\pi/4$ -QPSK in a Frequency-Selective Fast Rayleigh Fading Channel," *IEEE International Conference on Communications*, pp.335.7.1-335.7.5, Atlanta, April 1990.
- [32] R.H. Clarke, "A Statistical Theory of Mobile Radio Reception," *Bell System Technical Journal*, vol.27, pp.975-1000, July-August 1968.
- [33] W.C. Jakes, *Microwave Mobile Communications*, Wiley, New York, 1974.
- [34] L.W. Couch II, *Digital and Analog Communication Systems*, Macmillan, New York, 1989.
- [35] S.Y. Seidel, "UHF Indoor Radio Channel Models for Manufacturing Environments," *Masters Thesis in Electrical Engineering* at VPI&SU, 1989
- [36] S.Y. Seidel et. al, "Path Loss, Scattering and Multipath Delay Statistics in Four European Cities for Digital Cellular and Microcellular Radiotelephone," *IEEE Transactions on Vehicular Technology*, vol.40, No.4, pp.721-730, November 1991.
- [37] J.M. Cioffi, "When Do I Use an RLS Adaptive Filter?" *Proceedings, 19th Asilomar Conference on Circuits, Systems, and Computers*, Pacific Grove, California, November 1985.
- [38] T.S. Rappaport, "Wireless Personal Communications: Trends and Challenges," *IEEE Antennas and Propagation Magazine*, vol.33, No.5, pp.19-29, October 1991.
- [39] W.C.Y. Lee, "Theory of Wideband Radio Propagation", *IEEE Vehicular*

Technologies Conference 1991, St. Louis, pp 285 - 288, 1991.

- [40] S.U.H. Qureshi, "Adaptive Equalization," *Proceedings of the IEEE* vol.73, No. 9, pp.1340-1387, September 1985.
- [41] T.S. Rappaport et. al., "Statistical Channel Impulse Response Models for Factory and Open Plan Building Radio Communication System Design," *IEEE Transactions on Communications*, vol.39, No.5, pp.794-807, May 1991.
- [42] T.S. Rappaport, V. Fung, "Simulation of Bit Error Performance of FSK, BPSK, and $\pi/4$ DQPSK in Flat Fading Indoor Radio Channels Using a Measurement-Based Channel Model," *IEEE Transactions on Vehicular Technology*. Vol.40, No.4, pp.731-740, November 1991.

Appendix A. Software

This appendix presents a review of the equalization simulation programs. A complete program includes a main program, a channel program, and an equalization program. The main program controls the simulation, chooses the parameters of the equalizer, selects the channel, and records the BER. The channel program generates the pseudo-random binary signal, and convolves the signal with the raised cosine filter and channel, and also adds white Gaussian noise. The equalization program simulates the equalizer and the algorithm.

In the simulation of equalization for two-ray Rayleigh fading channels, three different equalizer structures are used: DFE with the Kalman algorithm, L-DFE with an RLS algorithm, and MLSE.

The Rayleigh fading channels are generated independently by *BERSIM*. Two channel files, one for the first ray and one for the second ray, are needed. There are a real column and an imaginary column in each file. Each time a symbol is sent by the receiver, a new channel is used by reading a new channel impulse response from the channel data files. There are some channel files in the Gateway 386 generated for different speeds in *C:\HTEMP\FCHAN**.DAT*, *SCHAN**.DAT*, where '**' is a number that stands for the speed of the mobile.

TWORAY is the main program controlling the simulation of an RLS-DFE for a two-ray channel. As shown in Figure A-1, it calls several units. In *TWORAY*, one can change the size of the equalizer (**n1**: number of forward taps, **n2**: number of the feedback taps), noise power (**noise**), weighting factor (**lambda**), delay of the second ray (**delayray**, when **delayray**=1, there is only one ray, and when **delayray**=13, the delay of the second ray is one symbol duration), etc. It also controls the total bits sent, the two-ray channel files and result output file.

COMPTU defines a complex type and is consisted of complex calculation subroutines, such as *CADD* (complex addition), *CSUB* (complex subtraction), *CMUL* (complex multiplication), *CDIV* (complex division). These subroutines significantly simplified the other programs. Therefore, *COMPTU* is widely called by other programs.

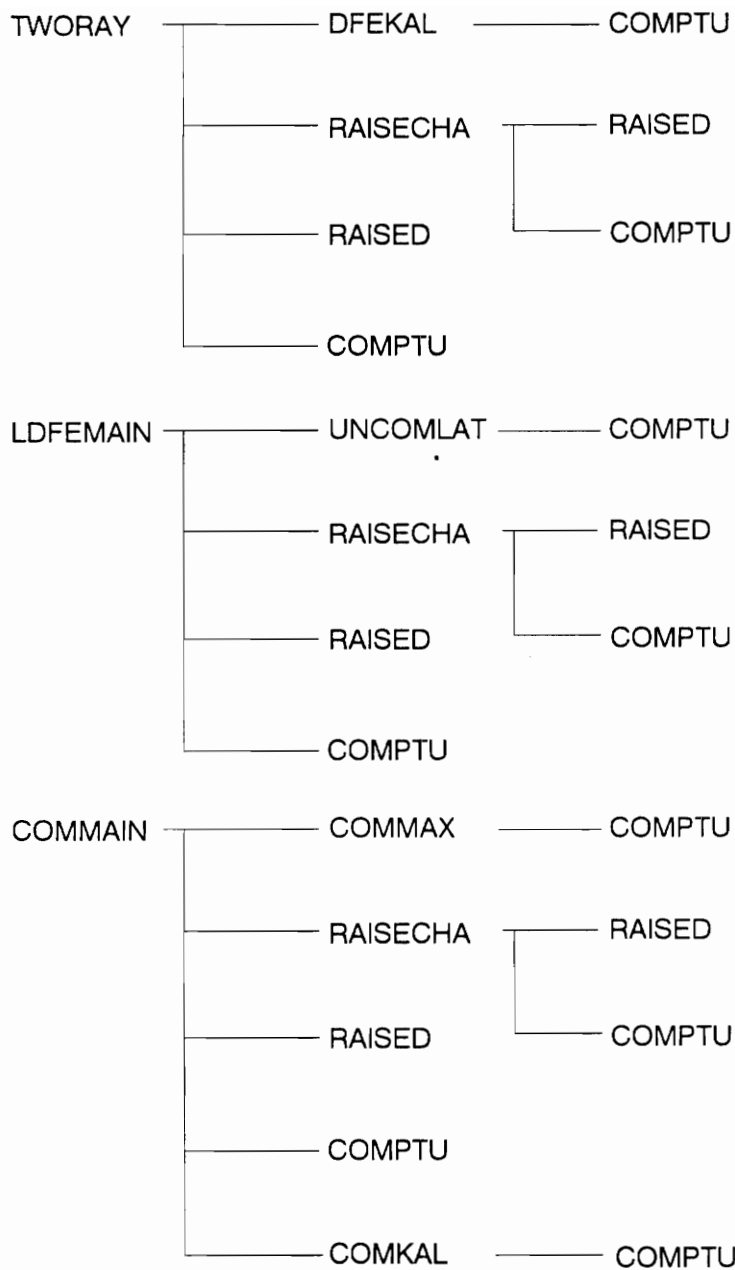


Figure A-1. Structures of program *TWORAY*: the Kalman DFE in two-ray Rayleigh fading channels, *LDFEMAIN*: RLS L-DFE in two-ray Rayleigh fading channels, and *COMMAIN*: the MLSE in two-ray Rayleigh fading channels.

DFEKAL is the unit simulating the DFE Kalman algorithm, which updates the equalizer coefficients. There is a limitation in the program: the total number of the taps should be smaller than 20. To increase this number, the sizes of arrays defined in *DFEKAL.PAS* should be increased. *INITIALKA* is the subroutine to initialize the equalizer and the algorithm. *MOVE* is used to shift the equalizer without applying the algorithm to change the tap-gains. It is used at the first few symbols of a block. *COKAL* is the subroutine executing one iteration of the Kalman algorithm.

RAISECHA is the unit simulates the transmitter and the fading channel. *MCINI* initializes the channel, and *MOBILECHANNEL* is the subroutine convolves the input signal with the raised-cosine filter and channel and adds noise at the output.

RAISED is the unit generating the impulse response of a raised-cosine filter. The rolloff factor is defined as 0.35, but it can be changed afterword.

LDfEMAIN (see Figure A-1) is similar to *TWORAY* except that it uses the L-DFE structure. Therefore, it calls the unit *UNCOMLAT* instead of *DFEKAL*. *INITIALAT* initializes the L-DFE, and *COMPLEXISALE* executes one iteration of the L-DFE algorithm.

COMMAIN in Figure A-1 simulates the MLSE algorithm. *COMMAX* is the unit consisting of the Viterbi decoder, which chooses the mostly likely data sequence as the output. *INITIALVI* is the subroutine initializing the Viterbi decoder, and *VITERBI* estimates one block of data. *COMMAIN* calls the same two-ray channel subroutines. *COMKAL* is the Kalman algorithm updating the CIRE.

The previous programs simulates the equalization in two-ray channels. The real world channel models, SIRCIM and SMRCIM are implemented. First, the SIRCIM and SMRCIM channel data files have to be adopted into data files that can be used for the simulation (see Section 8.2). A *.BN2* file can be generated by running SIRCIM or SMRCIM. Then, *BN2CUBIC* is executed. You will be asked information about SIRCIM or SMRCIM, data rate, and file names, etc. After that, subroutine *TRANSFER* in unit *DIRSMR* is called to read the channel files and adjust the resolution of excess delay. Then, *DOSPLINE* in unit

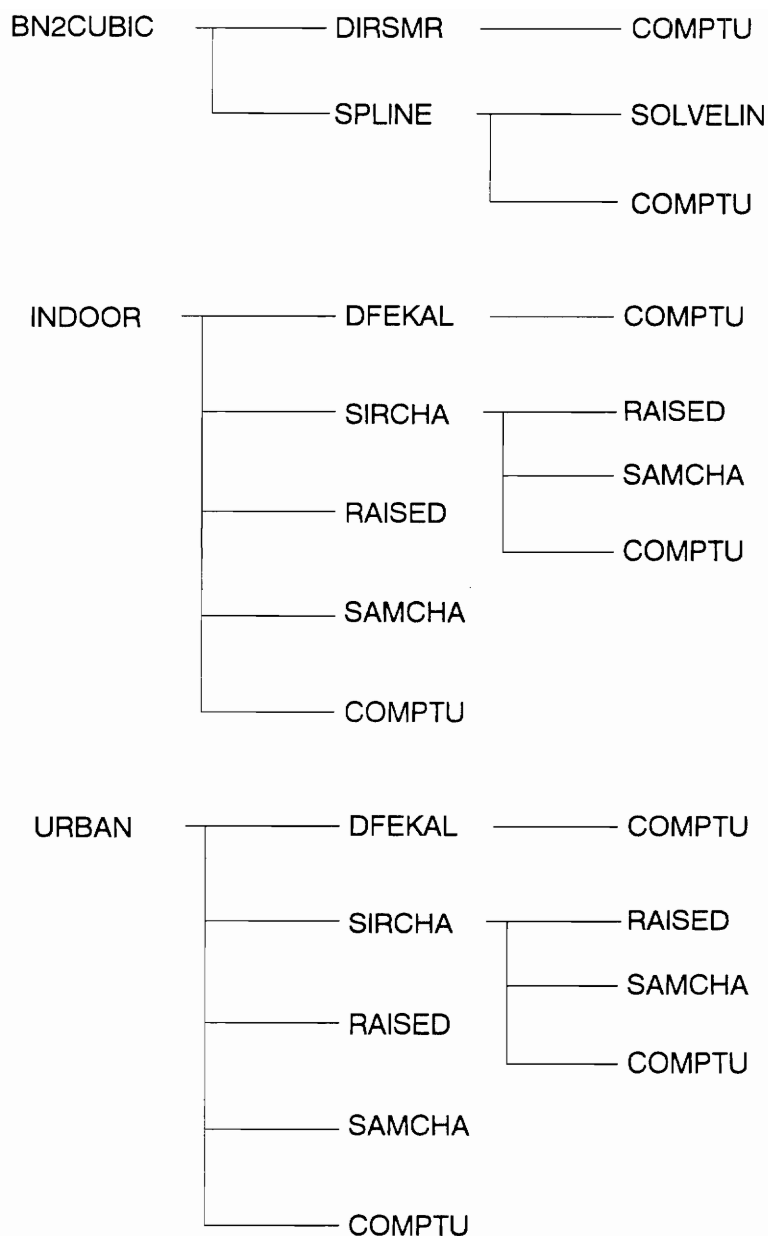


Figure A-2. Structures of program *BN2CUBIC*: interpolating SIRCIM and SMRCIM channels, *INDOOR*: the Kalman DFE in SIRCIM indoor channels, *URBAN*: the Kalman DFE in SMRCIM urban channels.

SPLINE is called to do the cubic spline interpolation. *SOLVELIN* is the unit to solve a set of linear equations using Gauss elimination, which is needed in cubic spline (see Section 8.2).

INDOOR and *URBAN* shown in Figure A-2 are the two programs using SIRCIM and SMRCIM channels. They are very similar to *TWORAY*, except they use different channel models. Hence, the unit simulating the channel, *SIRCHA*, is a little different from *RAISECHA* in that *SIRCHA* uses more than two rays in the channel. *SIRCHA* also calls *INPUTCHANNEL* in unit *SAMCHA*, which is used to obtain the impulse response of the channel. With the channel impulse response, *SIRCHA* convolves the channel with the data and the raised cosine filters to obtain the output of the channel. *DFEKAL* is the same unit used for the two-ray model in *TWORAY*.

DIFKAL is the main program controlling the simulation of the equalization in $\pi/4$ DQPSK modulation (see Figure A-3). *DIFKAL* uses the two-ray Rayleigh fading channel models as in *TWORAY*. Thus, all the units *DIFKAL* called are similar to *TWORAY*. *DRAISECHA* is similar to *RAISECHA*, and it is modified to fit $\pi/4$ DQPSK modulation, and so is *KALDQPSK*, the unit simulating the Kalman algorithm.

DINDOOR in Figure A-3 is the main program controlling the equalization in $\pi/4$ DQPSK for SIRCIM indoor channel models. It is very similar to *INDOOR* but it modified the Kalman algorithm (*KALDQPSK*) and channel program (*DSIRCHA*) to fit the differential modulation.

There are also some other simulation programs in *E:\MPRRG\WF-H\EQUALIZE* of UNIVERCITY SYSTEM. They were used to simulate the converging and tracking of the equalizers as shown in Chapter 6. These programs are self-supporting, so they will run without any other files.

