A Technique for Combining Equalization with Differential Detection

Kenneth Mark Aleong, B.Eng.
Department of Electrical Engineering
McGill University, Montreal

June, 1991

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of Master of Engineering

Abstract

A technique for combining equalization and differentially coherent detection is proposed for use in wireless communication when carrier phase recovery is difficult. A decision-feedback differentially coherent scheme, which generates an improved reference phase, is combined with a linear equalizer and the LMS algorithm is used to adapt the equalizer to an unknown channel. In addition, the proposed receiver is simulated for various two-dimensional signal constellations over multipath channels. It is shown that for high SNR, the degradation of this structure is negligible with respect to combined coherent detection and equalization. Therefore, this equalized differentially coherent detection scheme can be used when carrier phase tracking (i.e. coherent detection) is difficult and intersymbol interference is a major obstacle.

Résumé

Cette thèse propose une technique combinant l'égalisation et la détection cohérente différentielle pour la radiocommunication quand le rétablissement de la phase du signal porteur est difficile. Un système cohérent différentiel à rétroaction améliorant la phase de référence est combiné à un égalisateur linéaire. La procédure "CMM" est ensuite utilisée pour adapter l'égalisateur à un canal inconnu. De plus, une simulation du récepteur est faite avec des constellations de signaux à deux-dimensions pour des canaux multi-routes. Il est démontré que, pour un grand RSB, la dégradation de la performance de cette technique est négligeable par rapport à la combination classique de la détection cohérente et de l'égalisation. Donc, cette technique de détection cohérente différentielle égalisée peut-être utilisée quand la poursuite de la phase du signal porteur (c.a.d. la détection cohérente) est difficile et que l'interférence entre symboles est une probleme majeur.

Acknowledgement

I would sincerely like to thank Dr. Harry Leib for his numerous suggestions, helpful advice, encouragement, understanding and perseverance in the realization of this work.

I would also like to thank Dr. Peter Kabal for his invaluable guidance, understanding and financial support without which this work would not be done.

I wish to extend my gratitude to my loving parents and family for their continuous support and encouragement throughout my studies.

I would like to thank all my friends for their support. Special thanks to Aloknath De for his comments and to Marcel Sankeralli and Ronnie Quesnel for translating the abstract into French.

I wish to thank the Information and Network Systems Laboratory especially Sam Torrente and Ronnie Quesnel for their help with the computer simulations.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Meaning
$e^a, \exp[a]$	exponent of a
f	Frequency in [Hz]
ω	Angular Frequency in [Radians]
t	Time in [Secs]
T	Symbol of signaling interval
$[n]$	n-th symbol interval
$b[n]$	Amplitude or magnitude of actual or transmitted data symbol
$\varphi[n]$	Phase of actual data symbol
$\phi[n]$	Differentially encoded phase of transmitted data symbol
M	Number of signal points in constellation
$a_r[n]$	Real component of actual data symbol
$a_i[n]$	Imaginary component of actual data symbol
$\tilde{g}_T(t)$	Transmitter filter impulse response
$\tilde{g}_C(t)$	Complex channel impulse response
N_p	Number of paths in channel
$\rho[i]$	Amplitude attenuation in path i
$\theta[i]$	Phase Shift in path i
$\tau[i]$	Time Delay in path i
$\tilde{g}_R(t)$	Receiver filter impulse response

Symbol	**Meaning**
$\tilde{g}(t)$	Overall impulse response
$\tilde{G}_{T}(\omega)$	Transmitter filter transfer function
$\tilde{G}_{C}(\omega)$	Channel transfer function
$\tilde{G}_{F}(\omega)$	Folded channel transfer function
$\tilde{G}_{R}(\omega)$	Receiver filter transfer function
$\tilde{G}(\omega)$	Overall transfer function
$\tilde{n}(t)$	Additive white Gaussian noise
$\tilde{s}(t)$	Transmitted signal
$\tilde{r}(t)$	Receiver input signal
$\tilde{n}_{R}(t)$	Receiver filter output noise
$\tilde{y}(t)$	Receiver filter output
$y[n]$	Receiver filter sample (or equalizer input) at time nT
$\underline{y}[n]$	Equalizer input vector at time nT
$n_{R}[n]$	Sampled receiver filter noise
$c_{k}[n]$	k-th tap-coefficient or tap-gain of the linear equalizer
$\underline{c}[n]$	Linear equalizer coefficient vector
$2N+1$	Number of equalizer coefficients
$z[n]$	Equalizer output at time nT
$z'[n]$	Aligned or modified equalizer output
$v[n]$	Reference estimate
$\hat{\beta}[n]$	Reference phase estimate
L	Number of equalizer outputs used to generate $\hat{\beta}[n]$
$\epsilon[n]$	Error in data decision
$\varepsilon[n]$	Equalization error
A	Auto-correlation matrix

Symbol	**Meaning**
$\underline{B}$	Column vector with errorless reference phase estimation
$\underline{\hat{B}}$	Column vector with reference phase estimation errors
ξ	MSE, Mean Square Error
ξ_{min}	MMSE, minimum MSE
$\underline{c}_{opt}$	Optimum equalizer coefficient vector
γ	Particular value of L
μ_{γ}	Average gain in MMSE (L=1) dB, using L=γ
ν	Argument of Tikonov probability density
$\eta[n]$	Reference phase estimation error
$p_{\eta[n]}(\nu)$	Tikonov probability density
d_{min}	Minimum Euclidean distance between any two signal points
E_b	Energy per bit
σ	$\left(d^2_{min}\frac{\log_2 M}{\xi}\right)$ in dB
α	Roll-off factor for raised-cosine impulse response
λ	Step-size
λ_{opt}	Optimum step-size for fastest convergence
λ_1	First eigenvalue of the matrix A
λ_{2N+1}	($2N$+1)-th eigenvalue of the matrix A
$\lambda_{max}(A)$	Maximum eigenvalue of the matrix A
$Q_{\gamma}(C_i)$	Ratio of MMSE for L=1 to MMSE for L=γ, for channel C_i
N_c	Number of channels used to calculate μ
Λ, SNR	Signal to Noise Ratio
P_e	Probability of error
i, j, k, l	Indices

Chapter 1

Introduction

Recent years have witnessed an increased interest in bandwidth efficient modulation schemes. The simplest and most widely used technique for achieving high bandwidth efficiency is based on two-dimensional modulation formats [1]. With these schemes, demodulation is usually performed coherently, which means that carrier phase tracking is necessary. In many situations (such as communication over fading multipath channels, or short burst communications such as TDMA or Frequency Hopping), carrier phase tracking is a difficult task, and thus noncoherent demodulation techniques have to be used. The noncoherent demodulation methods for two-dimensional formats are based on differentially coherent techniques, and thus the phase information has to be differentially encoded. In these schemes, carrier phase tracking is not necessary; however, this is achieved at the expense of SNR performance.

In the last year, new differentially coherent detection techniques have been introduced [2]-[5]. The chief merit of these detection schemes is their low SNR degradation with respect to corresponding coherent detectors. One of the potential applications of the new differentially coherent strategies is for Indoor Wireless and Mobile Communications. In these systems, intersymbol interference due to multipath is a major problem. Therefore, the extent to which the new differentially coherent detection techniques can be suitable for these applications depends on the performance of these schemes in an intersymbol interference environment, and the possibility of combining them with equalization. This subject has not been considered yet (as far as we know), and this work makes a first step in this direction.

Two-dimensional modulation, where the data is encoded into the phase and amplitude of a sinusoidal carrier has been extensively studied in [1], [6]–[11]. In this work, Phase Shift Keying (PSK), Quadrature Amplitude Modulation (QAM) and V29 signal constellations [12], [13, page 243] will be used in a combined amplitude and differential phase modulation scheme, which uses amplitudes and phase differences to convey information. This modulation scheme is used instead of combined amplitude and phase modulation because the differential phase encoding enables the use of differentially coherent detection. Differentially coherent detection simplifies the receiver structure significantly since no phase tracking is performed and thus, is very attractive when carrier phase tracking is difficult. However, it has an SNR performance degradation compared to coherent detection that approaches 3 dB for MPSK (M>2). As a result, we propose to use the decision-feedback differentially coherent detection structure of [2] because of its low SNR degradation and relatively low complexity. Our objective is to consider this scheme over ISI channels, while focusing on the multipath environment. The decision-feedback differentially coherent detector of [2] can be naturally combined with known equalization techniques, while the other proposed differentially coherent detectors [3]–[5], seem to require special equalization methods.

In this work, we consider linear equalization, because of its reduced complexity. In addition, the Mean-Square-Error (MSE) criterion is used to find the optimum linear equalizer for known channels. However, in practice, the multipath characteristics of these channels are usually not known so that adaptive equalization is necessary. Therefore, we also consider the Least-Mean-Squares (LMS) adaptation algorithm [14], mainly because of its simplicity and robustness and also because it is one of the more popular algorithms used in practice.

This thesis is organized along the following lines. Chapter 2 presents the rationale of combining linear equalization with decision-feedback differentially coherent detection, and introduces the system model. In Chapter 3, the minimum MSE (MMSE) and optimum equalizer coefficients are derived for known channels, taking into account reference phase errors, and numerical results are presented for some multipath channels. In Chapter 4, the LMS adaptive algorithm is used for adapting the equalizer to an unknown channel and Adaptive Mean-Square-Error (AMSE) simulation results are presented. Finally, Chapter 5 states the conclusions and suggests further work. This is followed by a bibliography of related articles and two appendices. Appendix A presents an overview of the overall computer program and lists the MMSE program file and a sample test case. Appendix B lists the AMSE program file, a sample test case and additional program files.

Chapter 2

Combining Equalization and Decision-Feedback Differentially Coherent Detection

The subject of this chapter is the integration of linear equalization with differentially coherent detection. Section 2.1 discusses the need for differentially coherent detection and linear equalization in a communication system. Section 2.2 describes the baseband system model, including the proposed receiver which combines an improved differentially coherent detection structure with a linear equalizer. Finally, Section 2.3 focuses on the advantages of this proposed receiver over conventional coherent receivers which combine coherent detection and linear equalization.