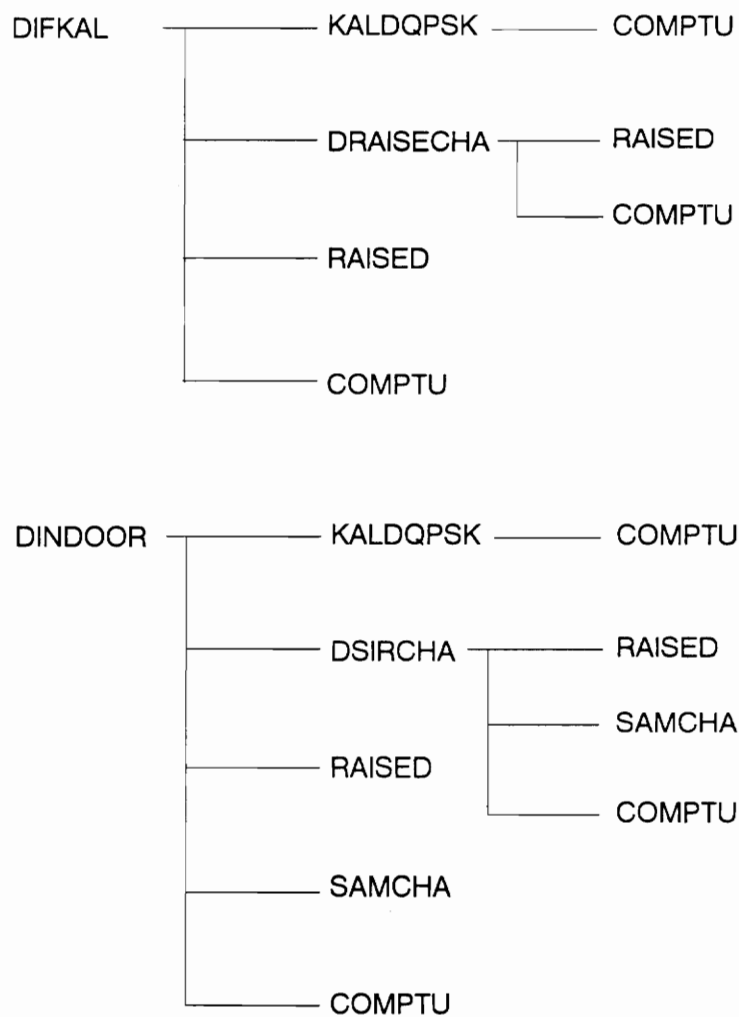


Figure A-3. Structures of program *DIFKAL*: the Kalman DFE for $\pi/4$ DQPSK in two-ray Rayleigh fading channels, *DINDOOR*: the Kalman DFE for $\pi/4$ DQPSK in SIRCIM indoor channels

Appendix B. Computer Programs

```

{*****}
FILE NAME: TWORAY.PAS
CONTAIN UNITS: CRT,COMPTU,DFEKAL,RAISECHA,RAISED
CONTAIN:
    procedure inputdata;
    procedure checkcb;
    procedure setsize;
    procedure setnoise;
    procedure main;
PURPOSE:
    This program uses DFE-Kalman to evaluate its performance
    on a two ray Rayleigh fading model.
    The delay of the second ray, the power ratio of the second
    ray are changeable. Raised-cosine filter is used.
{*****}
program usekal;
($N+)
    uses crt,comptu,dfekal,raisecha,raised;
    var i1,m1,id,totalerror,totalnum:longint;
        db,rol:integer;
        noise:real;
        filedat:text;
        filename:string[30];
        training:boolean;
        pwra,noi:array[1..10] of real;
        result:array [1..30] of real;

{*****}
PROCEDURE: INPUTDATA;
DESCRIPTION: Get an output from the mobile channel.
{*****}
    procedure inputdata;
    begin
        mobilechannel(noise);
    end;

{*****}
PROCEDURE: checkcb;
DESCRIPTION: Compared the decision and original symbol
            to see if there are errors.
INPUT PARMAS: a, b: data symbol and decision symbol
              n: the total number of error before this comparison
OUTPUT PARMAS: n: the total number of error after this comparison
{*****}
    procedure checkcb(var n:longint; a,b:complex);
    begin
        if round(a.re)<>round(b.re) then n:=n+1;
        if round(a.im)<>round(b.im) then n:=n+1;
    end;

{*****}
PROCEDURE: setsize;
DESCRIPTION: Initializes the size, weighting factor, delay of the
            equalizer.
OUTPUT PARMAS: n1: forward tap #
              n2: feedback tap #
              delay: delay of the equalizer;
              tN: total tap #
              lambda:weighting factor;
              dt: delay caused by the raised-cosine filter;
              result[]: simulation results to be monitored
{*****}
    procedure setsize;
    begin
        n1:=4;delay:=n1-1;
        n2:=2;
        tN:=n1+n2;

```

```

        lambda:=0.9;
        delta:=1e6;
        dt:=3;
        for i:=1 to 30 do
            result[i]:=100;
        end;
    end;
    {*****}
    PROCEDURE: setnoise;
    DESCRIPTION: setting noise power;
    OUTPUT PARMAS: noi[]: different noise power chosen;
    {*****}
    procedure setnoise;
    begin
        noi[1]:=0.158;
        noi[2]:=0.1;
        noi[3]:=0.05;
        noi[4]:=0.025;
        noi[5]:=0.0158;
        noi[6]:=0.01;
        noi[7]:=0.005;
    end;
    {*****}
    PROCEDURE: main;
    DESCRIPTION: Simulate 10000 bits using the chosen paramaters.
                Show the temperal results on the screen.
    INPUT PARMAS: xin; Symbol sent;
                d; symbol detected;
                xu; Symbol sent (synchronized to d);
                modulation; QPSK or DQPSK;
                file1,file2: Data files of the Rayleigh fading channel;
    OUTPUT PARMAS:totalerror: total # of error;
                totalnum: total # of bits sent
                result[]: recorded results
    CALLS:
                initialKa
                inputdata
                CoKal
                mcini;
    {*****}
    procedure main;
    var nnn,numoferr:longint;
        mse:real;
        ii,trainnum:integer;
    begin
        reset(file1);
        reset(file2);
        xu:=c0;
        d:=c0;
        xin:=c0;
        nnn:=0;
        totalnum:=0;
        totalerror:=0;
        mcini;
        initialKa;
        for ii:=1 to 6 do
            inputdata;
            mse:=0;
            if modulation='QPSK' then trainnum:=10
            else trainnum:=20;
            for ii:=1 to 1 do
                begin
                    mse:=0;
                    for m1:=1 to trainnum do
                        begin
                            inputdata;
                            training:=true;

```

```

        CoKal(training,inp[1],xin,modulation);
        mse=mse+alpha.re*alpha.re+alpha.im*alpha.im;
    end; writeln(mse/trainnum);
end;
repeat
    mse:=0;
    for ii:=1 to 100 do
        begin
            nnn=nnn+1;
            mse:=0;
            numoferr:=0;
            for m1:=1 to 10 do
                begin
                    inputdata;
                    training:=true;
                    CoKal(training,inp[1],xin,modulation);
                    checkcb(numoferr,xu,d);
                    mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
                end;
                totalnum:=totalnum+20;
                totalerror:=totalerror+numoferr;
                mse:=mse/10;
                writeln(delayray,' ',nnn*10-5,mse/2,' # of error: ',numoferr,'/20',totalerror
            end;
        until totalnum>8000;
    end;

(*****
Main Program to change the paramaters
variables: pwrratio: power ratio of two rays
           delayray: delay of the second ray (13 = T, 1 = 0)
           rolloff: rolloff factor
           noise: noise power;
           totalnum:total number of bits
           toterror: total number of errors
           file1,file2: the files contain the Rayleigh fading
                       channel (one ray in each file).
                       They are generated by BERSIM
           filedat: BER output data file
*****}
begin
    coherent:=false;
    Modulation:='QPSK';
    raisedcosine;
    assign(file1,'c:\htemp\fchan75');
    reset(file1);
    assign(file2,'c:\htemp\schan75');
    reset(file2);
    assign(filedat,'c:\htemp\dfc4275.d2'); {Output data file}
    rewrite(filedat);
    setsize;
    setnoise;
    db:=0;{dB ratio of the two ray}
    pwrratioDB:=2*db;
    rolloff:=0.35;
    for delayray:=1 to 14 do {Change the delay of the second ray}
        begin
            noise:=0;{choose the noise power}
            main;
            writeln(filedat,delayray,' ',totalerror/totalnum);
            result[delayray]:=totalerror/totalnum;
        end;
    repeat
        delayray:=delayray+2;

```

```
main;
    writeln(filedat,delayray,' ',totalerror/totalnum);
    result[delayray]:=totalerror/totalnum;
until delayray>25;
close(filedat);
close(file1);
close(file2);
end.(end of main)
```

```

{*****}
FILENAME: DFEKAL;
CONTAIN: Move;
        CoKal
        initialKa
        CoLMS
DESCRIPTION: Executes an iteration of Complex DFE-Kalman algorithm
             for QPSK and Pi/4 DQPSK (as DQPSK);
*****}
unit dfekal;
interface
  uses comptu;
  type
    data1=array[1..20] of complex;
  var
    n1,n2,tN:integer;
    findecision:array [0..19] of complex;
    u,din,td:array [1..20] of complex;
    x,w,kk:data1;
    Pv:array [1..20] of data1;
    dif,prealpha,sum,alpha,xu,d:complex;
    z:array [0..20] of complex;
    weigh,lambda,delta,mse,coe:real;
    OvDelay,number,count,delay:integer;
    test:boolean;
  procedure Move(inp:complex;xin:complex;modulation:string);
  procedure CoKal(training:boolean;inp:complex;xin:complex;modulation:string);
  procedure initialKa;
  procedure CoLMS(training:boolean;inp:complex;xin:complex;modulation:string);
implementation
{*****}
PROCEDURE: Move;
DESCRIPTION: Shift the registers
INPUT PARMAS: inp: output of the channel;
              xin: data sent
OUTPUT PARMAS: u; input vector;
              din; decision vector;
*****}

procedure Move(inp:complex;xin:complex;modulation:string);
  var i:integer;
      lin:real;
  begin
    for i:=0 to n1-2 do
      u[n1-i]:=u[n1-1-i];
    u[1]:=inp;
    for i:=0 to n2-2 do
      u[tN-i]:=u[tN-1-i];
    u[n1+1]:=din[delay+1];
    if modulation='QPSK' then
      begin
        for i:=0 to 18 do
          din[20-i]:=din[19-i];
        din[1]:=xin
      end
    else
      begin
        for i:=0 to 18 do
          td[20-i]:=td[19-i];
        td[1]:=xin;
        for i:=0 to 18 do
          din[20-i]:=din[19-i];
        cmul(din[1],xin,din[2]);
        lin:=sqrt(2)/2;

```

```

        rmul(din[1],din[1],lin);
    end;
end;
{*****}
PROCEDURE: CoKal;
DESCRIPTION: Executes an iteration of DFE-Kalman;
INPUT PARMAS: training: in the training or not
               inp: input of the equalizer
               xin: data sent
               modulation: choice of modulation (QPSK or DQPSK)
OUTPUT PARMAS: d; decision
               alpha; prediction error;
{*****}
procedure CoKal(training:boolean;inp:complex;xin:complex;modulation:string);
var i,ik:integer;
    nxu,ct,calp,tu:complex;
    cv,cu:data1;
    amt,lin:real;
begin

    for i:=0 to n1-2 do
        u[n1-i]:=u[n1-1-i];
    u[1]:=inp;
    for i:=0 to n2-2 do
        u[tN-i]:=u[tN-1-i];
    u[n1+1]:=din[delay+1];

    if modulation='QPSK' then
        begin
            for i:=0 to 18 do
                din[20-i]:=din[19-i];
            din[1]:=xin
        end
    else
        begin
            for i:=0 to 18 do
                td[20-i]:=td[19-i];
            td[1]:=xin;
            for i:=0 to 18 do
                din[20-i]:=din[19-i];
            cmul(din[1],xin,din[2]);
            lin:=sqrt(2)/2;
            rmul(din[1],din[1],lin);
        end;

    (
        if training=true then
            u[1]:=d[0vDelay]
        else u[1]:=findecision[L-step];
        for i:=0 to 18 do
            din[20-i]:=din[19-i];
        din[1]:=z[0vDelay];
    )

    alpha:=c0;
    for i:=1 to tN do
        begin
            conjugate(ct,w[i]);
            cmul(ct,ct,u[i]);
            cadd(alpha,alpha,ct);
        end;
        if Modulation='QPSK' then
            begin
                if alpha.re>0 then d.re:=1 else d.re:=-1;

```

```

    if alpha.im>0 then d.im:=1 else d.im:=-1;
    if training=true then xu:=din[delay+1]
      else xu:=d;
    csub(alpha,xu,alpha);
  end
else begin
  lin:=sqrt(2)/2;
  cdiv(dif,alpha,prealpha);
  if dif.re>0 then d.re:=1 else d.re:=-1;
  if dif.im>0 then d.im:=1 else d.im:=-1;
  if training=true then xu:=td[delay+1]
    else xu:=d;
  rmul(nxu,xu,lin);
  amt:=1/sqrt(prealpha.re*prealpha.re+prealpha.im*prealpha.im);
  rmul(prealpha,prealpha,amt);
  cmul(tu,nxu,prealpha);
  prealpha:=alpha;
  csub(alpha,tu,alpha);
end;

for i:=1 to tN do
begin
  x[i]:=c0;
  for ik:=1 to tN do
    begin
      cmul(ct,Pv[i,ik],u[ik]);
      cadd(x[i],x[i],ct);
    end;
  end;
  sum.re:=lambda;
  sum.im:=0;
  for i:=1 to tN do
    begin
      conjugate(ct,u[i]);
      cmul(ct,ct,x[i]);
      cadd(sum,sum,ct);
    end;
  end;
  for i:=1 to tN do
    begin
      cdiv(kk[i],x[i],sum);
    end;
  end;

  conjugate(calp,alpha);
  for i:=1 to tN do
    begin
      cmul(ct,kk[i],calp);
      cadd(w[i],w[i],ct);
    end;
  end;

  for i:=1 to tN do
    conjugate(cu[i],u[i]);
  for i:=1 to tN do
    begin
      cv[i]:=c0;
      for ik:=1 to tN do
        begin
          cmul(ct,cu[ik],Pv[ik,i]);
          cadd(cv[i],cv[i],ct);
        end;
      end;
    end;
  end;
  for i:=1 to tN do
    for ik:=1 to tN do
      begin
        cmul(ct,kk[i],cv[ik]);
      end;
    end;
  end;