2.1 Equalization and Decision-Feedback Differential Coherent Detection

Any communication system consists of three components: transmitter, channel and receiver. The main objective in any communication system is to transmit information as accurately as possible. The transmitter encodes the discrete-time information into a continuous-time signal which is transmitted over the channel. The receiver must recover the information from the received signal which is a distorted version of the transmitted signal. This distortion is due to the channel. Channel distortion can be generated by noise, fading, as well as time-dispersion. Therefore, the transmitter and receiver have to be designed with the communications channel in mind.

An important parameter of a communication system is the method by which the information is encoded into the transmitted signal, the modulation method. Much attention has been given to two-dimensional modulation, where the data is encoded into the phase and amplitude of a sinusoidal carrier [6]–[8], mainly because of its bandwidth efficiency. A close relative to this amplitude and phase modulation is amplitude and differential phase modulation.

Differential phase modulation structures the sinusoidal carrier such that carrier phase differences and not actual carrier phases convey information [15]. Thus, carrier phase tracking, which tracks absolute phases, is not necessary at the receiver since phase differences between successive signals (and not the absolute phases of the signals) convey information. The phase encoding adds little to the complexity of the transmitter. In this work, combined amplitude and differential phase modulation, with differentially coherent detection, is considered.

A differentially coherent detector estimates the transmitted information by making use of phase differences between successive symbols. In the absence of channel distortion, differentially coherent detection is an attractive alternative to coherent detection especially when carrier phase recovery is difficult. It has been successfully applied with PSK modulation, particularly for binary PSK (BPSK) signal [16, page 174]. This gives an extremely simple receiver for BPSK with a small degradation in performance. However, for MPSK (M>2), it gives an SNR degradation that approaches 3 dB as M increases. In [2], an improved differentially coherent detection technique was introduced. The proposed differential receiver structure uses past phase decisions to modify L previous received samples. These modified samples were then summed to give an improved phase reference. This strategy can be considered as an open loop version of a coherent receiver with decision-feedback carrier phase tracking. It was found that the performance of this improved differentially coherent detection approaches that of coherent detection for high SNR.

As stated earlier, the channel distorts the transmitted signal. In a time-dispersive channel, the effect of each transmitted symbol extends beyond the time-interval used to represent that symbol. This is due to the dispersion effect of the channel which broadens pulses and causes them to interfere with one another. The distortion caused by the resulting overlap of received signals is called intersymbol interference (ISI). Its effect is most easily described in an equivalent baseband pulse amplitude modulation (PAM) system. Such a system is shown in Figure 2.1.

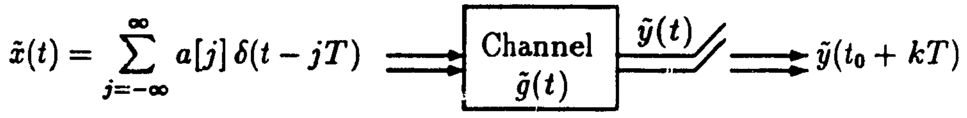
$\tilde{x}(t) = \sum_{j=-\infty}^{\infty} a[j]\,\delta(t - jT)$
Channel
$\tilde{g}(t)$
$\tilde{y}(t)$
$\tilde{y}(t_0 + kT)$

Figure 2.1: A Baseband PAM model

In Figure 2.1, $\delta(t)$ is the Dirac delta function and the "channel" includes the effect of the transmitter filters, the transmission medium and the receiver filters. The channel's impulse response is $\tilde{g}(t)$ and the input signal $\tilde{x}(t)$ is a sequence of data symbols $a[j]$ which are transmitted at instants jT through the channel where T is the signaling (or symbol) interval and $\tilde{}$ is used to represent the complex envelope (CE) notation. Therefore, the CE of the received signal $\tilde{y}(t)$ is given by

$$\tilde{y}(t) = \sum_{j=-\infty}^{\infty} a[j]\,\tilde{g}(t - jT) \tag{2.1}$$

If the received signal is sampled at instant $kT + t_0$, where t_0 accounts for the channel delay and the sampler phase, we get

$$\tilde{y}(t_0 + kT) = \underbrace{a[k]\,\tilde{g}(t_0)}_{desired\ term} + \underbrace{\sum_{j=-\infty,\, j\neq k}^{\infty} a[j]\,\tilde{g}(t_0 + kT - jT)}_{ISI} \tag{2.2}$$

The ISI is induced by $\tilde{g}(t_0 + iT)$, $i \neq 0$. The ISI is zero if $\tilde{g}(t_0 + iT)$=0, $\imath \neq 0$; that is, if $\tilde{g}(t)$ has zero crossings at T-spaced intervals. When $\tilde{g}(t)$ has such uniformly spaced zero crossings, it is said to satisfy Nyquist's criterion [13, page 157]. The criterion specifies a frequency-domain condition on the received pulses for zero ISI. It can be expressed as:

$$\tilde{G}_F(f) = \sum_{k=-\infty}^{\infty} \tilde{G}(f - \frac{k}{T}) = T \qquad for \quad |f| \leq \frac{1}{2T} \tag{2.3}$$

where $\tilde{G}(f)$ is the channel frequency response (i.e. the Fourier transform of $\tilde{g}(t)$), $\tilde{G}_F(f)$ is the folded channel spectral response after symbol-r^te sampling and the frequency band $|f| \leq \frac{1}{2T}$ is the Nyquist or minimum bandwidth.

One class of pulse shapes which are ISI-free and commonly used, is the raised-cosine family with cosine roll-off around $|f| = \frac{1}{2T}$. It can be expressed as

$$\tilde{g}(t) = \frac{\sin(\pi\frac{t}{T})}{(\pi\frac{t}{T})} \times \frac{\cos(\frac{\alpha\pi t}{T})}{[1 - (\frac{\alpha\pi t}{T})^2]} \tag{2.4}$$

where α is the roll-off factor with a value between 0 and 1. From [13, page 158], the transfer function $\tilde{G}(\omega)$ of $\tilde{g}(t)$ ($\omega = 2\pi f$) is given by

$$\tilde{G}(\omega) = \begin{cases} T & 0 \leq |\omega| \leq \frac{(1-\alpha)\pi}{T} \\ \frac{T}{2}\left(1 - \sin\left[\frac{T}{2\alpha}(|\omega| - \frac{\pi}{T})\right]\right) & \frac{(1-\alpha)\pi}{T} \leq |\omega| \leq \frac{(1+\alpha)\pi}{T} \\ 0 & |\omega| > \frac{(1+\alpha)\pi}{T} \end{cases} \tag{2.5}$$

$\tilde{G}(\omega)$ and $\tilde{g}(t)$ for $\alpha = 0, 0.3, 0.6, 1.0$ are shown in Figures 2.2 and 2.3. It is easily seen that these frequency responses $\tilde{G}(\omega)$ satisfy Nyquist's criterion, and thus there is no ISI. In practice, the effect of ISI can be seen from a trace of the received signal on an oscilloscope with its time base synchronized to the symbol clock. For a two-level PAM system, if the channel satisfies the zero ISI condition, there are only two distinct levels at the sampling instant.

Although the transmitter and receiver are designed so that Nyquist's criterion is satisfied, in practice, the channel distorts the signals so that actually the criterion is not satisfied and ISI results. As a result, equalizers, which are designed to deal with ISI, are used [17]. The objective of an equalizer is to reduce the effects of ISI on the process of data recovery from the received signal.

Equalizers which use delays and tap-gain multipliers, and operate in the time-domain are known as discrete-time filters. In these, current and past received signals (and maybe past receiver decisions) are weighted by different tap-gains, and used to reduce the ISI at a particular time instant. There are two categories of discrete-time equalizers, namely linear transversal equalizers and decision-feedback equalizers (DFEs). In linear transversal equalizers, current and past values of the received signal are linearly weighted by the equalizer taps and summed to produce an output. These equalizers are usually implemented with a finite number of taps for physical reasons, i.e. as a finite impulse response (FIR) filter. As a result, they cannot remove all ISI. In addition, a linear equalizer introduces gains at those frequencies where the folded channel has loss and this gain amplifies noise at those frequencies. Thus, the noise power at the equalizer output is larger than if the linear equalizer was not present, i.e. noise is enhanced by the linear equalizer. Nevertheless, linear equalizers are used in practice since they are good approximations to the ideal filter for a sufficient number of FIR filter taps and can be used in an adaptive mode. DFEs are recursive nonlinear equalizers that make use of past receiver decisions and are comprised of a forward

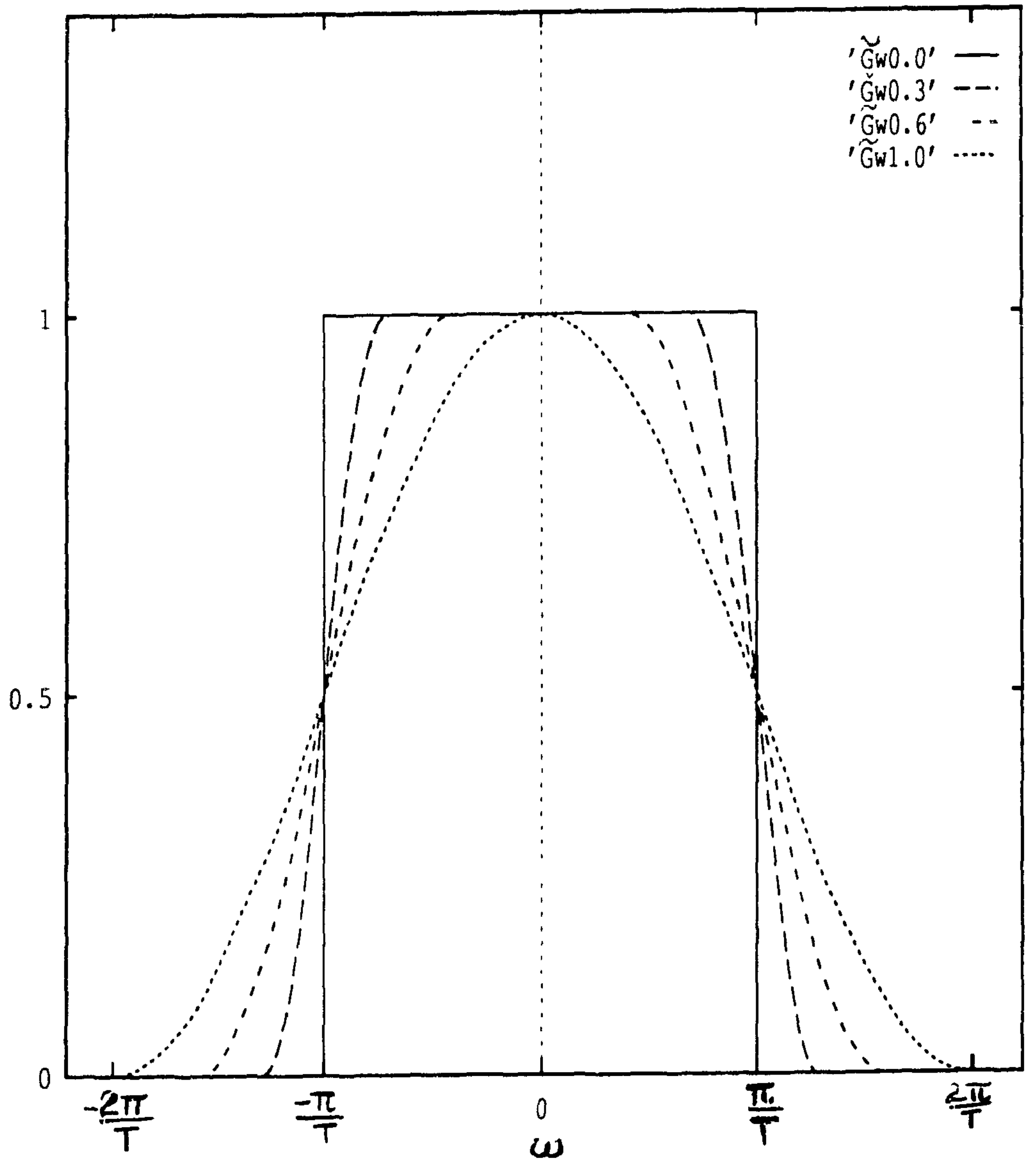

Figure 2.2: $\tilde{G}(\omega)$ which satisfy Nyquist criterion

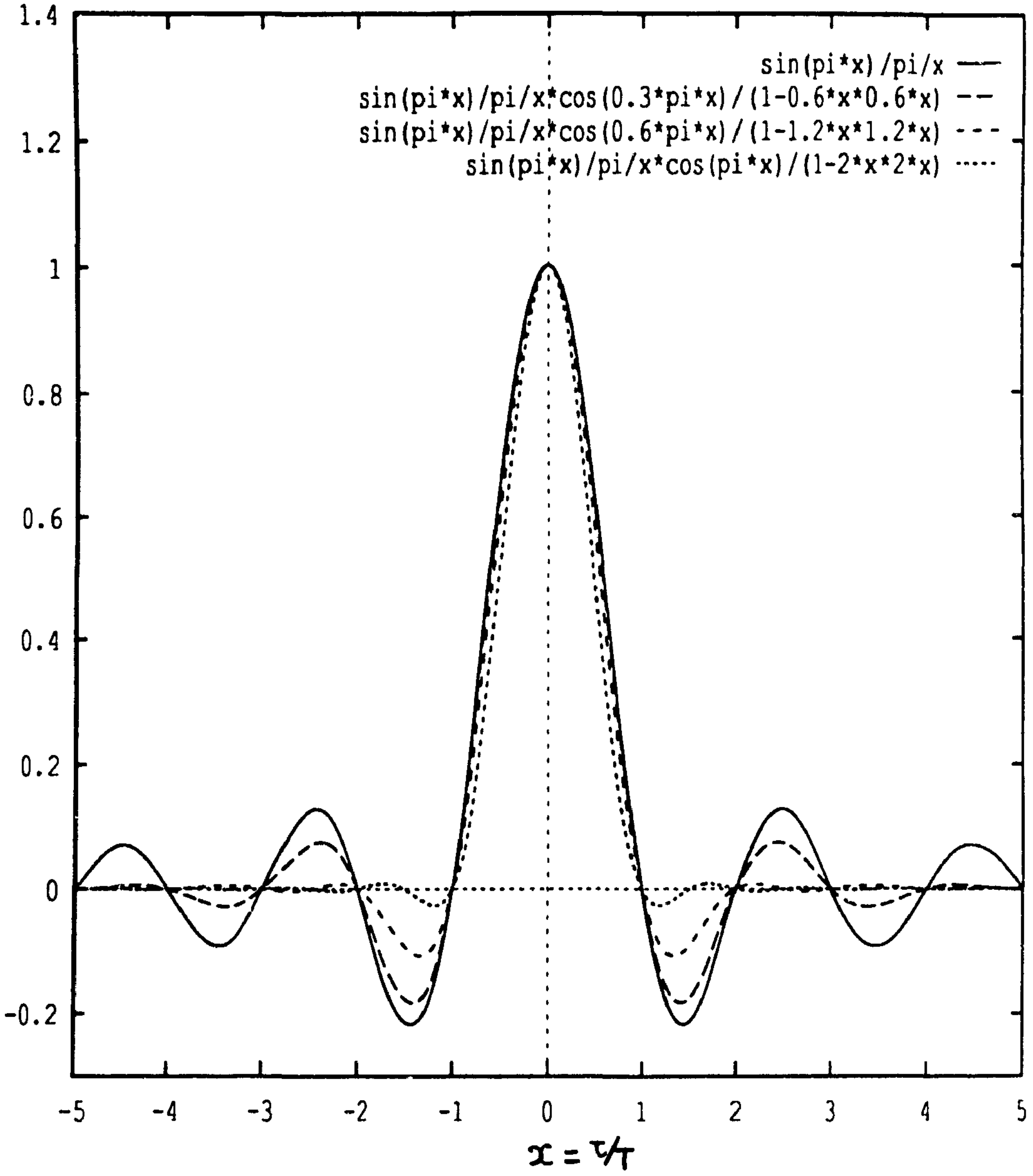

Figure 2.3: $\tilde{g}(t)$ which satisfy Nyquist criterion

and feedback filter. The forward filter is similar to a linear transversal filter. Its function is to eliminate precursor ISI (samples of the pulse response before the main lobe) while the function of the feedback filter is to cancel the postcursor ISI (samples of the pulse response after the main lobe), see Figure 2.3. In addition, DFEs do not enhance noise as much as linear equalizers and are less sensitive to sampling phase errors. However, DFEs suffer from feedback error propagation. Therefore, they are more difficult to use in adaptive mode due to this lack of guaranteed stability.

This work considers linear equalization for systems that employ differential detection. This subject has been given consideration in the literature [15], [18]–[20]. A linear equalizer following a differential detector as in [18], has the difficult task of equalizing a nonlinear channel due to the quadratic nature of the channel dependent terms at the differential detector output. As a result, a linear equalizer cannot effectively equalize the channel, and non-linear equalization techniques should be considered. Therefore, a linear equalizer should precede the differential detector as in [15], since it has to equalize a linear channel. In [19], a scheme for adaptive equalization of incoherently demodulated signals was presented. In the scheme, a linear equalizer, placed after an envelope detector, was used to make an estimate of the ISI due to multipath fading and acted as an ISI canceller (i.s.i.c). In addition, differential phase estimation and phase tracking estimation were both used in the receiver structure. Also, the equalizer structure had complex tap-gains and real input values, instead of the usual complex tap gains and complex input values, which reduced the system complexity by fifty percent. However, in this scheme, the linear equalizer has the difficult task of coping with the nonlinearity introduced by the envelope detector. Adaptive equalization for differential coherent reception in the presence of channel distortion was also studied in [20]. A linear equalizer, with seven taps, was placed before a differential detector and differential data encoding was performed by multiplying the previously transmitted data symbol by the current data symbol. Simulations were done at high SNR for BPSK and QPSK. Similar rates of convergence were shown for a coherent receiver and the differential detection receiver. However, the MSE obtained for the differential case was about 3 dB larger than that obtained in the coherent case. We intend to solve this problem by using the improved differentially coherent detection technique of [2].

In [2], an improved differentially coherent detection receiver was introduced for an ISI-free additive white Gaussian noise channel. The main advantage of this differentially coherent detection technique is its negligible degradation with respect to coherent detection. With ISI, there is need for an equalizer as well. By placing a linear equalizer before differentially coherent detection, the effects of the ISI can be reduced and detection is performed on an almost ISI-free signal. Furthermore, equalization is performed without the need for carrier phase tracking, improving the robustness of the system to carrier phase noise, and carrier phase hits.