```

```

        csub(Pv[i][ik],Pv[i][ik],ct);
        rmul(Pv[i][ik],Pv[i][ik],1/lambda);
    end;
end;
{*****}
PROCEDURE: CoLMS; (It does not work yet)
DESCRIPTION: Executes an iteration of DFE-LMS
INPUT PARAMS: training: in the training or not
               inp: input of the equalizer
               xin: data sent
               modulation: choice of modulation (QPSK or DQPSK)
OUTPUT PARAMS: d; decision
               alpha; prediction error;
               w[]: tap gain vector;
{*****}
procedure CoLMS(training:boolean;inp:complex;xin:complex;modulation:string);
var ct,q,error:complex;
    tem:real;
    i:integer;
begin
    tem:=0;
    for i:=1 to tN do
        tem:=tem+u[i].re*u[i].re+u[i].im*u[i].im;
    weigh:=coe/tem;
    move( inp,xin,modulation);
    alpha:=c0;
    for i:=1 to tN do
        begin
            conjugate(ct,w[i]);
            cmul(ct,ct,u[i]);
            cadd(alpha,alpha,ct);
        end;
        csub(alpha,din[i],alpha);

        conjugate(cerror,alpha);
        for i:=1 to tN do
            begin
                rmul(q,u[i],weigh);
                cmul(q,ceerror,q);
                csub(w[i],w[i],q);
            end;
        end;
end;

{*****}
PROCEDURE: initialKa;
DESCRIPTION: Initialize the equalizer;
{*****}
procedure initialKa;
var il,ill :integer;
begin
    c0.re:=0;
    c0.im:=0;
    delta:=100000;
    for il:=1 to 20 do
        begin
            for ill:=1 to 20 do
                Pv[il,ill]:=c0;
                Pv[il,il].re:=delta;
                w[il]:=c0;
                w[il].re:=0.3;
            end;
            for il:=1 to 20 do
                begin
                    din[il]:=c0;
                    z[il]:=c0;
                end;
            end;
        end;
    end;
end;

```

```
        u[i1]:=c0;  
    end;  
    z[0]:=c0;  
    prealpha.re:=1;  
    prealpha.im:=0;  
end;  
end.
```

```

{*****}
FILENAME: raisecha
        RAISED
        COMPTU
CONTAIN: mobilechannel
        mcini;
        putraised
DESCRIPTION: Simulates mobile channels
{*****}
Unit raisecha;
interface
    uses comptu,raised;
    const Ctodb=0.230258509;
    var anp,inp:array[1..100] of complex;
        xin,apre:complex;
        dase:array [1..10] of complex;
        cr,xl:array [1..100] of complex;
        c:array [1..30] of complex;
        amp,pwr:array[1..2] of real;
        nnn,delayray,dt,cl:integer;
        Ac,pwrratiordB,tnoi:real;
        Modulation:string[10];
        file1,file2:text;
        coherent:boolean;
    procedure mobilechannel(noise:real);
    procedure mcini;
implementation
    var rt:real;
{*****}
PROCEDURE: putraised
DESCRIPTION: convolve channel with the raised cosine filter
INPUT PARMAS: c[]: channel impulse response;
OUTPUT PARMAS: cf[]: after convolving with raised-cosine filter
{*****}
    procedure putraised;
        var i,n,k:integer;
            q:real;
            ctemp:complex;
        begin
            for i:=1 to 69 do
                begin
                    cr[i]:=c0;
                    n:=0;
                    repeat
                        n:=n+1;
                        rmul(ctemp,c[n],raisedfilter[i+1-n]);
                        cadd(cr[i],cr[i],ctemp);
                    (
                        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; )
                    until (n=cl) or (i+1-n=1);
                end;
            k:=0;
            for i:=70 to 68+cl do
                begin
                    cr[i]:=c0;
                    k:=k+1;n:=k;
                    repeat
                        n:=n+1;
                        rmul(ctemp,c[n],raisedfilter[i+1-n]);
                        cadd(cr[i],cr[i],ctemp);
                    (
                        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; )
                    until n=cl;
                end;
            end;
        end;
end;

```

```

{*****}
PROCEDURE: mobilechannel
DESCRIPTION: Convolving channel with data to get an output of the channel
INPUT PARMAS: noise : noise power
OUTPUT PARMAS: xin: data sent
               inp[1]: output of the channel
{*****}
  procedure mobilechannel(noise:real);
    const p=2;
    var i,n:integer;
        rt,q:real;
        a,temp,nmid:complex;
{*****}
FUNCTION: gauss:
DESCRIPTION: Generates a random Gaussian number 0 mean 1 variance
{*****}
  function gauss:real;
    var temp:real;
        i:integer;
    begin
      temp:=0;
      for i:=1 to 12 do
        temp:=temp+random;
      gauss:=temp-6;
    end;
{*****}
PROCEDURE powerratio;
DESCRIPTION: Setup different power ratio of the two ray
INPUT PARMAS: Ctodb
               pwrriatioDB: power ratio
               c[];
OUTPUT PARMAS: c[]:
{*****}
  procedure powerratio;
    var i:integer;
    begin
      rt:=exp(pwrriatioDB*Ctodb);
      pwr[2]:=1/(1+rt);
      pwr[1]:=rt*pwr[2];
      for i:=1 to cl do
        begin
          amp[i]:=sqrt(pwr[i]*0.5)*2;
          rmul(c[i],c[i],amp[i]);
        end;
    end;

    begin
      cl:=delayray;
      readln(file1,c[1].re,c[1].im);
      readln(file2,c[delayray].re,c[delayray].im);

      putraised;
      for i:=0 to 8 do
        dase[10-i]:=dase[9-i];

        if random>0.5 then dase[1].re:=1
        else dase[1].re:=-1;
        if random>0.5 then dase[1].im:=1
        else dase[1].im:=-1;

        if Modulation='QPSK' then
          a:=dase[1]
        else begin

```

```

        cmul(a,dase[1],apre);
        q:=sqrt(2)/2;
        rmul(a,a,q);
        apre:=a;
    end;
    for i:=12 downto 1 do
    begin
        for n:=0 to 98 do
            xl[100-n]:=xl[99-n];
            if i=12 then
                xl[1]:=a
            else xl[1]:=c0;
            anp[i].re:=0;
            anp[i].im:=0;
            for n:=1 to 68+cl do
            begin
                cmul(temp,cr[n],xl[n]);
                cadd(anp[i],anp[i],temp);
            end;
        end;
        xin:=dase[dt];
        apre:=a;
        anp[1]:=anp[2];
        inp[1]:=anp[1];
        tnoi:=sqrt(noise);
        temp.re:=gauss*tnoi;
        temp.im:=gauss*tnoi;
        cadd(inp[1],inp[1],temp);
    end;{of complex channel}

```

```

{*****
PROCEDURE: mcini;
DESCRIPTION: Initialize the channel
*****}
    procedure mcini;
    begin
        for nnn:=1 to 100 do
        begin
            anp[nnn].re:=1;
            anp[nnn].im:=1;
            inp[nnn].re:=1;
            inp[nnn].im:=1;
            xl[nnn].re:=0;
            xl[nnn].im:=0;
            cr[nnn].re:=0;
            cr[nnn].im:=0;
        end;
        for nnn:=1 to 30 do
        begin
            c[nnn].re:=0;
            c[nnn].im:=0;
        end;
        apre.re:=sqrt(2);
        apre.im:=0;
    end;
end.

```

```

{*****}
FILENAME: RAISED.PAS
CONTAIN: raisedcosine
PURPOSE: Compute the impulse response of the raised cosine filter
{*****}
unit raised;
interface
    var raisedfilter:array [1..69] of real;
        rolloff:real;
        i:integer;
    procedure raisedcosine;
implementation
    const T=12;
{*****}
FUNCTION: comraised
DESCRIPTION: compute the value of RC filter at a specific point
INPUT PARMAS: n: time
OUTPTU PARMAS: comraised: the value;
{*****}
    function comraised(n:integer):real;
    var q,temp:real;
    begin
        if n=0 then
            comraised:=1
        else begin
            q:=pi*n/T;
            temp:=1-sqr(2*n/T*rolloff);
            comraised:=sin(q)*cos(q*rolloff)/temp/q;
        end;
    end;
{*****}
PROCEDURE : raisedcosine;
DESCRIPTION: To obtain all the value of the RC filter
OUTPUT PARMAS: raisedfilter[]: the impulse response of the RC filter
{*****}
    procedure raisedcosine;
    var i:integer;
    begin
        for i:=0 to 34 do
            begin
                raisedfilter[i+35]:=comraised(i);
                raisedfilter[35-i]:=raisedfilter[i+35];
            end;
        end;
{*****}
Initializaiton
{*****}
    begin
        for i:=1 to 69 do
            raisedfilter[i]:=0;
        rolloff:=0.35;(roll off factor from 0 to 1)
    end.

```

```

(*****
FILENAME: COMPTU.PAS
DESCRIPTION: This unit composes some complex computation subroutines
CONTAIN:
    cadd
    csub
    cmul
    conjugate
    cdiv
    rmul
*****}
unit comptu;
{$N+}
interface
    type complex= record
        re:real;
        im:real;
    end; {Define complex type}
    var c0:complex; {Complex 0}
    procedure cadd(var z:complex; x,y:complex);
    procedure csub(var z:complex; x,y:complex);
    procedure cmul(var z:complex; x,y:complex);
    procedure conjugate(var z:complex; x:complex);
    procedure cdiv(var z:complex; x,y:complex);
    procedure rmul(var z:complex; x:complex; y:real);
implementation
(*****
PROCEDURE cadd;
DESCRIPTION: Complex addition  $z=x+y$ ;
INPUT PARMAS: x,y: two complex number to be added
OUTPUT PARMAS: z: result
*****}
    procedure cadd(var z:complex; x,y:complex);
    begin
        z.re:=x.re+y.re;
        z.im:=x.im+y.im;
    end;
(*****
PROCEDURE csub;
DESCRIPTION: Complex addition  $z=x-y$ ;
INPUT PARMAS: x,y: two complex number
OUTPUT PARMAS: z: result
*****}

    procedure csub(var z:complex; x,y:complex);
    begin
        z.re:=x.re-y.re;
        z.im:=x.im-y.im;
    end;

(*****
PROCEDURE cmul;
DESCRIPTION: Complex addition  $z=x * y$ ;
INPUT PARMAS: x,y: two complex number
OUTPUT PARMAS: z: result
*****}

    procedure cmul(var z:complex; x,y:complex);
    begin
        z.re:=x.re*y.re-x.im*y.im;
        z.im:=x.re*y.im+x.im*y.re;
    end;
(*****

```

```

PROCEDURE conjugate;
DESCRIPTION: computes complex conjugate of a complex number;
INPUT PARMAS: x: a complex number
OUTPUT PARMAS: z: result
*****}

procedure conjugate(var z:complex; x:complex);
begin
  z.re:=x.re;
  z.im:=-x.im;
end;

{*****}
PROCEDURE cdiv;
DESCRIPTION: computes complex division; z=x/y
INPUT PARMAS: x,y: a complex number
OUTPUT PARMAS: z: result
*****}

procedure cdiv(var z:complex; x,y:complex);
var a:complex;
    b:real;
begin
  conjugate(a,y);
  cmul(a,a,x);
  b:=y.re*y.re+y.im*y.im;
  z.re:=a.re/b;
  z.im:=a.im/b;
end;

{*****}
PROCEDURE rmul;
DESCRIPTION: computes a real number multiplying a complex number;
INPUT PARMAS: x,y:
OUTPUT PARMAS: z: result
*****}

procedure rmul(var z:complex; x:complex; y:real);
begin
  z.re:=x.re*y;
  z.im:=x.im*y;
end;

{*****}
PROCEDURE amplitude;
DESCRIPTION: computes the amplitude of a complex number;
INPUT PARMAS: x:
OUTPUT PARMAS: z: result
*****}
procedure amplitude(var z:real; x:complex);
begin
  z:=sqrt(x.re*x.re+x.im*x.im);
end;

{Define c0 as a complex zero}
begin
  c0.im:=0;
  c0.re:=0;
end.

```

```

{*****}
FILENAME: LDFEMAIN.PAS
CONTAIN: Inputdata;
        checkcb;
        start;
        main;
        initial;
PURPOSE: Simulating Lattice DFE.
{*****}
program LDFEMAIN;
{$N+}
uses crt,{usecrt,} uncomlat,comptu,raised,raisecha;
var mse,noise:real;
    kn,numoferr,totalnum,totalerror,i,count:integer;
    nnn:longint;
    filedat:text;
    result:array[0..50] of real;
    training:boolean;
{*****}
PROCEDURE: inputdata;
DESCRIPTION: call mobilechannel;
{*****}
    procedure inputdata;
    var nu1,nu2:integer;
        c:complex;
    begin
        mobilechannel(noise);
    end;
{*****}
PROCEDURE: checkcb;
DESCRIPTION: check if the output of the equalizer and the data sent
            are the same;
INPUT PARMAS: a,b: symbol sent and received
              n: the previous accumulated error number
OUTPUT PARMAS: n: the recent accumulated error number
{*****}
    procedure checkcb(var n:integer; a,b:complex);
    begin
        if round(a.re)<>round(b.re) then n:=n+1;
        if round(a.im)<>round(b.im) then n:=n+1;
    end;
{*****}
PROCEDURE: start;
DESCRIPTION: training
CALL:      inputdata;
          complexisale;
{*****}
    procedure start;
    begin
        for i:=1 to 10 do
        begin
            inputdata;
            training:=true;
            complexisale(training,inp[1],xin,modulation);
        end;
    end;
{*****}
PROCEDURE: main;
DESCRIPTION: using 10000 bits to evaluate the BER of one set of
            parmaters
CALL:      mcini
          initailat

```

```

        inputdata
        complexisale
        *****)
procedure main;
begin
    mcini;
    initialat;
    nnn:=0;
    reset(file1);
    reset(file2);
    xu:=c0;
    d:=c0;
    totalnum:=0;{total number of bits}
    totalerror:=0;{total number of errors}
    xin:=c0;
    pwrriodB:=0; {power ratio of the two rays}
    for i:=1 to 10 do
        inputdata;
    start;
    repeat
        mse:=0;
        numoferr:= 0;
        for kn:=1 to 5 do
            begin
                nnn=nnn+1;
                numoferr:=0; {number of the error within the block};
                for count:=1 to 10 do
                    begin
                        inputdata;
                        training:=true;
                        complexisale(training,inp[1],xin,modulation);
                        checkcb(numoferr,xu,d);
                        mse:=mse+e[n1].re*e[n1].re+e[n1].im*e[n1].im;
                    end;
                totalnum:=totalnum+20;
                totalerror:=totalerror+numoferr;
                mse:=mse/20;
                write(delayray:4,' ',10*nnn-5:5,' ', mse/2:10:6,' ',numoferr,'/20
                writeln(totalerror/totalnum:10:7);
            end;
        until (totalnum>10000);
        writeln(filedat,delayray,' ',totalerror/totalnum,' ',nnn);
        result[delayray]:=totalerror/totalnum;{temporal results}
    end;

{*****
PROCEDURE: initial
DESCRIPTION: Setting the initial value;
*****}

Procedure initial;
begin
    n1:=4;{forward stages}
    n2:=2;{feedback stages}
    m:=n1-n2;
    dt:=3;
    delay:=2;{delay of the equalizer}
    delta:=1e-8;
    for i:=0 to 50 do
        result[i]:=100;
    clrscr;
    { overhead;}
    Modulation:='QPSK';
    coherent:=false;

```

```

lambda:=0.9;
raisedcosine;
noise:=0;
end;

{*****
Main program to control the simulation
*****}

begin
  initial;
  assign(filedat,'c:\htemp\raiself.dat'); {Output data file}
  rewrite(filedat);
  assign(file1,'c:\htemp\fchan75');
  assign(file2,'c:\htemp\schan75'); {Channel files}
  reset(file1);
  reset(file2);
  {lambda and delay are the two initial value, where lambda is the forgetting}
  {coefficient which is set from 0.9 to 1, and delay is the delay of the }
  {equalizer, which depends on the length of the equalizer and channel }
  {lambda is changable and delay is unchangable during the process }

  for delayray:=1 to 14 do {delayray is the delay of the second ray}
    { main;} {13 is one symbol duration}
  repeat
    delayray:=delayray+2;
  until delayray>25;

  close(file1);
  close(file2);
  close(filedat);
end.