2.2 Baseband System Model

The baseband model (complex envelope) for the system considered in this work is shown in Figure 2.4. In this work, continuous-time signals use ( ) brackets and discrete-time signals [ ] rectangular brackets. Figure 2.4 will now be briefly described: The system is composed of three conceptual parts: transmitter, channel and receiver

2.2.1 Transmitter

The transmitter model consists of a differential phase encoder followed by a transmitter filter $\tilde{g}_T(t)$. Let us consider two dimensional modulated data signals specified by the complex envelope (CE) notation. The CE of the transmitted signal is given by

$$\tilde{s}(t) = \sum_{k=-\infty}^{\infty} b[k]e^{j\phi[k]}\tilde{g}_T(t-kT) \qquad (2.6)$$

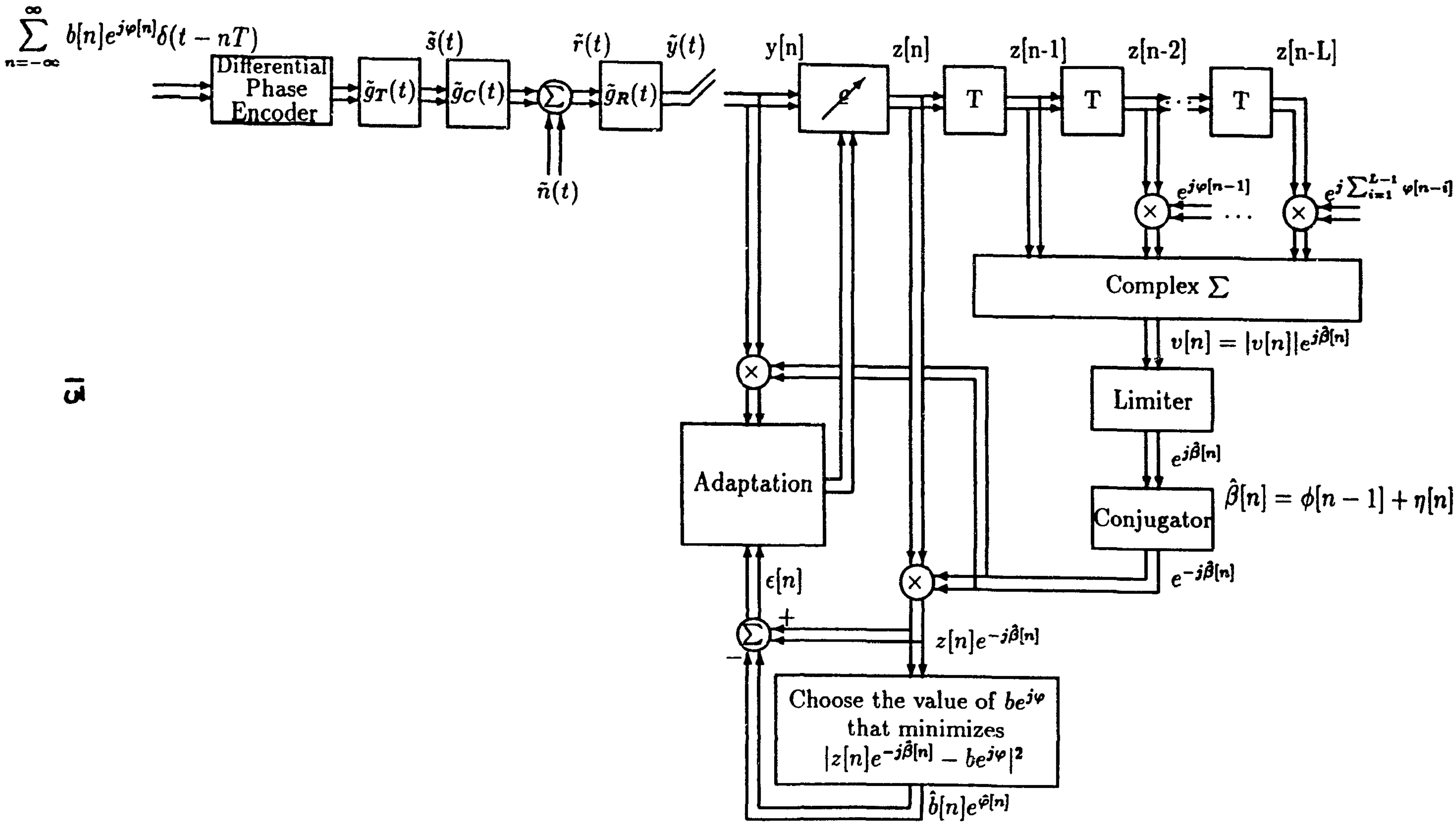
$\sum_{n=-\infty}^{\infty} b[n]e^{j\varphi[n]}\delta(t-nT)$
Differential Phase Encoder
$\tilde{g}_T(t)$
$\tilde{s}(t)$
$\tilde{g}_C(t)$
Σ
$\tilde{n}(t)$
$\tilde{r}(t)$
$\tilde{g}_R(t)$
$\tilde{y}(t)$
y[n]
c
z[n]
T
z[n-1]
T
z[n-2]
T
z[n-L]
$e^{j\varphi[n-1]}$
$e^{j\sum_{i=1}^{L-1}\varphi[n-i]}$
...
Complex Σ
$v[n] = |v[n]|e^{j\beta[n]}$
Limiter
$e^{j\hat{\beta}[n]}$
Conjugator
$\hat{\beta}[n] = \phi[n-1] + \eta[n]$
$e^{-j\hat{\beta}[n]}$
Adaptation
$\epsilon[n]$
+
−
$z[n]e^{-j\hat{\beta}[n]}$
Choose the value of $be^{j\varphi}$ that minimizes $|z[n]e^{-j\hat{\beta}[n]} - be^{j\varphi}|^2$
$\hat{b}[n]e^{\hat{\varphi}[n]}$

Figure 2.4: Baseband System Model

where $b[k]e^{j\phi[k]}$ are the amplitude and differentially phase-encoded data transmitted at time instant kT and T is the duration of a symbol interval.

Amplitude and Differential Phase Modulation

Symmetric signal constellations e.g. PSK, QAM, V29, are commonly used for two-dimensional modulation. In this work, the symmetric constellations shown in Figure 2.5 are used and each constellation point is specified by an amplitude b and phase φ. In our scheme, the transmitted phase data is differentially encoded so that phase *differences* and not absolute phase values convey information. The encoded phase ϕ[n] is given by

$$\phi[n] = \phi[n-1] \oplus \varphi[n] \tag{2.7}$$

where $\oplus$ means phase addition modulo 2π. Therefore, the transmitted *amplitude and differential phase encoded* information symbols are $b[n]e^{j\phi[n]}$ where $b[n]e^{j\varphi[n]}$ ($= a[n] = a_r[n] + ja_i[n]$) are the actual data symbols and $a_r[n]$ and $a_i[n]$ are the real and imaginary components of the actual data respectively.

The average power $E[b^2[n]]$ of each constellation is normalized to unity. Therefore, all points in a MPSK constellation will have unit amplitude with each point k having a phase of $\frac{2\pi k}{M}$ where $k = 1, \ldots, M$. In a 4PSK system, $b[n] = 1$ and $\phi[n]$ assumes values from the set of $(0, \pm\frac{\pi}{2}, \pi)$. In addition, the minimum Euclidean distance d_{min} for this constellation is $\sqrt{2}$. For 8PSK, $\phi[n]$ assumes values from $(0, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3\pi}{4}, \pi)$ and the minimum distance is 0.7654.

For the 16QAM system, $a_r[n]$ and $a_i[n]$ are first chosen independently from the set $[\pm 1, \pm 3]$. The average signal power is normalized to one and the signal points are rescaled accordingly. Therefore, $b[n]$ assumes values from ($\frac{1}{\sqrt{5}}$, 1, $\frac{3}{\sqrt{5}}$) and $\varphi[n]$ (and not $\phi[n]$) from the set of (0, $\pm 0.1\pi$, $\pm 0.25\pi$, $\pm 0.4\pi$, $\pm 0.6\pi$, $\pm 0.75\pi$, $\pm 0.9\pi$, π) depending on which signal point is transmitted. In addition, the minimum distance between any two signal points is equal to 0.6325.

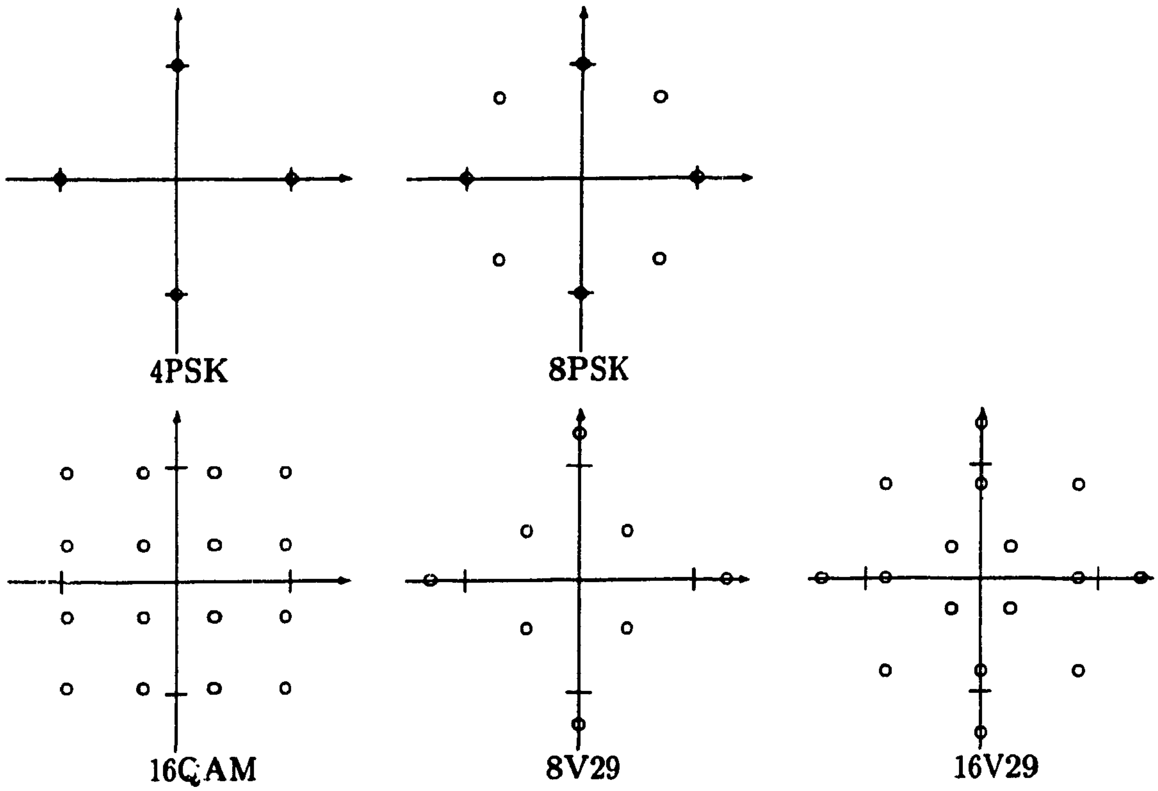

Figure 2.5: Symmetric Two-Dimensional Signal Constellations

The 8V29 constellation consists of two sets of QPSK signals on different circles where the outer circle has a radius $\frac{3}{\sqrt{2}}$ times that of the inner radius. Also, the two QPSK constellations are out of phase by $\frac{\pi}{4}$. Thus, $b[n]$ assumes values from the set of $(\frac{2}{\sqrt{11}}, \frac{3\sqrt{2}}{\sqrt{11}})$ and $\phi[n]$ from $(0, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3\pi}{4}, \pi)$. In addition, its minimum distance is equal to 0.8528.