```

```

(*****
FILENAME: UNCOMLAT.PAS
CONTAIN: complexisale
        initialat
DESCRIPTION: Executes an iteration of Lattice DFE
            The algorithm is from Ling and Proakis [12]
*****}

unit uncomlat;
{$N+}
interface
    uses comptu;
    type
        vector2 = array [1..2] of complex;
        mat22 = array[1..2,1..2] of complex;
        mmat = array [0..20] of complex;
        n1mat = array[0..20] of complex;
        n12mat = array [0..20] of vector2;
        n122mat = array [0..20] of mat22;
    var
        n1,n2,m,delay:integer;
        rfm1,rfm,rbm2,rbm1,rbm: array [0..20] of real;
        fm,bm1,fm1,bm,km,kxm :mmat;
        e,xtie,e1:n1mat;
        vb1,vb2,vf1,vb,vf,vkx: n12mat;
        vk,vrb2,vrb1,vrf1,vrf,vrb:n122mat;
        alpha,alpha1:array [0..20] of real;
        d,xu,btmm,d1,kbmm: complex;
        inxl:array [0..20] of complex;
        delta,lambda:real;
        procedure complexisale(training:boolean;inp:complex; xin:complex;modulation:string);
        procedure initialat;
    implementation
        var i,k,n:integer;

(*****
PROCEDURE: rdiv
Description: complex division.
            z=x/y;
INPUT PARMAS: x:
              y
OUTPUT PARMAS: z
*****}

    procedure rdiv(var z:complex; x:complex;y:real);
    begin
        z.re:=x.re/y;
        z.im:=x.im/y;
    end;

(*****
FUNCTION: cabs
DESCRIPTION: compute the power of the complex number x
INPUT PARMAS: x:
OUTPUT PARMAS: cabs;
*****}

    function cabs(x:complex):real;
    begin
        cabs:=x.re*x.re+x.im*x.im;
    end;

```

```

(*****
PROCEDURE matrixmult
DESCRIPTION: Matrix multiplication;
              resultmat=mat1*vct2
INPUT PARMAS: mat1,vct2:
OUTPUT PARMAS: resultvct:
*****)

procedure matrixmult (var resultvct:vector2; mat1: mat22; vct2: vector2);
var
  i,k :integer;
  sum,temp :complex;
begin
  for i:=1 to 2 do
    begin
      sum.re:=0;
      sum.im:=0;
      for k:=1 to 2 do
        begin
          cmul(temp,mat1[i,k],vct2[k]);
          cadd(sum,sum,temp);
        end;
        resultvct[i]:=sum;
      end;
    end;(end of matrixmult)

(*****
PROCEDURE: contranpose:
DESCRIPTION: H-transform(Transpose and conjugate)
INPUT PARMAS: mat1;
OUTPUT PARMAS: mat2;
*****)

procedure contranpose (var mat2:mat22; mat1: mat22);
var i,k: integer;
    temp:complex;
begin
  for i:= 1 to 2 do
    for k:= 1 to 2 do
      begin
        conjugate(temp,mat1[k,i]);
        mat2[i,k]:=temp;
      end;
    end; (end of transpose)

(*****
PROCEDURE: matitera
DESCPTION: matrix computation
INPUT PARMAS: alpha,lambda,y
OUTPUT PARMAS: z
*****)

procedure matitera(var z: mat22; y: vector2; alpha,lambda:real);
var
  temp:vector2;
  a:mat22;
  i,n:integer;
begin
  for i:=1 to 2 do
    conjugate(temp[i],y[i]);
  for i:=1 to 2 do
    for n:=1 to 2 do
      begin
        cmul(a[i,n],y[i],temp[n]);
        rmul(a[i,n],a[i,n],alpha);

```

```

        rmul(z[i,n],z[i,n],lambda);
        cadd(z[i,n],z[i,n],a[i,n]);
    end;
end;

{*****}
PROCEDURE: Cinverse
DESCRIPTION: inversing a matrix;
INPUT PARMAS: x: input matrix;
OUTPUT PARMAS: z: the inverse matrix of x;
{*****}

procedure cinverse(var z:mat22; x:mat22);
var i,n,k:integer;
    temp:mat22;
    q,a,b:complex;
begin
    cmul(a,x[1,1],x[2,2]);
    cmul(b,x[2,1],x[1,2]);
    csub(q,a,b);
    cdiv(z[1,1], x[2,2],q);
    cdiv(z[2,2], x[1,1],q);
    rmul(x[1,2],x[1,2],-1);
    rmul(x[2,1],x[2,1],-1);
    cdiv(z[1,2],x[1,2], q);
    cdiv(z[2,1],x[2,1], q);
end;

{*****}
PROCEDURE matitera2;
DESCRIPTION: matrix computation
INPUT PARMAS: z1,b,k,r
OUTPUT PARMAS: z;
{*****}

procedure matitera2(var z:vector2; z1,b:vector2; k,r:mat22);
var
    a,temp:mat22;
    v: vector2;
    i,n:integer;
begin
    cinverse(temp,r);
    matrixmult(v,temp,b);
    matrixmult(v,k,v);
    for i:=1 to 2 do
        csub(z[i],z1[i],v[i]);
    end;

{*****}
PROCEDURE: complexisale
DESCRIPTION: Executes an iteration of L-DFE algorithm
INPUT PARMAS: training : training status (true or false)
                inp: input of the equalizer;
                xin: the data sent
                lambda: weighting factor
OUTPUT PARMAS: d: detected symbol
                e: equalizer prediction error vector.
{*****}

procedure complexisale(training:boolean;inp:complex; xin:complex; modulation:string);
var q:real;
    temp,a,signal:complex;
    mt,inv:mat22;

```

```

        vt,vt2:vector2;
        i,k,n:integer;
begin
for i:=0 to 9 do
    inxl[10-i]:=inxl[9-i];
inxl[0]:=xin;
xu:=inxl[delay];

for i:=0 to m do
begin
    bm1[i]:=bm[i];
    fm1[i]:=fm[i];
    rfm1[i]:=rfm[i];
    rbm2[i]:=rbm1[i];
    rbm1[i]:=rbm[i];
end;
for i:=0 to n1 do
begin
    alpha1[i]:=alpha[i];
    e1[i]:=e[i];
end;
for i:= m to n1 do
    for n:=1 to 2 do
        begin
            vb1[i][n]:=vb[i][n];
            for k:=1 to 2 do
                begin
                    vrb2[i][n,k]:=vrb1[i][n,k];
                    vrb1[i][n,k]:=vrb[i][n,k];
                    vrf1[i][n,k]:=vrf[i][n,k];
                end;
            end;

bm[0]:=inp;
fm[0]:=inp;
rfm[0]:=lambda*rfm[0]+cabs(inp);
rbm[0]:=rfm[0];
( e[0]:=xu;)
xtie[0].re:=0;
xtie[0].im:=0;
alpha[0]:=1;

{Scalar stages}
for i:=1 to m do
begin
    conjugate(a,km[i]);
    cmul(temp,a,bm1[i-1]);
    rdiv(temp,temp,rbm2[i-1]);
    csub(fm[i],fm[i-1],temp);

    cmul(temp,km[i],fm[i-1]);
    rdiv(temp,temp,rfm1[i-1]);
    csub(bm[i],bm1[i-1],temp);

    conjugate(a,fm[i-1]);
    cmul(temp,a,bm1[i-1]);
    rmul(temp,temp,alpha1[i-1]);
    rmul(a,km[i],lambda);
    cadd(km[i],a,temp);
end;
for i:=1 to m-1 do
begin
    q:=cabs(km[i]);
    rfm[i]:=rfm[i-1]-q/rbm1[i-1];

```

```

    rbm[i]:=rbm1[i-1]-q/rfm[i-1];
end;
for i:=1 to m do
begin
    conjugate(temp,kxm[i]);
    cmul(temp,temp,bm[i-1]);
    rdiv(temp,temp,rbm1[i-1]);
    cadd(xtie[i],xtie[i-1],temp);
    alpha[i]:=alpha[i-1]-cabs(bm[i-1])*alpha[i-1]*alpha[i-1]/rbm[i-1];
end;
{Transition Stage}
cmul(temp,kbmm,fm[m-1]);
rdiv(temp,temp,rfm1[m-1]);
csub(btmm,e1[m-1],temp);
conjugate(temp,fm1[m-1]);
cmul(temp,temp,e1[m-1]);
rmul(temp,temp,alpha1[m-1]);
rmul(a,kbmm,lambda);
cadd(kbmm,a,temp);
vf[m][1]:=fm[m];
vf[m][2]:=e1[m];
vb[m][1]:=bm[m];
vb[m][2]:=btmm;
matitera(vrf[m],vf[m],alpha1[m],lambda);
matitera(vrb[m],vb[m],alpha[m],lambda);

{Two dimentional stages}
for i:=m+1 to n1-1 do
begin
    contranpose(mt,vk[i]);
    matitera2(vf[i],vf[i-1],vb1[i-1],mt,vrb2[i-1]);
    matitera2(vb[i],vb1[i-1],vf[i-1],vk[i],vrf1[i-1]);
    for n:=1 to 2 do
        conjugate(vt[n],vf[i-1][n]);
    for n:=1 to 2 do
        for k:=1 to 2 do
            begin
                cmul(mt[n,k],vb1[i-1][n],vt[k]);
                rmul(mt[n,k],mt[n,k],alpha1[i-1]);
                rmul(vk[i][n,k],vk[i][n,k],lambda);
                cadd(vk[i][n,k],vk[i][n,k],mt[n,k]);
            end;
        matitera(vrf[i],vf[i],alpha[i],lambda);
        cinverse(inv,vrb[i-1]);
        matrixmult(vt,inv,vb[i-1]);
        for n:=1 to 2 do
            begin
                conjugate(vt2[n],vb[i-1][n]);
                cmul(vt2[n],vt2[n],vt[n]);
            end;
        cadd(temp,vt2[1],vt2[2]);
        alpha[i]:=alpha[i-1]*(1-alpha[i-1]*temp.re);
        matitera(vrb[i],vb[i],alpha[i],lambda);
    end;

for i:=m+1 to n1 do
begin
    cinverse(mt,vrb1[i-1]);
    matrixmult(vt,mt,vb[i-1]);
    for n:=1 to 2 do
        begin
            conjugate(vt2[n],vkx[i][n]);
            cmul(vt2[n],vt2[n],vt[n]);
        end;
        cadd(temp,vt2[1],vt2[2]);

```

```

        cadd(xtie[i],xtie[i-1],temp);
    end;
    if xtie[n1].re>0 then d.re:=1 else d.re:=-1;
    if xtie[n1].im>0 then d.im:=1 else d.im:=-1;
    case training of
        true : signal:=xu;
        false: signal:=d;
    end;(case)
    e[0]:=signal;
    for i:=1 to m do
    begin
        csub(e[i],signal,xtie[i]);
        conjugate(temp,e[i-1]);
        cmul(temp,temp,bm[i-1]);
        rmul(temp,temp,alpha[i-1]);
        rmul(a,kxm[i],lambda);
        cadd(kxm[i],a,temp);
    end;
    for i:=m+1 to n1 do
    begin
        csub(e[i],signal,xtie[i]);
        conjugate(a,e[i-1]);
        rmul(a,a,alpha[i-1]);
        for n:=1 to 2 do
        begin
            cmul(vt[n],a,vb[i-1][n]);
            rmul(vkx[i][n],vkx[i][n],lambda);
            cadd(vkx[i][n],vkx[i][n],vt[n]);
        end;
    end;
end;
end;

{*****
PROCEDURE: initialat;
DESCRIPTION: Initialize the equalizer and the algorithm
*****}

procedure initialat;
begin
    for i:=0 to 20 do
        inxl[i]:=c0;
    rfm[0]:=0;
    rbm[0]:=0;
    kbmm:=c0;
    for i:=0 to m do
    begin
        bm[i]:=c0;
        fm[i]:=c0;
        km[i]:=c0;
        kxm[i]:=c0;
        rfm[i]:=delta;
        rfm1[i]:=delta;
        rbm[i]:=delta;
        rbm1[i]:=delta;
    end;
    for i:=0 to n1 do
    begin
        e[i]:=c0;
        alpha[i]:=1;
    end;
    for i:=m to n1 do
        for n:=1 to 2 do
            begin
                vkx[i][n]:=c0;

```

```

vb[i][n]:=c0;
vf[i][n]:=c0;
for k:=1 to 2 do
begin
  vk[i][n,k]:=c0;
  vrf[i][n,k]:=c0;
  vrb[i][n,k]:=c0;
  vrb1[i][n,k]:=c0;
end;
vrb[i][n,n].re:=delta;
vrb1[i][n,n].re:=delta;
vrf[i][n,n].re:=delta;
end;
end;
end.