The 16V29 constellation consists of four sets of QPSK signals on different circles where the second circle has a radius $\frac{3}{\sqrt{2}}$ times that of the inner radius, the third circle has a radius $\sqrt{2}$ times that the second and the fourth is $\frac{5}{3\sqrt{2}}$ times that of the third. Also, QPSK constellations on odd circles are out of phase with respect to QPSK constellations on the even circles by $\frac{\pi}{4}$. Thus, $b[n]$ assumes values from the set of $(\frac{2}{3\sqrt{3}}, \frac{\sqrt{2}}{\sqrt{3}}, \frac{2}{\sqrt{3}}, \frac{5\sqrt{2}}{3\sqrt{3}})$ and $\phi[n]$ from $(0, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3\pi}{4}, \pi)$. Finally, its minimum distance is equal to 0.5443.

Transmitter Filter

The transmitter filter is a pulse shaping filter with a real impulse response $\tilde{g}_T(t)$. The desired overall impulse response $\tilde{g}(t)$ ($= \tilde{g}_T(t) * \tilde{g}_C(t) * \tilde{g}_R(t)$ where $*$ denotes convolution.) is a Nyquist raised-cosine response with roll-off factor α, assuming $\tilde{g}_C(t) = \delta(t)$. Also, the transfer function of the desired Nyquist raised-ccsine response is divided equally between the transmitter and the receiver filters. Thus, the transmitter filter is designed so that its transfer function $\tilde{G}_T(\omega)$ is equal to $\sqrt{\tilde{G}(\omega)}$ where $\tilde{G}(\omega)$ is the transfer function of the desired Nyquist response $\tilde{g}(t)$. In our model, the roll-off factor α is set to zero so that the raised-cosine Nyquist response has zero excess-bandwidth. Therefore, the transmitter's impulse response $\tilde{g}_T(t)$ can be expressed as:

$$\tilde{g}_T(t) = \frac{sin(\pi \frac{t}{T})}{(\pi \frac{t}{T})} \tag{2.8}$$

and the transfer function $\tilde{G}_T(\omega)$ is given by

$$\tilde{G}_T(\omega) = \begin{cases} T & |f| \leq \frac{1}{2T} \\ 0 & |f| > \frac{1}{2T} \end{cases} \tag{2.9}$$

2.2.2 Channel

The channel response is represented by the complex impulse response $\tilde{g}_C(t)$ and additive white Gaussian noise $\tilde{n}(t)$. A multipath channel model is used. Thus, the complex impulse response $\tilde{g}_C(t)$ can be expressed as

$$\tilde{g}_C(t) = \sum_{i=1}^{N_p} \rho[i] e^{j\theta[i]} \delta(t - \tau[i]) \tag{2.10}$$

where N_p is the number of paths in the channel, $\rho[i]$ is the amplitude attenuation in path i, $\theta[i]$ is the phase-shift in path i and $\tau[i]$ is the relative signal delay due to path *i*. Consequently, the receiver input, $\tilde{r}(t)$ can be expressed as

$$\tilde{r}(t) = \tilde{s}(t) * \tilde{g}_C(t) + \tilde{n}(t) \tag{2.11}$$

where $\tilde{s}(t)$ is the transmitted signal, $\tilde{g}_C(t)$ is the channel impulse response and $\tilde{n}(t)$ is additive white Gaussian noise with zero-mean and N_0 [Watt/Hz] power spectral density of the real and imaginary component.

2.2.3 Receiver

The baseband equivalent receiver consists of a filter with impulse response $\tilde{g}_R(t)$ followed by a sampler. The sampler is followed by a linear equalizer and then by the decision-feedback differential coherent detection structure of [2].

Receiver Filter

As previously stated, the transmitter and receiver filters are designed so that the overall response in an ideal channel is a Nyquist raised-cosine response. In addition, the desired Nyquist transfer function is divided equally between the two filters, which gives an optimal receiver structure for an ISI-free channel. Thus, the receiver filter has transfer function $\tilde{G}_R(\omega)$ which is given by

$$\tilde{G}_R(\omega) = \tilde{G}_T(\omega) = \sqrt{\tilde{G}(\omega)} \tag{2.12}$$

where $\tilde{G}_T(\omega)$ is the transfer function of the transmitter filter impulse response, which is given in (2.9) and $\tilde{G}(\omega)$ is the transfer function of the desired overall response. Using (2.11), the receiver filter output $\tilde{y}(t)$ is given by

$$\tilde{y}(t) = \{\tilde{s}(t) * \tilde{g}_C(t) + \tilde{n}(t)\} * \tilde{g}_R(t) \tag{2.13}$$

where $\tilde{s}(t)$ is the transmitted signal, $\tilde{g}_C(\mathrm{t})$ is the channel impulse response, $\tilde{n}(\mathrm{t})$ is the channel additive white Gaussian noise and $\tilde{g}_R(t)$ is the receiver filter impulse response.

Thus, the receiver filter output can also be expressed as

$$\tilde{y}(t) = \sum_{k=-\infty}^{\infty} b[k]e^{j\phi[k]}\tilde{g}(t - kT) + \tilde{n}_R(t) \tag{2.14}$$

where

$$\tilde{g}(t) = \tilde{g}_T(t) * \tilde{g}_C(t) * \tilde{g}_R(t)$$

and

$$\tilde{n}_R(t) = \int_{-\infty}^{\infty} \tilde{n}(t - \tau)\tilde{g}_R(\tau)d\tau$$

Therefore, the noise $\tilde{n}_R(t)$ has zero-mean and power spectrum density

$$P(\omega) = 2N_0|\tilde{G}_R(\omega)|^2 \tag{2.15}$$

where $\tilde{G}_R(\omega)$ denotes the Fourier transform of $\tilde{g}_R(t)$. Sampling the received signal $\tilde{y}(t)$ at $t = nT$, the discrete-time output $y[n]$ can be expressed as:

$$y[n] = \sum_{k=-\infty}^{\infty} b[k]e^{j\phi[k]}g[n - k] + n_R[n] \tag{2.16}$$

where $g[n - k] = \tilde{g}([n - k]T)$, $n_R[n] = \tilde{n}_R(nT)$ and $b[k]e^{j\phi[k]}$ are the amplitude and differentially phase-encoded data transmitted at time instant kT.

Linear Equalizer

The linear equalizer has $2N$+1 complex taps and equalizes both in-phase and quadrature components using its real and imaginary taps. The input to the linear equalizer is given in (2.16). The adaptive digital equalizer has complex coefficients $c_k[n]$: $k = -N, \ldots, 0, \ldots, N$ where $c_0[n]$ is the reference tap and [n] corresponds to a particular symbol interval or iteration. Thus, the equalizer output $z[n]$ is given by:

$$z[n] = \sum_{k=-N}^{N} c_k[n]y[n - k] \tag{2.17}$$

There are many criteria for obtaining the optimum linear equalizer coefficients for a known channel. The peak distortion criterion would have been sufficient if only the

ISI is to be minimized [21]. However, the noise must be taken into account. Therefore, the Mean Square Error(MSE) criterion is used.

For an unknown or time-varying channel, the equalizer must adapt itself. The speed and stability of convergence are important factors which must be considered in choosing an adaptive algorithm. In fact, many different adaptive algorithms exist and a survey on adaptive equalization can be found in [22]. One adaptive algorithm is the Least-Mean-Squares (LMS) gradient algorithm, which was proposed in [14] and has been extensively used in the last few decades. In this work, the LMS algorithm is employed because of its simplicity and robustness and is the subject of Chapter 4. Finally, there has been recent work on faster-converging algorithms [23]–[25], and these algorithms are briefly discussed in Chapter 4.

Decision-Feedback Differentially Coherent Detection

We use an improved differentially coherent detection structure, introduced in [2] which can reduce the SNR degradation with respect to coherent detection. The principles on which this detection strategy rely on will now be discussed.

One way of interpreting a differentially encoded scheme is in terms of phase references. Differential phase encoding preprocesses the signal such that the required phase reference for estimating the information is carried by the previous symbol. Therefore, in differentially coherent detection, there is no need to establish an absolute phase reference, since the previous symbol phase is used for that. This simplifies the receiver structure when compared to coherent detection which requires carrier phase tracking. However, this is achieved at the expense of a loss of about 3 dB in performance relative to coherent MPSK($M>$ 2). This is because in a differentially coherent (DC) scheme, the phase reference is impaired by channel noise in the same way as the information phase. Therefore, in a DC scheme, detection is performed with a noisy phase reference, and when compared to ideal coherent detection, where the phase reference is noise-free, it gives a degradation in performance. Quantitatively, in a DC scheme, the SNR of the reference signal is the same as the SNR of the information signal. In a coherent scheme, the SNR of the reference signal is infinite (ideal coherent case) and the SNR of the information signal is finite. Thus, the DC detection technique can be generalized so that the reference signal is extracted from a number of past symbols which results in smoothing the channel noise. Using this method, the SNR of the reference signal is increased and the performance should approach that of a coherent scheme. This is the approach used in [2].