```

```

{*****}
FILENAME: COMMAIN.PAS
CONTAIN UNITS: CRT, COMPTU, COMMAX, RAISECHA, COMKALM, RAISED.
CONTAIN:
    procedure inputdata,
        update,
        trainstage,
        checkcb,
        restart,
        main.
PURPOSE: the main program controls the MLSE in two-ray
        Rayleigh fading channels.
        The delay of the second ray, the noise power
        are changeable.
{*****}
program commain;
uses crt, comptu,commax,raisecha,
comkalm,raised;
var db,zl,ii,right,nu1,nu2,iz:integer;
ren,noise,meserr:real;
filedat:text;
filename:string[30];
errornum,totalerror,Modu:integer;
totalnum:longint;
start:boolean;
result:array[0..50] of real;
{*****}
PROCEDURE: inputdata
DESCRIPTION: Get an output from the mobile channel and set
        the noise power.
{*****}
    procedure inputdata;
    begin
        mobilechannel(noise);
    end;

{*****}
PROCEDURE: update
DESCRIPTION: Using RLS to update the channel impulse response estimator
variables: L+1: the length of the CIRE
OverallDelay: the delay of the MLSE
mse: mean square error
{*****}

    procedure update(training:boolean);
    var output,outdec:real;
        i,k:integer;
        OverallDelay:integer;
    begin
        OverallDelay:=L+1;
        mse:=0;
        for k:=0 to L do
            begin
                inputdata;
                ColMS(k,training);
                Cokal(k,training);
                ren:=din[1].re*din[1].re+din[1].im*din[1].im;
                mse:=mse+(alpha.re*alpha.re+alpha.im*alpha.im);
            end;
            mse:=mse/(1+L);
        end;{update}
{*****}
PROCEDURE: trainstage
DESCRIPTION: train the CIRE
{*****}

```

```

procedure trainstage;
var i,K:integer;
begin
  for k:=1 to 4 do
    for i:=0 to L do
      begin
        inputdata;
        CoKal(i,true);
      end;
    end;
  end;{trainstage}

{*****}
PROCEDURE: checkcb
DESCRIPTION: check if the detected symbol value is the same as the symbol
             value sent
VARIABLES : c:total number of bit errors
             a, b: the two symbols to be compared
{*****}

procedure checkcb(var c:integer;a,b:complex);
begin
  if round(a.re)<>round(b.re) then c:=c+1;
  if round(a.im)<>round(b.im) then c:=c+1;
end;

{*****}
PROCEDURE: restart
DESCRIPTION: restart the equalizer
{*****}

procedure restart;
var nnn:integer;
begin
  initialKa;
  initialvi;
  for nnn:=1 to 10 do
    begin
      inputdata;
      Move;
    end;
  trainstage;
  for zl:=1 to 4 do
    begin
      viterbi(start);
      update(true);
      start:=false;
    end;
  end;
end;

{*****}
PROCEDURE: main
DESCRIPTION: the main program to obtain BER of one point
VARIABLE : lambda:weighting factor
           noise : noise power
           totalnum:total number of bits
           totalerror: total number of bit errors
           finaldecson: the detected symbol value vector
           d: transmitted symbol value vector
{*****}

procedure main;
begin
  reset(file1);
  reset(file2);
  noise:=0.0;
  OvDelay:=L+L+2;

```

```

totalerror:=0;
totalnum:=0;
mcini;
initialKa;
initialvi;
for nnn:=1 to 50 do
begin
    inputdata;
    Move;
end;
nnn:=0;
lambda:=0.9;
trainstage;
for ii:=0 to L do
    findecision[ii]:=d[L+ii+2];
start:=true;
for zl:=1 to 4 do
begin
    viterbi(start);
    update(true);
    start:=false;
end;
repeat
    meserr:=0;
    for iz:=1 to 5 do
    begin
        right:=0;
        for zl:=1 to 5 do
        begin
            nnn:=nnn+1;
            viterbi(start);
            start:=false;
            for ii:=0 to L do
                checkcb(right,d[ii+L+2],findecision[ii]);
            nnn:=nnn;
            update(true);
            meserr:=meserr+mse/40;
        end;
        totalerror:=totalerror+right;
        totalnum:=totalnum+40;
        writeln (delayray,' ',nnn,meserr,' ',right,'/40',totalerror/totalnum:13:6);
    end ;
    restart; }
until (totalnum>8000);
writeln(filedat,delayray,' ',totalerror/totalnum);
result[delayray]:=totalerror/totalnum;
end;

{*****
DESCRIPTION: main program controls the simulation
VARIABLES: result: temporal BER results
            Modulation: Only choose QPSK
            file1, file2: the Rayleigh fading channel generated by
                        BERSIM
            filedat: output BER file
            delayray: delay of the second ray(13=T, 1=0)
*****}
begin
    for i:=0 to 50 do
        result[i]:=100;
    Modu:=2;
    if Modu=1 then
        Modulation:='Pi/4QPSK'
    else if Modu=2 then
        Modulation:='QPSK';

```

```

assign(filedat,'d:\htem\raisemax.dat');
rewrite(filedat);
assign(file1,'d:\htem\fchan75.dat');
assign(file2,'d:\htem\schan75.dat');
raisedcosine;
dt:=3;
for delayray:=1 to 14 do
  main;
repeat
  delayray:=delayray+2;
  main;
until delayray>25;
close(filedat);
close(file1);
close(file2);
end.

```

```

{*****
FILENAME: commax.pas
CONTAIN UNITS: CRT, COMKALM, COMPTU, RAISECHA.
CONTAIN:      Viterbi, initialvi, temdec,
DESCRIPTION: This unit contains the Viterbi algorithm which
              uses the impulse response of the CIRE to test
              the possible data sequences and choose one with
              the maximum probability as the estimated output
              *****)
unit commax;
interface
  uses crt,comkalm,comptu,raisecha;
  const Lay=4;{# of states per symbol, 4 for QPSK}
  type vapa=record
    value:real;
    path:array[0..19] of complex ;
  end;
  var  dec:array[0..L] of complex; {L is the length of CIRE }
      noise:real;
      J:array[1..Lay] of vapa;
  procedure Viterbi(start:boolean);
  procedure initialvi;
implementation

{*****
PROCEDURE: temdec
DESCRIPTION: transfer the state number k to the symbol value tran
*****}

  procedure temdec(var tran:complex; k:integer);
  begin
    case k of
      1: begin tran.re:=-1;tran.im:=-1; end;
      2: begin tran.re:=-1;tran.im:=1; end;
      3: begin tran.re:=1;tran.im:=-1; end;
      4: begin tran.re:=1;tran.im:=1; end;
    end;
  end;

{*****
PROCEDURE: Viterbi;
DESCRIPTION: Using Viterbi algorithm to test different data
              sequence.
VARIABLES:   finaldecision: the final detected data sequence
              w: the tap gain vector of the CIRE.
*****}
  procedure Viterbi(start:boolean);
  var
    stage:integer;
    finalvalue:real;
    i,finalpath:integer;
    cq:complex;
    rq:real;
    fff:real;
  procedure firststep;
  var i,k:integer;
      sum:complex;

      a:array [1..Lay] of complex;
  begin
    sum:=c0;
    for i:=1 to L do
      begin
        conjugate(cq,w[i+1]);

```

```

        cmul(cq,cq,findecision[i-1]);
        cadd(sum,sum,cq);
    end;
    for k:=1 to Lay do
    begin
        temdec(a[k],k);
        conjugate(cq,w[1]);
        cmul(cq,cq,a[k]);
        cadd(cq,cq,sum);
        csub(cq,z[L+1],cq);
        J[k].value:=(cq.re*cq.re+cq.im*cq.im);
        for i:=1 to L do
            J[k].path[i]:=findecision[i-1];
        J[k].path[0]:=a[k];
    end;
    rq:=-10000000;
    for i:=1 to Lay do
        if J[i].value>rq then
            begin
                rq:=J[i].value;
                dec[L]:=a[i];
            end;
    end;(firststep)

procedure Onestep;
var i,k:integer;
    check,F:array [1..Lay,1..Lay] of real;
    cq,fem:complex;
    Jmem:array[1..Lay,0..19] of complex;
    maxc: array[1..Lay] of real;
    num:array[1..Lay] of integer;
    a:array [1..Lay] of complex;
    ch:real;
begin
    for i:=1 to Lay do
    begin
        fem:=c0;
        for k:=1 to L do
            begin
                conjugate(cq,w[k+1]);
                cmul(cq,cq,J[i].path[k-1]);
                cadd(fem,fem,cq);
            end;
        for k:=1 to Lay do
            begin
                temdec(a[k],k);
                conjugate(cq,w[1]);
                cmul(cq,cq,a[k]);
                cadd(cq,cq,fem);
                csub(cq,z[L+1-stage],cq);
                F[i][k]:=cq.re*cq.re+cq.im*cq.im;
                check[i][k]:=J[i].value-F[i][k];
            end;
        end;
    end;

    for k:=1 to Lay do
    begin
        maxc[k]:=check[1][k]; num[k]:=1;
        for i:=2 to Lay do
            begin
                if check[i][k]>maxc[k] then
                    begin
                        maxc[k]:=check[i][k];
                        num[k]:=i;
                    end;
            end;
        end;
    end;
end;

```

```

        end;
    end;
    ch:=-1000000;
    for i:=1 to Lay do
        if maxc[i]>ch then
            begin
                ch:=maxc[i];
                dec[L-stage]:=a[i];
            end;
        end;

    for k:=1 to Lay do
        for i:=0 to 19 do
            Jmem[k][i]:=J[k].path[i];

        for k:=1 to Lay do
            begin
                J[k].value:=maxc[k];
                for i:=0 to 18 do
                    begin
                        J[k].path[19-i]:=Jmem[num[k]][18-i];
                    end;
                J[k].path[0]:=a[k];
            end;
        end;
    end;{Onestep}

begin
    stage:=0;
    if start=true then
        firststep
    else onestep;
    fff:=1;
    for stage:=1 to L do
        begin
            Onestep;
            fff:=1;
        end;
    finalvalue:=J[1].value;
    finalpath:=1;
    for i:=2 to Lay do
        begin
            if J[i].value>finalvalue then
                begin
                    finalvalue:=J[i].value;
                    finalpath:=i;
                end;
            end;
        for i:=0 to L do
            findecision[i]:=J[finalpath].path[i+L+1];
        for i:=1 to Lay do
            J[i].value:=J[i].value-finalvalue;
        end;{Viterbi}

{*****
PROCEUDRE: initialvi;
DESCRIPTION: initialize the Viterbi decoder
*****}
procedure initialvi;
var
    i,k:integer;
begin
    for i:=0 to 19 do
        begin
            for k:=1 to 4 do
                J[k].path[i]:=c0;
            findecision[i]:=c0;

```

```
end;  
for i:=1 to 4 do  
  J[i].value:=0;  
end;{initial}  
end.
```

```

{*****
FILENAME: INDOOR.PAS
CONTAIN UNITS: CRT, COMPTU, DFEKAL, SIRCHA, RAISED, SAMCHA;
CONTAIN: inputdata
        checkcb
        setsize
        setnoise
        main
PORPUSE: This program uses DFE-Kalman to evaluate the performance on
        SIRCIM(SMRCIM) channel model
*****}
program indoor;
{$N+}
    uses crt,comptu,dfekal,sircha,raised,samcha;
    var id,m1,ii,numoferr,totalnum,totalerror:longint;
        nnn:longint;
        db,rol:integer;
        mse:real;
        noise:real;
        filedat:text;
        filename,fn2:string[30];
        training:boolean;
        pwra,noi:array[1..10] of real;
        result:array[1..10] of real;
        cum:real;
{*****
PROCEDURE: inputdata;
DESCRIPTION: to obtain the output of the channel
*****}
    procedure inputdata;
    var nu1,nu2:integer;
    begin
        mobilechannel(noise);
    end;

{*****
PROCEDURE: checkcb;
DESCRIPTION: compared the decision and original symbols to
            see if error exists
INPUT PARMAS: a,b: data symbol and detected
            n: the previous accumulated error number

OUTPUT PARMAS: n: the current accumulated error number
*****}
    procedure checkcb(var n:longint; a,b:complex);
    begin
        if round(a.re)<>round(b.re) then n:=n+1;
        if round(a.im)<>round(b.im) then n:=n+1;
    end;
{*****
PROCEDURE: reini
DESCRIPTION: Reinitializes the system
CALL:      initialKa;
          inputdata;
          CoKal;
*****}
    procedure reini;
    var i,m1,traintime:integer;
    begin
        initialKa;
        for ii:=1 to 20 do inputchannel;
        for ii:=1 to 10 do
            inputdata;
        mse:=0;

```

```

        if modulation='QPSK' then
            traintime:=30
        else traintime:=100; (# of training number)
        for m1:=1 to traintime do
            begin
                inputdata;
                training:=true;
                CoKal(training,inp[1],xin,modulation);
                mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
            end; writeln(mse/traintime);
        end;
    {*****}
    FUNCTION: nspwr:
    DESCRIPTION: Converse Eb/No to noise power
    INPUT PARMAS: EbNo: Eb/No value;
    OUTPUT PARMAS: nspwr: the noise power
    {*****}
    function nspwr(EbNo:real):real;

    begin
        EbNo:=exp(EbNo/10*ln(10));
        nspwr:=2/(4*EbNo);
    end;

    {*****}
    PROCEDURE: setnoise
    DESCRIPTION: the different noise power
    {*****}
    procedure setnoise;
    begin
        noi[1]:=nspwr(3);
        noi[2]:=0.1;
        noi[3]:=0.05;
        noi[4]:=0.025;
        noi[5]:=0.01;
        noi[6]:=0.005;
        noi[7]:=0.0005;
        noi[8]:=0;
    end;
    {*****}
    PROCEDURE: main;
    DESCRIPTION: Simulate 18000 bits evenly distributed
                  using the chosen paramaters
    INPUT PARMAS: xin; symbol sent
                  d: detected symbol
                  modulation: QPSK OR DQPSK
                  filedatre,filedatim: the channel data of real and im profiles
    OUTPUT PARMAS: totalerror: total number of error in the 18000 bits
                  totalnum: total number bits sent
                  result[]: recorded the BER results
    CALL:
            initialka
            inputdata
            Cokal
            mcini
    {*****}
    procedure main;
    begin
        reset(filedatre);
        reset(filedatim);
        xu:=c0;
        d:=c0;
        xin:=c0;
        totalnum:=0;
        totalerror:=0;
        mcini;
    end;

```

```

        inisamcha;
        initialka;
        nnn=-1; distance:=-1/18+0.00001;
        repeat
            nnn=nnn+1; distance:=nnn/18+0.00001;
            area:=nnn;
            reini;
            mse:=0;
            for ii:=1 to 50 do
                begin
                    mse:=0;
                    numoferr:=0;
                    lambda:=0.99;
                    for m1:=1 to 10 do
                        begin
                            inputdata;
                            training:=true;
                            CoKal(training,inp[1],xin,modulation);
                                ( nonadaptive; )
                                ( coe:=0.1;
                                    CoLMS(true,inp[1],xin,modulation);
                                )
                            checkcb(numoferr,din[delay+1],d);
                            mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
                        end;
                    totalnum:=totalnum+20;
                    totalerror:=totalerror+numoferr;
                    mse:=mse/10;
                    writeln(noise:10:6,' ',totalnum,' ',mse/2:12,' # of error: ',numoferr,'/20',t
                end;
            until nnn=17;(totalnum>8000;)}

        end;

{*****
PROCEDURE: setsize;
DESCRIPTION: Set some initail value
OUTPUT PARMAS: fn2: the SIRCIM channel name:
                n1: forward tap #
                n2: feedback tap #
                delay: the delay of the equalizer
                lambda: weighing factor
*****}
        procedure setsize;
        begin
            fn2:='HARD122';
            n1:=2;delay:=n1-1;
            n2:=2;
            tN:=n1+n2;
            lambda:=0.99;
            delta:=1e6 ;
            dt:=3;
        end;

{*****
Main program control the parmeters and simulation
*****}
        begin
            randomize;
            for i:=1 to 10 do
                result[i]:=100;
            Modulation:='QPSK';
            raisedcosine;
            speed:=10;
            symbolrate:=1000000;

```

```

assign(filedatre,'c:\htemp\HARD1RE.dat');
assign(filedatim,'c:\htemp\HARD1IM.dat');
reset(filedatre);
reset(filedatim);
setsize;
assign(filedat,'c:\htemp\' +fn2+'.dat');
rewrite(filedat);
setnoise;
    writeln(filedat);
    rolloff:=0.35;
    noise:=0;
    db:=0;
    pwrratiodb:=2*db;
    for i:=1 to 4 do
    begin
        noise:=noi[i];
        main;
        writeln(filedat,noise,' ',totalerror/totalnum);
        result[i]:=totalerror/totalnum;
    end;

close(filedat);
close(filedatre);
close(filedatim);
end.{end of main}

```

```

{*****
FILENAME: URBAN.PAS
CONTAIN UNITS: COMPTU, DFEKAL, SIRCHA, RAISED, SAMCHA.
CONTAIN: inputdata
        checkcb
        setsize
        setnoise
        main
PURPOSE: This program uses DFE-Kalman to evaluate its performance
        on SMRCIM urban channel model
*****}
program urban;
{$N+}
    uses comptu,dfekal,sircha,raised,samcha;
    var id,m1,ii,numoferr,totalnum,totalerror:longint;
        nnn:longint;
        totnum,db,rol,length:integer;
        mse:real;
        noise:real;
        filedat:text;
        ss,smrname,filename:string[30];
        training:boolean;
        pwra,noi:array[1..10] of real;
        result:array[1..10] of real;
        cum:real;
{*****
PROCEDURE: inputdata
DESCRIPTION: Get an output from the SMRCIM channel
*****}
    procedure inputdata;
    var nu1,nu2:integer;
    begin
        mobilechannel(noise);
    end;

{*****
PROCEDURE: checkcb;
DESCRIPTION: Compared the decision and original symbol
            to see if there are errors
INPUT PARAMS: a,b: data and detected symbols
              n: the previous accumulated error number
OUTPUT PARAMS: n: the current accumulated error number
*****}
    procedure checkcb(var n:longint; a,b:complex);
    begin
        if round(a.re)<>round(b.re) then n:=n+1;
        if round(a.im)<>round(b.im) then n:=n+1;
    end;

{*****
PROCEDURE: reini;
DESCRIPTION: reinitailize the system
*****}
    procedure reini;
    var traintime,i,m1:integer;

    begin
        for ii:=1 to 20 do inputchannel;
        for ii:=1 to 10 do
            inputdata;
            mse:=0;
            if modulation='QPSK' then
                traintime:=13
            else traintime:=30;

```

```

        for m1:=1 to traintime do
        begin
            inputdata;
            training:=true;
            CoKal(training,inp[1],xin,modulation);
            mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
            end; writeln(mse/20);
        end;

{*****}
PROCEDURE: setsize
DESCRIPTION: Set some initial value
OUTPUT PARMAS: n1: forward tap #
                n2: feedback tap #
                delay : delay of the equalizer
                lambda: weighting factor
{*****}
procedure setsize;
begin
    n1:=2;delay:=n1-1;
    n2:=1;
    tN:=n1+n2;
    lambda:=0.99;
    delta:=1e6;
    dt:=3;
end;
{*****}
FUNCTION: nspwr
DESCRIPTION: transfer Eb/No to noise power
INPUT PARMAS: EbNo: Eb/No value
OUTPUT PARMAS: nspwr: noise power value;
{*****}

function nspwr(EbNo:real):real;
begin
    EbNo:=exp(EbNo/10*ln(10));
    nspwr:=2/(4*EbNo);
end;

{*****}
PROCEDURE: setnoise
DESCRIPTION: Choose some noise power
OUTPUT PARMAS: noi[]: noise power
{*****}
procedure setnoise;
begin
    noi[1]:=0.25;
    noi[2]:=0.1;
    noi[3]:=0.05;
    noi[4]:=0.025;
    noi[5]:=0.01;
    noi[6]:=0.005;
    noi[7]:=0.0005;
    noi[8]:=0;
end;
{*****}
PROCEDURE: main
DESCRIPTION: simulation over the one meter SMRCIM channel
INPUT PARMAS: xin: symbol sent
                d: symbol detected
                modulation: QPSK and DQPSK
OUTPUT PARMAS: totalerror; total number of the error

```

```

        totalnum: total number of bits sent
        result[]: recorded results
CALLS:   initialKa
        inputdata
        CoKal
        mcini
*****}
procedure main;
var i:integer;
begin
{   reset(file1);
    reset(file2); }
    reset(filedatre);
    reset(filedatim);
    xu:=c0;
    d:=c0;
    xin:=c0;
    totalnum:=0;
    totalerror:=0;
    mcini;
    inisamcha;
    initialKa;
    reini;
    mse:=0;
    repeat
        mse:=0;
        numoferr:=0;
        lambda:=0.9;
        i:=0;
        while (over=false) and (i<10) do
            begin
                i:=i+1;
                inputdata;
                training:=true;
                CoKal(training,inp[1],xin,modulation);
                    { nonadaptive;}
                    {   coe:=0.1;
                        CoLMS(true,inp[1],xin,modulation);
                    }
                checkcb(numoferr,din[delay+1],d);
                mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
            end;
            totalnum:=totalnum+20;
            totalerror:=totalerror+numoferr;
            mse:=mse/10;
            writeln(noise:10:6,' ',totalnum,' ',mse/2:12,' # of error: ',numoferr,'/20',tot
        until over {or totalnum>8000;}
    end;

{*****}
Main program to control the parmaters and simulation
variables:  Modulation: QPSK
            rolloff: Roll off factor of the raised cosine filter
            noise: noise power;
            speed: vehicle speed (km/h)
            symbolrate: symbol rate(sps)
            totalnum: total number of bits sent
            totalerror: total number of bit errors
            filedatre, filedatim:
                input channel cubic spline data.
                It should be changed for differet channels
            filedat: BER output file
*****}
begin
for totnum:=4 to 10 do
begin

```

```

for i:=1 to 10 do
    result[i]:=100;
Modulation:='QPSK';
length:=1;
str(totnum,ss);
smrname:='small'+ss;
raisedcosine;
assign(filedatre,'c:\htemp\smrdata\' + smrname + 're.dat');
assign(filedatim,'c:\htemp\smrdata\' + smrname + 'im.dat');
reset(filedatre);
reset(filedatim);
assign(filedat,'c:\htemp\smrout\' + smrname + '.s20');
rewrite(filedat);
setsize;
setnoise;
    writeln(filedat);
    rolloff:=0.35;
    noise:=0;
    db:=0;
    speed:=20;
    symbolrate:=24300;
    for i:=1 to 6 do
        begin
            noise:=noi[i];
            main;
            writeln(filedat,noise,' ',totalerror/totalnum);
            result[i]:=totalerror/totalnum;
        end;

    close(filedat);
    close(filedatre);
    close(filedatim);
end;
end.{end of main}

```

```

{*****
FILENAME: sircha.pas
CONTAIN UNITS: COMPTU, RAISED, SAMCHA.
CONTAIN: mobilechannel
      mcini;
DESCRIPTION: Using SIRCIM channel to form the impulse response
      and send data through the channel
*****}
Unit sircha;
interface
  uses comptu,raised,samcha;
  var anp,inp:array[1..200] of complex;
      anpre,xin,apre:complex;
      dase,ideald:array [1..10] of complex;
      cr,xl:array [1..200] of complex;
      amp,pwr:array[1..2] of real;
      nnn,delayray,dt,cl:integer;
      Ac,pwrratioDB:real;
      Modulation:string[10];
      file1,file2:text;
  procedure mobilechannel(noise:real);
  procedure mcini;
implementation
  var tnoi:real;
{*****
PROCEDURE: putraised
DESCRIPTION: Convolve the channel impulse response with the raised-cosine
      filter.
INPUT PARMAS: c[]: the channel impulse response
      raisedfilter[]: the impulse response of RC filter
      cl: the time duration of the channel
OUTPUT PARMAS: cr: the impulse response of channel and RC filter
*****}
  procedure putraised;
  var i,n,k:integer;
      q:real;
      ctemp:complex;
  begin
    for i:=1 to 69 do
    begin
      cr[i]:=c0;
      n:=0;
      repeat
        n:=n+1;
        rmul(ctemp,c[n],raisedfilter[i+1-n]);
        cadd(cr[i],cr[i],ctemp);
      (
        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; )
      until (n=cl) or (i+1-n=1);
    end;
    k:=0;
    for i:=70 to 68+cl do
    begin
      cr[i]:=c0;
      k:=k+1;n:=k;
      repeat
        n:=n+1;
        rmul(ctemp,c[n],raisedfilter[i+1-n]);
        cadd(cr[i],cr[i],ctemp);
      (
        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n];)
      until n=cl;
    end;
  end;
{*****
PROCEDURE: mobilechannel

```

```

DESCRIPTION: convolving the channel with the random data to form
              the output of the channel
INPUT PARMAS: file1,file2: the two ray channel file
              noise: the noise power;
OUTPUT PARMAS: inp[1]: the output of the channel
              xin: the data sent
*****}
procedure mobilechannel(noise:real);
const p=2;
var i,n:integer;
    rt,q:real;
    a,temp:complex;
function gauss:real;
var temp:real;
    i:integer;
begin
    temp:=0;
    for i:=1 to 12 do
        temp:=temp+random;
    gauss:=temp-6;
end;

begin
    inputchannel;{to obtain the channel impulse response}
    cl:=totofbn+1;{Total length of the channel impulse response}
(This is for coherent detection)
    putraised;
    for i:=0 to 8 do
        begin
            dase[10-i]:=dase[9-i];
            ideald[10-i]:=ideald[9-i];
        end;
    if random>0.5 then ideald[1].re:=1
    else ideald[1].re:=-1;
    if random>0.5 then ideald[1].im:=1
    else ideald[1].im:=-1;      {form the random data}
    dase[1]:=ideald[1]; {if noise is here, this to be deleted}
    a:=dase[1];

    for i:=12 downto 1 do
        begin
            for n:=0 to 98 do
                xl[100-n]:=xl[99-n];
            if i=12 then
                xl[1]:=a
            else xl[1]:=c0;
            anp[i].re:=0;
            anp[i].im:=0;
            for n:=1 to 68+cl do
                begin
                    cmul(temp,cr[n],xl[n]);
                    cadd(anp[i],anp[i],temp);
                end;
            end;
        end;
    xin:=ideald[dt]; {the data sent}
    apre:=a;
    anp[1]:=anp[2];
    if Modulation='QPSK' then
        inp[1]:=anp[1]; {output of the channel}

    tnoi:=sqrt(noise);
    temp.re:=gauss*tnoi;
    temp.im:=gauss*tnoi;
    cadd(inp[1],inp[1],temp); {Add noise}

```

```

end;(of complex channel)

{*****
PROCEDURE: mcini;
DESCRIPTION: initialize the channel;
*****}
procedure mcini;
begin
  for nnn:=1 to 200 do
  begin
    anp[nnn].re:=1;
    anp[nnn].im:=1;
    inp[nnn].re:=1;
    inp[nnn].im:=1;
    xl[nnn].re:=0;
    xl[nnn].im:=0;
    cr[nnn].re:=0;
    cr[nnn].im:=0;
  end;
  for nnn:=1 to 10 do
  begin
    ideald[nnn]:=c0;
    dase[nnn]:=c0;
  end;
  apre.re:=sqrt(2);
  apre.im:=0;
  anpre.re:=1;
  anpre.im:=1;
end;
end.

```

```

{*****
FILENAME: samcha.pas
CONTAIN UNITS: COMPTU;
CONTAIN: inputchannel;
        inismacha;
DESCRIPTION: Obtain the SIRCIM or SMRCIM channel files
              and use the interpolation to send the
              channel impulse response at a specific
              location to SIRCHA.
*****}
unit samcha;
interface
  uses comptu;
  var c:array[0..90] of complex;
      area,totofbin,kkj:integer;
      distance,step,speed,symbolrate:real;
      ax,bx,cx,dx: array[0..90] of complex;
      filedatre,filedatim:text;
      startpoint,over :boolean;
  procedure inputchannel;
  procedure inisamcha;
implementation
{*****
PROCEDURE: checktotbin:
DESCRIPTION: check the length of the channel impulse response
*****}
  var tracklength:real;
  procedure checktotbin;
  begin
    readln(filedatre,totofbin,tracklength);
    readln(filedatim);
  end;

{*****
FUNCTION: fx
DESCRIPTION: using the function of cubic spline interpolation
              to recover the value of location x
INPUT PARMAS: a,b,c,d: paramaters of the cubic spline function
              x0,x1: the range of the cubic spline
              x: the location of the value wanted to know
OUTPUT PARMAS: fx: the value of the location
*****}
  function fx(x,x0,x1:real;a,b,c,d:real):real;
  var t0,t1:real;
      g:real;
  begin
    t1:=x1-x;
    t0:=x-x0;
    g:=a*t1*t1*t1+b*t0*t0*t0+c*t0+d*t1;
    fx:=g;
  end;

{*****
PROCEDURE: readdata;
DESCRIPTION: read in the cubic spline paramaters from the data
              files
INPUT PARMAS: totofbin
OUTPUT PARMAS: ax,bx,cx,dx: the cubic spline paramaters of
              different time bins
*****}

  procedure readdata;
  var n,k:integer;
  begin

```

```

    for n:=0 to totofbin do
    begin
        readln(filedatre,k,k,ax[n].re,bx[n].re,cx[n].re,dx[n].re);
        readln(filedatim,k,k,ax[n].im,bx[n].im,cx[n].im,dx[n].im);
    end;
    area:=area+1;
end;

{*****
PROCEDURE: setsample
DESCRIPTION: Locate the range of a point, set the step size
             to compute the value of the point
INPUT PARMAS: area: the area of the point;
              speed: the speed of the mobile (in km/h)
              symbolrate: the baud rate (sps)
OUTPUT PARMAS: c: the value of position at distance
               distance: the next point's distance to the origin.
*****}
procedure setsample;
var i:integer;
    stepdis,x0,x1:real;
begin
    x0:=tracklength*(area-1)/18;
    x1:=tracklength*area/18;
    stepdis:=speed/3.6/symbolrate;
    for i:=0 to totofbin do
    begin
        c[i+1].re:=fx(distance,x0,x1,ax[i].re,bx[i].re,cx[i].re,dx[i].re);{the real val
        c[i+1].im:=fx(distance,x0,x1,ax[i].im,bx[i].im,cx[i].im,dx[i].im);{the im value
    end;
    distance:=distance+stepdis;{Distance to the origin}
end;

{*****
PROCEDURE: checkdistance:
DESCRIPTION: check if the distance if over one specific area
             then inport new paramaters for the cubic spline.
             Also check if the total channel is over
*****}
procedure checkdistance;
var i,n:integer;
    check:boolean;
begin
    if distance>area/18 then
    begin
        readdata;
    end;
    if area=19 then
    begin
        over:=true;
    end;
end;

{*****
PROCEDURE: inputchannel
DESCRIPTION: to get the new channel value of each distance
*****}
procedure inputchannel;
begin
    if startpoint=true then
    begin
        checktotbin;
        readdata;
        setsample;
        startpoint:=false;
    end
end

```

```

        else
        begin
            checkdistance;
            setsample;
        end;
    end;
{*****
PROCEDURE: inisamcha;
DESCRIPTION: Initialization
*****}
    procedure inisamcha;
    var i:integer;
    begin
        for i:=0 to 50 do
            c[i]:=c0;
            startpoint:=true;
            over:=false;
            distance:=0;
            area:=0;
        end;
    end.