The differentially coherent detection structure generates a reference phase by summing the aligned past L equalizer outputs $z[n-L], \ldots, z[n-1]$. Each of the previous L equalizer outputs, except the most previous one, i.e. $z[n-1]$, has its phase incremented by the sum of the phase decisions φ's of the signals between it and $z[n-1]$. Therefore, the aligned equalizer outputs $z'[n-i]$ $i = 2, \ldots, L$ are given by

$$z'[n-i] = z[n-i]\ \exp\left[j\sum_{k=1}^{i-1}\varphi[n-k]\right] \tag{2.18}$$

Summing the $z'[n-i]$, $i = 1, \ldots, L$ where $z'[n-1] = z[n-1]$ gives

$$v[n] = |v[n]|e^{j\hat{\beta}[n]} = \sum_{i=1}^{L} z'[n-i] = \sum_{i=1}^{L} z[n-i]\ \exp\left[j\sum_{k=1}^{i-1}\varphi[n-k]\right] \tag{2.19}$$

The result of this coherent summation of the equalizer outputs, $v[n]$, has a larger SNR due to the smoothing of the noise and as a result, its phase $\hat{\beta}[n]$ is a better estimate of the exact phase reference $\phi[n-1]$. The reference phase estimate $\hat{\beta}[n]$ is then subtracted from the phase of the equalizer output $z[n]$. Thus, the decision variable presented to the threshold detector is $z[n]e^{-j\hat{\beta}[n]}$. The threshold detector generates an output decision symbol $\hat{b}e^{j\hat{\varphi}}$ which minimizes the squared error $|z[n]e^{-j\hat{\beta}[n]} - be^{j\varphi}|^2$. The error $\epsilon[n]$ is then used to adapt the equalizer coefficients.

The reference phase estimation process derived above was analyzed for an additive white Gaussian noise channel in [2] for MPSK. In the alignment of the vectors, actual information phases $\varphi[n-k]\}_{k=1}^{L-1}$ are used. In practice, the receiver operates in a decision-feedback mode (i.e. φ's used in the alignment process would be the $\hat{\varphi}$ decisions on previous phases). To simplify the analysis, the feedback decisions are assumed error-free. The effect of errors in the feedback decisions would be to reduce momentarily the SNR of the reference signal which obviously depends on L. For small L, a decision-feedback error is more noticeable. However, the persistence time of this effect is only L symbols and is thus short. For L=1, this is just the double error effect in DC receivers. For large L, a decision-feedback error is not very noticeable since the SNR reduction in the reference signal is small. However, the effect lasts for L symbols.

2.3 Comparison with Equalization and Coherent Detection

The advantages of our "differential" receiver, which combines an improved differentially coherent detection scheme and linear equalization, over conventional "coherent" receivers, which combine coherent detection and equalization, will now be discussed.

The first advantage of the differential receiver is that it can be used in fading multipath channels where carrier phase tracking is difficult. This is because the proposed differential receiver avoids carrier phase tracking with little performance degradation. If a coherent receiver were employed, carrier phase tracking would be quite complicated since carrier phase recovery is very difficult in these channels and since there is coupling between the phase estimation and equalization which affects the system performance. Therefore, the improved differentially coherent detection scheme is very attractive for fading multipath channels.

The second advantage of the differential receiver is that it can be used in burst communication. In burst communication, data is usually transmitted in short bursts, i.e. over a very short time period. As a result, coherent receivers cannot be used since there is not enough data for carrier phase tracking. The proposed differential receiver is ideal for this situation since it does not track absolute carrier phases and can adapt very quickly to bursts of data.

The third advantage is that the differential receiver employs baseband equalization. Baseband equalization is preferred for many technologiral reasons and can be used to compensate for asymmetrical baseband impairments [26]. However, for coherent receivers, it introduces a delay in decision-oriented carrier phase estimation loops, which causes inaccurate ᵈᵉtection. As a result, passband equalization (which is more difficult to implement digitally) is usually employed since it allows coherent receivers to deal with carrier phase tracking more easily. For the differential receiver, no carrier ρhase tracking is necessary and therefore baseband equalization (which can be implemented more easily in a digital fashion) can always be used without any of the disadvantages associated with coherent receivers.

Finally, the proposed differential receiver avoids phase ambiguities due to symmetric signal constellations since it assumes that phase differences (and not absolute phases as coherent receivers with decision-directed phase tracking assume) convey information.

Chapter 3

Equalization for Known Channels

This chapter analyses the equalized decision-feedback differentially coherent detection technique of Chapter 2, using the MSE criterion for channels whose characteristics are known beforehand. Section 3.1 derives the MMSE and optimum equalizer coefficients in terms of the auto-correlation matrix A and the cross-correlation column vector $\hat{\underline{\beta}}$. Section 3.2 expresses these two quantities in terms of the channel characteristics, assuming perfect reference phase estimation. Section 3.3 analyzes reference phase estimation errors and their effects on MMSE calculations. Section 3.4 presents numerical results. Finally, Section 3.5 concludes the chapter by discussing the MMSE numerical results.

3.1 MMSE Analysis

In this section, the MSE criterion is used to derive the optimum equalizer coefficients and the minimum MSE (MMSE) for known channels. All quantities involved in the analysis are shown in Figure 2.4.

The actual data symbols $b[n]e^{j\varphi[n]}$ are assumed to be statistically independent and equiprobable. In addition, the average signal power of each constellation is normalized to one. Thus,

$$E\left[b^2[n]\right] = 1 \tag{3.1}$$

The optimum equalizer coefficients will now be derived using the MSE criterion. The equalizer coefficients are optimum if they minimize the MSE :

$$E|\epsilon[n]|^2$$

where $\epsilon[n]$ is the error between the differentially detected equalized output and the desired data symbol. It can be expressed as

$$\epsilon[n] = z[n]e^{-j\hat{\beta}[n]} - \underbrace{b[n]e^{j\varphi[n]}}_{\textit{desired signal}} \tag{3.2}$$

where $\hat{\beta}[n]$ is the reference phase estimate, i.e. phase estimate of $\phi[n-1]$. Thus,

$$E|\epsilon[n]|^2 = E\left|z[n]e^{-j\hat{\beta}[n]} - b[n]e^{j\varphi[n]}\right|^2 \tag{3.3}$$

Now if $\underline{c}[n] = [c_{-N}[n], \ldots, c_0[n], \ldots, c_N[n]\,]^T$ represents the $(2N+1)$ equalizer coefficients at the n-th symbol interval and $\underline{y}^T[n] = [y[n-N], \ldots, y[0], \ldots, y[n+N]\,]$, then (2.17) becomes

$$z[n] = \underline{c}^T[n]\,\underline{y}[n] \tag{3.4}$$

Substituting (3.4) into (3.3), we get

$$E|\epsilon[n]|^2 = E\left|\underline{c}^T[n]\underline{y}[n]e^{-j\hat{\beta}[n]} - b[n]e^{j\varphi[n]}\right|^2 \tag{3.5}$$

After some manipulation,

$$E|\epsilon[n]|^2 = \underline{c}^{*T}[n]A\underline{c}[n] - 2Re\left[\underline{c}^{*T}[n]\hat{\underline{B}}\right] + E\left[b^2[n]\right] \tag{3.6}$$

where

$$A = E\left[\underline{y}^*[n]\,\underline{y}^T[n]\,\right] \tag{3.7}$$

and

$$\hat{\underline{B}} = E\left[b[n]e^{j\varphi[n]}\underline{y}^*[n]e^{j\hat{\beta}[n]}\right] \tag{3.8}$$

Thus, it is easily seen that A is the auto-correlation matrix of $y[n]$ and $\underline{\hat{B}}$ is the cross-correlation matrix between the received data $y[n]$ (phase-shifted by $\hat{\beta}[n]$) and the transmitted data symbols $b[n]e^{j\phi[n]}$. The MSE can be minimized by differentiating with respect to $\underline{c}[n]$ and equating to zero. Therefore

$$\frac{\partial E|\epsilon[n]|^2}{\partial \underline{c}[n]} = 2A\underline{c}[n] - 2\underline{\hat{B}} = 0 \tag{3.9}$$

and the optimum solution is

$$\underline{c}_{opt}[n] = A^{-1}\underline{\hat{B}} \tag{3.10}$$

Now, using (3.1) and (3.10) in (3.6), the MMSE ξ_{min} can be expressed as

$$\xi_{min} = 1 - \underline{\hat{B}}^{T^*} A^{-1}\underline{\hat{B}} \tag{3.11}$$

3.2 MMSE with Perfect Reference Phase

From the previous section, it is seen that $\underline{\hat{B}}$ depends on the reference phase estimate $\hat{\beta}[n]$ which is related to the exact reference phase $\phi[n-1]$ via:

$$\hat{\beta}[n] = \phi[n-1] + \eta[n] \tag{3.12}$$

where $\eta[n]$ is the error of this estimator. The random variable $\eta[n]$ depends on an ensemble of samples $z[k]$, $k \leq n-1$, and thus, it is almost uncorrelated with any single sample $y[n-k]$, $-N \leq k \leq N$. Using this assumption and substituting (3.12) in (3.8), $\underline{\hat{B}}$ can be expressed as:

$$\underline{\hat{B}} = E\left[b[n]e^{j\varphi[n]}\underline{y}^*[n]e^{j\phi(n-1)}\right]E\left[e^{j\eta[n]}\right] \tag{3.13}$$

$$\underline{\hat{B}} = \underbrace{E\left[b[n]e^{j\phi[n]}\underline{y}^*[n]\right]}_{\underline{B}}E\left[e^{j\eta[n]}\right] \tag{3.14}$$

$$\underline{\hat{B}} = \underline{B}\,.\,E\left[e^{j\eta[n]}\right] \tag{3.15}$$

where $\underline{B}$ is the cross-correlation vector with errorless reference phase estimation. For the moment, perfect reference phase estimation will be assumed (i.e. η[n]=0 and $\underline{\hat{B}}$

$= \underline{B}$). The matrix A and the column vector $\underline{B}$ will now be simplified in terms of the overall impulse response and the SNR. From (3.7),

$$A = E\left[\underline{y}^*[n]\underline{y}^T[n]\right]$$

where

$$\underline{y}^T[n] = [y[n-N], \ldots, y[n], \ldots, y[n+N]]$$

and $y[n]$ is defined in (2.16). Thus, for $i, j = -N, \ldots, 0, \ldots, N$,

$$\begin{aligned}
A_{ij} &= E\left[y^*[n+i]y[n+j]\,\right] \\
&= E\left[\left\{\sum_{k=-\infty}^{\infty} b[n+i-k]e^{-j\phi[n+i-k]}g^*[k] + n_R^*[n+i]\right\}\right. \\
&\qquad \left.\times\left\{\sum_{l=-\infty}^{\infty} b[n+j-l]e^{j\phi[n+j-l]}g[l] + n_R[n+j]\right\}\right] \\
&= E\left[\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} b[n+i-k]b[n+j-l]e^{-j\phi[n+i-k]}e^{j\phi[n+j-l]}g^*[k]\,g[l]\right] \\
&\qquad +2\Re\left\{E\left[\sum_{k=-\infty}^{\infty} b[n+i-k]e^{-j\phi[n+i-k]}\,g^*[k]\,n_R[n+j]\right]\right\} \\
&\qquad\qquad +E\,\left[n_R^*[n+i]\,n_R[n+j]\,\right] \\
&= (1) + (2) + (3) \\
(1) &= \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} g^*[k]g[l]\,\underbrace{E\left[b[n+i-k]b[n+j-l]e^{j\{\phi[n+j-l]-\phi[n+i-k]\}}\right]}_{(4)}
\end{aligned}$$

For $i-k \neq j-l$,

$$\begin{aligned}
(4) &= E\left[b[n+i-k]\,b[n+j-l]\,e^{j\{\varphi[n+j-l]+\ldots+\varphi[n+i-k+1]\,\}}\,\right] \\
&= E\left[b[n+i-k]\,b[n+j-l]\,\right]\,E\left[e^{j\{\varphi[n+j-l]+\ldots+\varphi[n+i-k+1]\,\}}\,\right] \\
&= E\left[b[n+i-k]\,b[n+j-l]\,\right]\,\underbrace{E\left[e^{j\varphi[n+j-l]}\right]}_{0}\cdots\,\underbrace{E\left[e^{j\varphi[n+i-k+1]}\right]}_{0} \\
&= 0
\end{aligned}$$

since the φ's are statistically independent and $E[e^{j\varphi}] = 0$ for symmetrical constellations. Now, for $i - k = j - l$,

$$\begin{aligned}(4) \;&=\; E\left[b^2[n+i-k]\right] \\ &=\; 1\end{aligned}$$

Therefore,

$$E\left[b[n+i-k]b[n+j-l]e^{j\{\phi[n+j-l]-\phi[n+i-k]\}}\right] = \delta_{i-k,j-l} \tag{3.16}$$

where $\delta_{ij} = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases}$. Thus, using (3.16),

$$\begin{aligned}(1) \;&=\; \sum_{k=-\infty}^{\infty} g^*[k] \sum_{l=-\infty}^{\infty} g[l]\delta_{i-k,j-l} \\ &=\; \sum_{k=-\infty}^{\infty} g^*[k] \sum_{l=-\infty}^{\infty} g[l]\delta_{l,k-i+j} \\ &=\; \sum_{k=-\infty}^{\infty} g^*[k]\, g[k-i+j] \\ (2) \;&=\; 2\Re\left\{ \sum_{k=-\infty}^{\infty} g^*[k]\; E\left[b[n+i-k]\, e^{-j\phi[n+i-k]}\; n_R[n+j]\right]\right\} \\ &=\; 2\Re\left\{ \sum_{k=-\infty}^{\infty} g^*[k]\; E\left[b[n+i-k]\, e^{-j\phi[n+i-k]}\right]\; \underbrace{E\left[n_R[n+j]\right]}_{0}\right\} \\ &=\; 0\end{aligned}$$

where the noise n_R and data $be^{j\varphi}$ are assumed uncorrelated and $\tilde{n}_R(t)$ is a zero-mean process.

$$\begin{aligned}(3) \;&=\; E\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{n}^*(\tau)\, \tilde{g}_R^*([n+i]T-\tau)\, \tilde{n}(\tau')\, \tilde{g}_R([n+j]T-\tau')\, d\tau d\tau'\right] \\ &=\; \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \underbrace{E\left[\tilde{n}^*(\tau)\, \tilde{n}(\tau')\right]}_{2N_0\delta(\tau-\tau')}\, \tilde{g}_R^*([n+i]T-\tau)\, \tilde{g}_R([n+j]T-\tau')\, d\tau d\tau' \\ &=\; 2N_0 \int_{-\infty}^{\infty} \tilde{g}_R^*([n+i]T-\tau)\, \tilde{g}_R([n+j]T-\tau)\, d\tau \\ &=\; 2N_0\delta_{ij}\end{aligned}$$

since $\tilde{n}(t)$ is white and $\tilde{g}_R(t) * \tilde{g}_R(t)$ satisfies Nyquist's first criterion. Therefore, for $i, j = -N, \ldots, 0, \ldots, N$.

$$A_{ij} = \sum_{k=-\infty}^{\infty} g^*[k]\, g[k - i + j] + 2N_0\delta_{ij} \tag{3.17}$$

The matrix A is Hermitian and positive semi-definite. Now, from (3.14),

$$\underline{B} = E\left[b[n]e^{j\phi[n]}\underline{y}^*[n]\right]$$

where

$$\begin{aligned}
\underline{B}^T &= [B[-N], \ldots, B[0], \ldots, B[N]\,] \\
\underline{y}^T[n] &= [y[n-N], \ldots, y[n], \ldots, y[n+N]\,]
\end{aligned}$$

Thus, for $i = -N, \ldots, 0, \ldots, N$,

$$\begin{aligned}
B[i] &= E\left[b[n]e^{j\phi[n]}y^*[n+i]\right] \\
&= E\left[b[n]e^{j\phi[n]}\left\{\sum_{k=-\infty}^{\infty}\left[b[n+i-k]e^{-j\phi[n+i-k]}g^*[k]\right] + n_R^*[n+i]\right\}\right] \\
&= E\left[\sum_{k=-\infty}^{\infty} b[n]\, b[n+i-k]\, e^{j\{\phi[n]-\phi[n+i-k]\}}g[k]\right] + E\left[b[n]e^{j\phi[n]}\right]\ \underbrace{E\left[n_R^*[n+i]\,\right]}_{0}
\end{aligned}$$

The summation and expectation operators can be interchanged since they are linear. Therefore,

$$\begin{aligned}
B[i] &= \sum_{k=-\infty}^{\infty} g[k]\ \underbrace{E\left[b[n]\, b[n+i-k]\, e^{j\{\phi[n]-\phi[n+i-k]\}}\right]}_{\delta_{0,i-k}} \\
&= \sum_{k=-\infty}^{\infty} g[k]\, \delta_{ik} \\
B[i] &= g[i]
\end{aligned} \tag{3.18}$$

using (3.16). Therefore, the errorless column vector $\underline{B}$ is simply a truncated overall impulse response vector, i.e. $\underline{B} = [g[-N], \ldots, g[0], \ldots, g[N]\,]$.

3.3 Reference Phase Error Analysis

In the previous section, perfect reference phase estimation was assumed. However, in practice, phase estimation errors will occur. In our analysis, perfect receiver decisions are assumed, and estimation errors are due mainly to channel noise and ISI.

From, Section 3.2, only $\underline{\hat{B}}$ depends on $\eta[n]$ via (3.15). Therefore, the dependence of $\underline{\hat{B}}$ (and the MMSE) on the reference phase error $\eta[n]$ defined in (3.12) will now be found by analyzing $E\left[e^{j\eta[n]}\right]$. From (2.19),

$$\begin{aligned}
|v[n]|\, e^{j\beta[n]} &= \sum_{i=1}^{L} z[n-i] \exp\left[j \sum_{k=1}^{i-1} \varphi[n-k]\right] \\
&= \sum_{i=1}^{L} z[n-i] \exp[j\{\phi[n-1] - \phi[n-i]\}] \qquad (3.19)
\end{aligned}$$

Also, the past equalizer outputs can be expressed by :

$$z[n-i] = b[n-i]\, e^{j\phi[n-i]} + \varepsilon[n-i] \qquad (3.20)$$

where $\varepsilon[n-i]$ is the equalization error. From [13], for high SNR and with $E\,[b^2[n]] = 1$,

$$\begin{aligned}
E\,[\varepsilon[n-i]\,] &= 0 \qquad (3.21) \\
E|\varepsilon[n-i]\,|^2 &= N_o T \int_{-\frac{1}{2T}}^{\frac{1}{2T}} \frac{df}{\frac{1}{T}\sum_{m=-\infty}^{\infty} \left|\tilde{G}_C(f-\frac{m}{T})\right|^2 + N_o} \qquad (3.22) \\
&\simeq N_o T \int_{-\frac{1}{2T}}^{\frac{1}{2T}} \frac{df}{\frac{1}{T}\sum_{m=-\infty}^{\infty} \left|\tilde{G}_C(f-\frac{m}{T})\right|^2} \qquad (3.23)
\end{aligned}$$

Substituting (3.20) into (3.19), we get

$$\begin{aligned}
|v[n]\,|e^{j\beta[n]} &= \left\{\sum_{i=1}^{L} b[n-i] + \sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}\right\} e^{j\phi[n-1]} \qquad (3.24) \\
&= |v[n]\,|\, e^{j\{\phi[n-1]+\eta[n]\}} \qquad (3.25)
\end{aligned}$$

where

$$|v[n]|e^{j\eta[n]} = \sum_{i=1}^{L} b[n-i] + \sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]} \qquad (3.26)$$

and $\eta[n]$ is the phase estimation error. For $\eta[n] \ll 1$ and $E\left[\eta[n]\right] = 0$,

$$E\left[e^{j\eta[n]}\right] \simeq 1 - \frac{1}{2}E\left[\eta^2[n]\right] \tag{3.27}$$