```

```

{*****
FILENAME: DIFKAL.PAS
CONTAIN UNITS: CRT, COMPTU, DRAISECH, RAISED, KALDQPSK.
CONTAIN: inputdata, checkcb, setsize,
        setnoise, reini, main.
DESCRIPTION: This program implements DFE into pi/4 DQPSK
*****}
program DIFKAL;
{$N+}
uses crt,comptu,draisech,raised,kaldqpsk;
var cuu,i1,i2,m1,ii,numoferr,totalnum,totalerror:longint;
    nnn:longint;
    db,rol:integer;
    mse:real;
    noise:real;
    filedat:text;
    filename:string[30];
    training:boolean;
    pwra,noi,la:array[1..10] of real;
    result:array [1..30] of real;
{*****
PROCEDURE: inputdata;
DESCRIPTION: Get an output from mobile channel
*****}
procedure inputdata;
var nu1,nu2:integer;
begin
    mobilechannel(noise);
end;
{*****
PROCEDURE: checkcb
DESCRIPTION: Check if detected symbols are right
VARIABLES:
    Input n: accumulated bit-error #
            a,b:one is the true data and the other detected
*****}
procedure checkcb(var n:longint; a,b:complex);
begin
    if round(a.re)<>round(b.re) then n:=n+1;
    if round(a.im)<>round(b.im) then n:=n+1;
end;
{*****
PROCEDURE: setsize;
DESCRIPTION: Set size of DFE, weighting factor, delay
VARIABLES
    n1: DFE forward #
    n2: DFE backward #
    lambda: weighting factor 0..1
    dt: delay
*****}
procedure setsize;
begin
    n1:=3;delay:=n1-1;
    n2:=2;
    tN:=n1+n2;
    lambda:=0.90;
    delta:=1e6;
    dt:=3;
end;
{*****
PROCEDURE: setnoise
DESCRIPTION: Set noise power
*****}
procedure setnoise;

```

```

begin
    noi[1]:=0.158;
    noi[2]:=0.1;
    noi[3]:=0.05;
    noi[4]:=0.025;
    noi[5]:=0.0158;
    noi[6]:=0.01;
    noi[7]:=0.005;
    la[1]:=0.85;
    la[2]:=0.9;

    end;
{*****}
PROCEDURE: reini;
DESCRIPTION: reinitialize the DFE and RLS algorithm and train
              DFE.
{*****}
procedure reini;
begin
    initialka;
    for m1:=1 to 30 do
        begin
            lambda:=0.9;
            inputdata;
            training:=true;
            CoKal(training);
            mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
            end; writeln(mse/30);
        end;
{*****}
PROCEDURE: main;
DESCRIPTION: obtain the BER of one specific point
VARIABLES: lambda: weighting factor
            totalnum: total number of bits
            totalerror: total number of bit errors
            mse: local mean square error
{*****}
procedure main;
begin
    reset(file1);
    reset(file2);
    xu:=c0;
    d:=c0;
    xin:=c0;
    nnn:=0;
    totalnum:=0;
    noise:=0;
    totalerror:=0;
    mcini;
    initialKa;
    for ii:=1 to 10 do
        inputdata;
        mse:=0;
        for ii:=1 to 1 do
            begin
                mse:=0;
                for m1:=1 to 10 do
                    begin
                        lambda:=0.9;
                        inputdata;
                        training:=true;
                        CoKal(training);
                        mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
                        end; writeln(mse/10);
                    end;
                end;
            end;
        end;
    end;

```

```

for ii:=1 to 2 do
begin
mse:=0;
for m1:=1 to 10 do
begin
lambda:=0.9;
inputdata;
training:=true;
CoKal(training);
mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
end; writeln(mse/10);
end;
repeat
mse:=0;
nnn:=0;
for ii:=1 to 20 do
begin
nnn:=nnn+1;
mse:=0;
numoferr:=0;
for m1:=1 to 10 do
begin
noise:=0.;
lambda:=0.9;
inputdata;
training:=true;
CoKal(training);
checkcb(numoferr,td[delay+1],d);
mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
end;
totalnum:=totalnum+20;
totalerror:=totalerror+numoferr;
mse:=mse/10;
writeln(delayray,' ',totalnum,mse/2,' # of error: ',numoferr,'/20',totalerror
end;
( reini; )
until totalnum>9600;
writeln(filedat,delayray,' ',totalerror/totalnum);
result[delayray]:=totalerror/totalnum;
end;

{*****
Main program
DESCRIPTION: control the operation of the simulation
VARIABLES : file1,file2: Rayleigh fading channel files
            generated by BERSIM
            filedat: BER output file
            noise: noise power
            delayray: delay of the second ray (13=T, 1=0)
*****}

begin
assign(file1,'c:\htemp\fchan100.dat');
assign(file2,'c:\htemp\schan100.dat');
reset(file1);
reset(file2);
assign(filedat,'c:\htemp\D1003290.dat');
rewrite(filedat);
raisedcosine;
setsize;
noise:=0;
setnoise;
for delayray:=1 to 13 do
main;
repeat

```

```
    delayray:=delayray+2;
    main;
until delayray>25;
close(file1);
close(file2);
close(filedat);
end.(end of main)
```

```

{*****
FILENAME: KALDQPSK;
CONTAIN UNITS: CRT, COMPTU, DSIRCHA;
CONTAIN: Move, CoKal, initialKa, reinitialKa
DESCRIPTION: Executes an iteration of Complex DFE-Kalman algorithm
              for Pi/4 DQPSK
*****}
unit kaldqpsk;
interface
  uses crt,comptu,dsircha;
  type
    data1=array[1..20] of complex;
  var
    n1,n2,tN:integer;
    findecision:array [0..19] of complex;
    u,td,din:array [1..20] of complex;
    x,w,kk:data1;
    Pv:array [1..20] of data1;
    prealpha,sum,alpha,xu,d,tu:complex;
    z:array [0..20] of complex;
    weigh,lambda,delta,mse,coe:real;
    OvDelay,number,count,delay:integer;
    dif:complex;
    test:boolean;
  procedure Move;
  procedure CoKal(training:boolean);
  procedure initialKa;
  procedure reinitialKa;
  procedure CoLMS(training:boolean);
implementation
  var lin:real;
{*****
PROCEDURE: Move;
DESCRIPTION: Shift the registers
INPUT PARMAS: inp: output of the channels
              xin: data sent
OUTPUT PARMAS: u: input vector
              din: decision vector;
*****}
  procedure Move;
  var i:integer;
  begin
    for i:=0 to n1-2 do
      u[n1-i]:=u[n1-1-i];
    u[1]:=inp[1];
    for i:=0 to n2-2 do
      u[tN-i]:=u[tN-1-i];
    u[n1+1]:=din[delay+1];
    for i:=0 to 18 do
      td[20-i]:=td[19-i];
    td[1]:=xin;
    for i:=0 to 18 do
      din[20-i]:=din[19-i];
    cmul(din[1],xin,din[2]);
    lin:=sqrt(2)/2;
    rmul(din[1],din[1],lin);
  end;

{*****
PROCEDURE: CoKal
DESCRIPTION: Executes an iteration of DFE-Kalman
INPUT PARMAS: training: in the training or not
              inp: input of the equalizer
              xin: data sent
OUTPUT PARMAS: d: decision

```

```

        alpha: prediction error
*****}
procedure CoKal(training:boolean);
var i,ik:integer;
    at,bt,ct,calp:complex;
    cv,cu:data1;
    amt,rn:real;
    nxu:complex;

begin
    for i:=0 to n1-2 do
        u[n1-i]:=u[n1-1-i];
    u[1]:=inp[1];
    for i:=0 to n2-2 do
        u[tN-i]:=u[tN-1-i];
    if training=true then
        u[n1+1]:=din[delay+1]
    else begin
        rn:=sqrt(2)/2;
        rmul(at,d,rn);
        cmul(u[n1+1],u[n1+2],at);
    end;
    for i:=0 to 18 do
        td[20-i]:=td[19-i];
    td[1]:=xin;
    for i:=0 to 18 do
        din[20-i]:=din[19-i];
    cmul(din[1],xin,din[2]);
    lin:=sqrt(2)/2;
    rmul(din[1],din[1],lin);

    alpha:=c0;
    for i:=1 to tN do
    begin
        conjugate(ct,w[i]);
        cmul(ct,ct,u[i]);
        cadd(alpha,alpha,ct);
    end;
    cdiv(dif,alpha,prealpha);
    if dif.re>0 then d.re:=1 else d.re:=-1;
    if dif.im>0 then d.im:=1 else d.im:=-1;
    if training=true then xu:=td[delay+1]
    else xu:=d;
    rmul(nxu,xu,lin);

    amt:=1/sqrt(prealpha.re*prealpha.re+prealpha.im*prealpha.im);
    rmul(prealpha,prealpha,amt);
    cmul(tu,nxu,prealpha);
    prealpha:=alpha;

    csub(alpha,tu,alpha);

    for i:=1 to tN do
    begin
        x[i]:=c0;
        for ik:=1 to tN do
            begin
                cmul(ct,Pv[i,ik],u[ik]);
                cadd(x[i],x[i],ct);
            end;
    end;
    sum.re:=lambda;
    sum.im:=0;

```

```

for i:=1 to tN do
begin
  conjugate(ct,u[i]);
  cmul(ct,ct,x[i]);
  cadd(sum,sum,ct);
end;
for i:=1 to tN do
begin
  cdiv(kk[i],x[i],sum);
end;

conjugate(calp,alpha);
for i:=1 to tN do
begin
  cmul(ct,kk[i],calp);
  cadd(w[i],w[i],ct);
end;

for i:=1 to tN do
  conjugate(cu[i],u[i]);
for i:=1 to tN do
begin
  cv[i]:=c0;
  for ik:=1 to tN do
  begin
    cmul(ct,cu[ik],Pv[ik,i]);
    cadd(cv[i],cv[i],ct);
  end;
end;

for i:=1 to tN do
  for ik:=1 to tN do
  begin
    cmul(ct,kk[i],cv[ik]);
    csub(Pv[i][ik],Pv[i][ik],ct);
    rmul(Pv[i][ik],Pv[i][ik],1/lambda);
  end;
end;
end;
procedure CoLMS(training:boolean);
var ct,q,cerror:complex;
    tem:real;
    i:integer;
begin
  tem:=0;
  for i:=1 to tN do
    tem:=tem+u[i].re*u[i].re+u[i].im*u[i].im;
  weigh:=coe/tem;
  move;
  alpha:=c0;
  for i:=1 to tN do
  begin
    conjugate(ct,w[i]);
    cmul(ct,ct,u[i]);
    cadd(alpha,alpha,ct);
  end;
  csub(alpha,din[1],alpha);

  conjugate(cerror,alpha);
  for i:=1 to tN do
  begin
    rmul(q,u[i],weigh);
    cmul(q,cerror,q);
    csub(w[i],w[i],q);
  end;
end;
end;

```

```

{*****}
Procedure: initialKa;
DESCRIPTION: Initialize the equalizer and algorithm
{*****}

procedure initialKa;
var il,ill :integer;
begin
  delta:=100000;
  for il:=1 to 20 do
    begin
      for ill:=1 to 20 do
        Pv[il,ill]:=c0;
        Pv[il,il].re:=delta;
        w[il].re:=0.3;
        w[il].im:=0;
      end;
      for il:=1 to 20 do
        begin
          din[il]:=c0;
          z[il]:=c0;
          u[il]:=c0;
          td[il]:=c0;
        end;
        z[0]:=c0;
        din[1].re:=1;
        din[1].im:=0;
        d:=din[1];
        prealpha.re:=1;
        prealpha.im:=0;
      end;
    end;
  end;
{*****}
PROCEDURE: reinitialKa
DESCRIPTION: Reinitialize the equalizer and algorithm,
            do not set the tap-gain vector to zero.
{*****}

procedure reinitialKa;
var il,ill :integer;
begin
  delta:=100000;
  for il:=1 to 20 do
    begin
      for ill:=1 to 20 do
        Pv[il,ill]:=c0;
        Pv[il,il].re:=delta;
      end;
      prealpha.re:=1;
      prealpha.im:=0;
      for il:=1 to 20 do
        begin

        end;
      end;
    end;
  end;
end;
end.

```

```

{*****
FILENAME: DRAISECH.PAS
CONTAIN UNITS: RAISED, COMPTU;
CONTAIN: mobilechannel, mcini, putraised,
DESCRIPTION: Simulates the pi/4 DQPSK transmitter
              and the mobile channel
*****}
Unit draisech;
interface
  uses comptu,raised;
  const Ctodb=0.230258509;
  var anp,inp:array[1..100] of complex;
      anpre,xin,apre:complex;
      dase:array [1..10] of complex;
      cr,xl:array [1..100] of complex;
      c:array [1..30] of complex;
      amp,pwr:array[1..2] of real;
      nnn,delayray,dt,cl:integer;
      Ac,pwrratioDB:real;
      Modulation:string[10];
      file1,file2:text;
  procedure mobilechannel(noise:real);
  procedure mcini;
implementation
{*****
PROCEDURE: putraised
DESCRIPTION: convolve channel with the raised cosine filter.
INPUT PARAMS: c[]: channel impulse
OUTPUT PARAMS: cr[]: after convolving with raised-cosine filter.
*****}
  procedure putraised;
    var i,n,k:integer;
        q:real;
        ctemp:complex;
  begin
    for i:=1 to 69 do
    begin
      cr[i]:=c0;
      n:=0;
      repeat
        n:=n+1;
        rmul(ctemp,c[n],raisedfilter[i+1-n]);
        cadd(cr[i],cr[i],ctemp);
      {
        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; }
      until (n=cl) or (i+1-n=1);
    end;
    k:=0;
    for i:=70 to 68+cl do
    begin
      cr[i]:=c0;
      k:=k+1;n:=k;
      repeat
        n:=n+1;
        rmul(ctemp,c[n],raisedfilter[i+1-n]);
        cadd(cr[i],cr[i],ctemp);
      {
        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; }
      until n=cl;
    end;
  end;
{*****
PROCEDURE: mobilechannel
DESCRIPTION: Differential coding, convolving channel with
              random data to get an output of the channel
              and also add noise power
*****}

```

```

INPUT PARMAS: noise: noise power
OUTPUT PARMAS: xin: data sent
               inp[1]: output of the channel
*****}
procedure mobilechannel(noise:real);
const p=2;
var i,n:integer;
    tnoi,rt,q:real;
    a,temp:complex;
function gauss:real;
var temp:real;
    i:integer;
begin
    temp:=0;
    for i:=1 to 12 do
        temp:=temp+random;
    gauss:=temp-6;
end;
{*****}
PROCEDURE: powerratio
DESCRIPTION: change the mean power ratio of the two ray
*****}
procedure powerratio;
var i:integer;
begin
    rt:=exp(pwrratiotdB*CtodB);
    pwr[2]:=1/(1+rt);
    pwr[1]:=rt*pwr[2];
    for i:=1 to cl do
        begin
            amp[i]:=sqrt(pwr[i]*0.5)*2;
            rmul(c[i],c[i],amp[i]);
        end;
end;

begin
    cl:=delayray;
    readln(file1,c[1].re,c[1].im);
    readln(file2,c[delayray].re,c[delayray].im);
(
    c[1].re:=1;c[1].im:=0.;
    c[delayray].re:=0.5;c[delayray].im:=0.5;    )

{This is for coherent detection}
(
    for i:=1 to cl do
        begin
            c[i].re:=sqrt(c[i].re*c[i].re+c[i].im*c[i].im);
            c[i].im:=0;
        end;    )

    putraised;
(
    for i:=1 to 100 do
        begin
            cr[i].re:=sqrt(cr[i].re*cr[i].re+cr[i].im*cr[i].im);
            cr[i].im:=0;
        end;    )
    for i:=0 to 8 do
        dase[10-i]:=dase[9-i];

    if random>0.5 then dase[1].re:=1
    else dase[1].re:=-1;
    if random>0.5 then dase[1].im:=1
    else dase[1].im:=-1;

    cmul(a,dase[1],apre);

```

```

q:=sqrt(2)/2;
rmul(a,a,q);

{ anpre:=anp[1];}
for i:=12 downto 1 do
begin
  for n:=0 to 98 do
    xl[100-n]:=xl[99-n];
  if i=12 then
    xl[1]:=a
  else xl[1]:=c0;
  anp[i]:=c0;

  for n:=1 to 68+cl do
  begin
    cmul(temp,cr[n],xl[n]);
    cadd(anp[i],anp[i],temp);
  end;
end;
xin:=dase[dt];
apre:=a;

inp[1]:=anp[2];
tnoi:=sqrt(noise);
tnoi:=sqrt(noise);
temp.re:=gauss*sqrt(tnoi);
temp.im:=gauss*sqrt(tnoi);

cadd(inp[1],inp[1],temp);

end;(of complex channel)
{*****}
PROCEDURE: mcini
DESCRIPTION: Initialize the channel
{*****}
procedure mcini;
begin
  for nnn:=1 to 100 do
  begin
    anp[nnn].re:=1;
    anp[nnn].im:=1;
    inp[nnn].re:=1;
    inp[nnn].im:=1;
    xl[nnn].re:=0;
    xl[nnn].im:=0;
    cr[nnn].re:=0;
    cr[nnn].im:=0;
  end;
  for nnn:=1 to 30 do
  begin
    c[nnn].re:=0;
    c[nnn].im:=0;
  end;
  apre.re:=sqrt(2);
  apre.im:=0;
  anpre.re:=1;
  anpre.im:=1;
end;
end.

```

```

{*****}
FILENAME: DINDOOR.PAS
CONTAIN UNITS: COMPTU, DSIRCHA, SAMCHA, RAISED, KALDQPSK
CONTAIN: inputdata, setnoise, reini, main
DESCRIPTION: This program implements DFE into pi/4 DQPSK
              Using the data files of SIRCIM
              First BU2CUBIC in \sir has to be runned
{*****}
program DINDOOR;
{$N+}
  uses crt,comptu,dsircha,samcha,raised,kaldqpsk;
  var i,cuu,i1,i2,m1,ii,numoferr,totalnum,totalerror:longint;
      nnn:longint;
      db,rol:integer;
      mse:real;
      noise:real;
      filedat:text;
      name,filename:string[30];
      training:boolean;
      pwra,noi,la:array[1..10] of real;
      result:array [1..30] of real;
{*****}
PROCEDURE: inputdata
DESCRIPTION: get an output from the mobile channel
{*****}
  procedure inputdata;
  var nu1,nu2:integer;
  begin
    mobilechannel(noise);
  end;
{*****}
PROCEDURE: checkcb
DESCRIPTION: Check if detected symbols are right
VARIABLES n:accumulated bit-error #
           a,b:one is the true data and the other detected
{*****}
  procedure checkcb(var n:longint; a,b:complex);
  begin
    if round(a.re)<>round(b.re) then n:=n+1;
    if round(a.im)<>round(b.im) then n:=n+1;
  end;
{*****}
PROCEDURE: setnoise
DESCRIPTION: set noise power
{*****}
  procedure setnoise;
  begin
    noi[1]:=0.25;
    noi[2]:=0.1;
    noi[3]:=0.05;
    noi[4]:=0.025;
    noi[5]:=0.01;
    noi[6]:=0.005;
    noi[7]:=0.005;
    la[1]:=0.85;
    la[2]:=0.9;

  end;
{*****}
PROCEDURE: reini
DESCRIPTION : Reinitialize the DFE and RLS algorithm, retrain DFE
{*****}
  procedure reini;
  var m1:integer;

```

```

begin
    initialka;
    for m1:=1 to 20 do
        inputchannel;
        for m1:=1 to 10 do
            inputdata;
            for m1:=1 to 20 do
                begin
                    lambda:=0.99;
                    inputdata;
                    training:=true;
                    CoKal(training);
                    mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
                end; writeln(mse/50);
            end;
        end;
    end;
{*****
PROCEDURE: main
DESCRIPTION: Obtain BER of a specific point using 10000 bits
              show temporal results on the screen
VARIABLES: lambda: weighting factor
            totalnum: total number of bits
            totalerror: total number of bit errors
*****}
procedure main;
var ii,m1:integer;
begin
    reset(filedatre);
    reset(filedatim);
    xu:=c0;
    d:=c0;
    xin:=c0;
    nnn:=0;
    totalnum:=0;
    totalerror:=0;
    mcini;
    nnn:=-1;
    repeat
        nnn:=nnn+1;
        distance:=nnn/18+0.0001;
        area:=nnn;
        reini;
        mse:=0;

        for ii:=1 to 50 do
            begin
                mse:=0;
                numoferr:=0;
                for m1:=1 to 10 do
                    begin
                        lambda:=0.99;
                        inputdata;
                        training:=true;
                        CoKal(training);
                        checkcb(numoferr,td[delay+1],d);
                        mse:=mse+alpha.re*alpha.re+alpha.im*alpha.im;
                    end;
                totalnum:=totalnum+20;
                totalerror:=totalerror+numoferr;
                mse:=mse/10;
                writeln(noise:8:5,' ',totalnum,mse/2:14,' # of error: ',numoferr,'/20',totale
            end;
        ( reini;
        until nnn=17;
        writeln(filedat,noise,' ',totalerror/totalnum);
        result[delayray]:=totalerror/totalnum;

```

```

end;

{*****}
PROCEDURE: setsize
DESCRIPTION: Set size of DFE, weighting factor, delay
VARIABLES   n1: DFE forward #
             n2: DFE backward #
             lambda: weighting factor 0..1
             dt: delay
{*****}
procedure setsize;
begin
  n1:=2;delay:=n1-1;
  n2:=2;
  tN:=n1+n2;
  lambda:=0.99;
  delta:=1e6;
  dt:=3;
end;
{*****}
main program
DESCRIPTION: Function: control the simulations
VARIABLES   : symbolrate: symbol rate
             noise: noise power;
             filedatre,filedatim: input SIRCIM channels
             cubic spline data files
             filedat: BER output data file
{*****}
begin
  for i:=1 to 10 do result[i]:=100;
  raisedcosine;
  speed:=10;
  name:='s1';
  symbolrate:=1000000;
  assign(filedatre,'c:\htemp\' + name + 're.dat');
  assign(filedatim,'c:\htemp\' + name + 'im.dat');
  reset(filedatre);
  reset(filedatim);
  assign(filedat,'c:\htemp\difout\' + name + '.try');
  rewrite(filedat);
  setsize;
  inisamcha;
  noise:=0;
  setnoise;
  for i:=1 to 6 do
  begin
    noise:=noi[i];
    main;
  end;
  close(filedatre);
  close(filedatim);
  close(filedat);
end.(end of main)

```

```

(*****
FILENAME: DSIRCHA.PAS
CONTAIN UNITS: COMPTU, RAISED, SMACHA
CONTAIN: mobilechannel, mcini,
DESCRIPTION: Simulates pi/4 DQPSK transmitter and using SIRCIM
            or SMRCIM as the mobile channel.
*****)
Unit dsircha;
interface
    uses comptu,raised,samcha;
    var anp,inp:array[1..200] of complex;
        anpre,xin,apre:complex;
        dase:array [1..10] of complex;
        cr,xl:array [1..200] of complex;
        amp,pwr:array[1..2] of real;
        nnn,delayray,dt,cl:integer;
        Ac,pwrratioDB:real;
        Modulation:string[10];
        file1,file2:text;
    procedure mobilechannel(noise:real);
    procedure mcini;
implementation
(*****
PROCEDURE: putraised
DESCRIPTION: convolves the raised cosine filter with the mobile
            channel
INPUT PARMAS: c[]: channel impulse response
OUTPUT PARMAS: cr[]: after convolving wiht raised-cosine filter
*****)
    procedure putraised;
    var i,n,k:integer;
        q:real;
        ctemp:complex;
    begin
        for i:=1 to 69 do
            begin
                cr[i]:=c0;
                n:=0;
                repeat
                    n:=n+1;
                    rmul(ctemp,c[n],raisedfilter[i+1-n]);
                    cadd(cr[i],cr[i],ctemp);
                (
                    cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n]; )
                until (n=cl) or (i+1-n=1);
            end;
            k:=0;
            for i:=70 to 68+cl do
                begin
                    cr[i]:=c0;
                    k:=k+1;n:=k;
                    repeat
                        n:=n+1;
                        rmul(ctemp,c[n],raisedfilter[i+1-n]);
                        cadd(cr[i],cr[i],ctemp);
                    (
                        cr[i]:=cr[i]+c[n]*raisedfilter[i+1-n];)
                    until n=cl;
                end;
            end;
(*****
PROCEDURE: mobilechannel
DESCRIPTION: Simulates differential coder, convolves channel with
            the random data, adds noise
INPUT PARMAS: noise: noise power
OUTPUT PARMAS: xin: data sent

```

```

        inp[1]: output of the channel
*****}
procedure mobilechannel(noise:real);
const p=2;
var i,n:integer;
    tnoi,rt,q:real;
    a,temp:complex;
function gauss:real;
var temp:real;
    i:integer;
begin
    temp:=0;
    for i:=1 to 12 do
        temp:=temp+random;
    gauss:=temp-6;
end;

begin
    inputchannel;
    cl:=totoftbin+1;
    putraised;
    for i:=0 to 8 do
        dase[10-i]:=dase[9-i];

        if random>0.5 then dase[1].re:=1
        else dase[1].re:=-1;
        if random>0.5 then dase[1].im:=1
        else dase[1].im:=-1;

        cmul(a,dase[1],apre);
        q:=sqrt(2)/2;
        rmul(a,a,q);

        for i:=12 downto 1 do
            begin
                for n:=0 to 198 do
                    xl[200-n]:=xl[199-n];
                if i=12 then
                    xl[1]:=a
                else xl[1]:=c0;
                anp[i]:=c0;

                for n:=1 to 68+cl do
                    begin
                        cmul(temp,cr[n],xl[n]);
                        cadd(anp[i],anp[i],temp);
                    end;
            end;
            xin:=dase[dt];
            apre:=a;

            inp[1]:=anp[2];
            tnoi:=sqrt(noise);
            temp.re:=gauss*(tnoi);
            temp.im:=gauss*(tnoi);

            cadd(inp[1],inp[1],temp);

        end;(of complex channel)
*****}
PROCEDURE: mcini
DESCRIPTION: Initialize the transmitter and the mobile channel
*****}
procedure mcini;

```

```

begin
  for nnn:=1 to 200 do
    begin
      anp[nnn].re:=1;
      anp[nnn].im:=1;
      inp[nnn].re:=1;
      inp[nnn].im:=1;
      xl[nnn].re:=0;
      xl[nnn].im:=0;
      cr[nnn].re:=0;
      cr[nnn].im:=0;
    end;
    apre.re:=sqrt(2);
    apre.im:=0;
    anpre.re:=1;
    anpre.im:=1;
  end;
end.

```

Vita

Weifeng Huang was born in Shanghai, China, on November 27, 1964. He received the B.S. degree in Biomedical Engineering from Shanghai Jiao-Tong University in 1987. From 1987 to 1989, he was a design engineer in Shanghai Medical Electronic Instrument Factory, where he worked on the design of Electrocardiographs. In 1989, he joined the Mobile and Portable Radio Research Group at Virginia Tech, where he focused on the simulation of adaptive equalization for mobile radio, and the simulation of wideband outdoor channel impulse responses.

Mr. Huang is a member of the IEEE.