From (3.26), it is seen that $\eta[n]$ is the phase error of a (real) phasor $\sum_{i=1}^{L} b[n-i]$ perturbed by noise $\sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}$, and thus the results from [2] can be used. Thus, we fix $b[n-1], \ldots, b[n-L]$ and calculate the *conditional* variance of $\eta[n]$, i.e. $E\left[\eta^2[n] \mid b[n-1], \ldots, b[n-L]\right]$. For high SNR and fixed $b[n-i]$, $\imath = 1, \ldots, L$, the asymptotic distribution of $\eta[n]$ is Tikonov [2] and the conditional probability density function (pdf) $p_{(\eta[n] \mid b[n-1],\ldots,b[n-L])}(\nu)$ can be expressed as :

$$p_{(\eta[n] \mid b[n-1],\ldots,b[n-L])}(\nu) \simeq \frac{\exp[\,\Lambda[b[n-1],\ldots,b[n-L]\,]cos(\nu)]}{2\pi I_0\left[\Lambda\left[b[n-1],\ldots,b[n-L]\right]\right]} \tag{3.28}$$

where I_0 is the modified Bessel function of order zero and Λ is the SNR of the (real) phasor $\sum_{i=1}^{L} b[n-i]$ perturbed by the noise $\sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}$ which can be expressed as :

$$\Lambda\left[b[n-1],\ldots,b[n-L]\right] = \frac{\left\{\sum_{i=1}^{L} b[n-i]\right\}^2}{E\left[\left|\sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}\right|^2 \,\Big|\, b[n-1],\ldots,b[n-L]\right]} \tag{3.29}$$

where the numerator is the power of $\sum_{i=1}^{L} b[n-i]$ and the denominator is the variance of $\sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}$ for fixed $b[n-i]$, $i = 1, \ldots, L$. Now, the denominator can be expressed as

$$E\left[\left|\sum_{i=1}^{L} \varepsilon[n-i]e^{-j\phi[n-i]}\right|^2 \,\Big|\, b[n-1],\ldots,b[n-L]\right]$$

$$= E\left[\sum_{i=1}^{L}\sum_{k=1}^{L} \varepsilon[n-i]\varepsilon^*[n-k]e^{-j\{\phi[n-i]-\phi[n-k]\}} \,\Big|\, b[n-1],\ldots,b[n-L]\right]$$

$$
\begin{aligned}
&= \sum_{i=1}^{L}\sum_{k=1}^{L} E\left[\varepsilon[n-i]\varepsilon^{*}[n-k] \mid b[n-1],\ldots,b[n-L]\right] \times \\
&\qquad\qquad \underbrace{E\left[e^{-j\{\phi[n-i]-\phi[n-k]\}} \;\middle|\; b[n-1],\ldots,b[n-L]\right]}_{=\delta_{ik}\ \text{since}\ E[e^{j\varphi}\,|\,b]=0\ \text{for symmetrical constellations.}} \\
&= \sum_{i=1}^{L} E\left[|\varepsilon[n-i]\,|^{2} \;\middle|\; b[n-1],\ldots,b[n-L]\right] \qquad (3.30)
\end{aligned}
$$

where we used the fact that the equalization error $\varepsilon[n-i]$ is practically uncorrelated with $e^{j\phi[n-i]}$. Therefore, substituting (3.30) into (3.29), we get

$$
\Lambda\left[b[n-1],\ldots,b[n-L]\right] = \frac{\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2}}{\sum_{i=1}^{L} E\left[|\varepsilon[n-i]\,|^{2} \;\middle|\; b[n-1],\ldots,b[n-L]\right]} \qquad (3.31)
$$

and

$$
\begin{aligned}
E\left[\eta^{2}[n] \;\middle|\; b[n-1],.,b[n-L]\right] &\simeq \frac{1}{\Lambda\left[b[n-1],\ldots,b[n-L]\right]} &(3.32)\\
&= \frac{\sum_{i=1}^{L} E\left[|\varepsilon[n-i]\,|^{2} \;\middle|\; b[n-1],.,b[n-L]\right]}{\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2}} &(3.33)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
E\left[\eta^{2}[n]\right] &= E\left[\frac{\sum_{i=1}^{L} E\left[|\varepsilon[n-i]\,|^{2} \;\middle|\; b[n-1],\ldots,b[n-L]\right]}{\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2}}\right] &(3.34)\\
&\simeq \sum_{i=1}^{L} E|\varepsilon[n-i]\,|^{2} . E\left[\frac{1}{\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2}}\right] &(3.35)
\end{aligned}
$$

Here we assumed that $E\left[|\varepsilon[n-i]\,|^{2} \mid b[n-1],\ldots,b[n-L]\right]$ is uncorrelated with $\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2}$. For large L, then $\left\{\sum_{i=1}^{L} b[n-i]\right\}^{2} \simeq L^{2}\,\{E[b]\}^{2} \simeq K$, where K

is a constant and thus, it is clear that the assumption is valid. For small L, then $E[\,|\varepsilon[n-i]\,|^2 \mid b[n-1],\ldots,b[n-L]\,] \simeq E\,[|\varepsilon[n-i]\,|^2]$ since only a small fraction of signal samples which are stored in the equalizer are fixed, and thus the equalization error is almost the same as the one obtained when no signal sample is constrained. Thus, the assumption is valid again. Therefore, with (3.23) the variance of the phase estimation error is given by

$$E\left[\eta^2[n]\right] \simeq LN_oT\int_{-\frac{1}{2T}}^{\frac{1}{2T}} \frac{df}{\frac{1}{T}\sum_{m=-\infty}^{\infty}\left|\tilde{G}_C(f-\frac{m}{T})\right|^2}\cdot E\left[\frac{1}{\left\{\sum_{i=1}^{L} b[n-i]\right\}^2}\right] \tag{3.36}$$

Also, from (3.15) and (3.27),

$$\underline{\hat{B}} = \underline{B}\,.\,\left\{1 - E\left[\eta^2[n]\right]\right\} \tag{3.37}$$

which shows that

$$|\underline{\hat{B}}| \leq |\underline{B}| \tag{3.38}$$

The MMSE

$$\xi_{min} = 1 - \underline{\hat{B}}^{T^*} A^{-1}\underline{\hat{B}} \tag{3.39}$$

is larger than the one with $\underline{B}$ (perfect reference phase estimation). The optimum equalizer coefficients are

$$\underline{c}_{opt}[n] = A^{-1}\underline{\hat{B}} \tag{3.40}$$

The optimum equalizer coefficients for a known channel can be computed by finding A^{-1} first. However, there is another numerical way of finding the optimum equalizer coefficients without inverting the matrix A. This is done by using the MSE Gradient (MSEG) algorithm [13] :

$$\underline{c}[n+1] = \underline{c}[n] - \frac{\lambda}{2}\frac{\partial E|\epsilon[n]|^2}{\partial \underline{c}[n]} \tag{3.41}$$

$$\underline{c}[n+1] = \underline{c}[n] - \lambda\,[\underline{\hat{B}} - A\underline{c}[n]] \tag{3.42}$$

$$\underline{c}[n+1] = [I + \lambda A]\,\underline{c}[n] - \lambda\underline{\hat{B}} \tag{3.43}$$

It should be noted that in (3.43), $[n]$ denotes the number of iterations and not a particular time instant nT in the data symbol sequence. To ensure convergence, the step-size λ must satisfy

$$0 < \lambda < \frac{2}{\lambda_{max}(A)}$$

where $\lambda_{max}(A)$ is the maximum eigenvalue of the matrix A.

3.4 Numerical Results

The MMSE was calculated for various 2-D constellations, channels, SNRs, number of equalizer taps $(2N+1)$ and L (number of equalizer outputs used to generate the phase estimate of previous transmitted symbol) which are listed below.

- Five constellations: 4PSK, 8PSK, 16QAM, 8V29 and 16V29.
- Five channels: A, B, C, X, and Y.
- Three SNRs$(=\frac{E[b^2[n]]}{N_0}=\frac{1}{N_0})$: 8 dB, 15 dB, and 25 dB.
- Number of equalizer taps $(2N+1)$: $1, 3, \ldots, 21$.
- Values of L used: 1, 2, 3 and 5.

The five channels tested were multipath channels with impulse response given by (2.10). Multipath propagation, in these channels, can be viewed as signal transmission subjected to different paths with differing relative amplitude attenuation, phase-shifts and delays. In addition, if $\sum_{i=2}^{L} \rho[i] < 1$ and $\rho[1]=1$, $\theta[1]=0$, $\tau[1]=0$, in (2.10), the channel is minimum phase [24] and has mainly postcursor ISI. In our simulations, all the channels tested are minimum phase. The five channels and their impulse responses are listed below. Channels A, B and C each have two paths each, while X and Y have three and five paths respectively.

$$\mathrm{A} : \tilde{g}_C(t) = \delta(t) - 0.5\delta(t - 0.5T)$$

$$\mathrm{B} : \tilde{g}_C(t) = \delta(t) - 0.5\delta(t - 1.5T)$$

$$\mathrm{C} : \tilde{g}_C(t) = \delta(t) - 0.5\delta(t - 3.5T)$$

$$\mathrm{X} : \tilde{g}_C(t) = \delta(t) - 0.3\delta(t - 0.5T) + 0.5j\delta(t - 3.5T)$$

$$\begin{aligned}\mathrm{Y} : \tilde{g}_C(t) = \delta(t) - 0.3\delta(t - 0.7T) - 0.07\delta(t - 1.5T) + 0.07j\delta(t - 1.5T)\\ +0.1\delta(t - 1.8T) + 0.2j\delta(t - 3.5T)\end{aligned}$$

The matrix A and the column vector $\underline{\hat{B}}$ had complex values due to the complex impulse response of the multi-path channels. A zero roll-off factor was used. The element values of A and $\underline{\hat{B}}$ were calculated using the equations (3.17), (3.18), (3.36) and (3.37). The MMSE calculations were performed by matrix inversion for various N and L. For each constellation, the squared minimum distance d^2_{min} between any two points was compared with the MMSE results to get a better indication of the system performance.

Tables 3.1-5 list the MMSE results for each constellation with L=1, for various SNRs and number of equalizer taps (=2N+1). Table 3.6 lists the average gain μ in MMSE (in dB) that is achieved by increasing the value of L for nine equalizer taps. The average gain μ (for a particular SNR and constellation) was calculated as follows: Assume we want to calculate μ for L equal to γ, i.e. μ_γ. For each channel C_i, the MMSE result for L=1 was divided by the MMSE result for L=γ to give a MMSE ratio $Q_\gamma(C_i)$. The $Q_\gamma(C_i)$s for each channel C_i were then summed and the total was divided by the number of channels tested N_c, i.e. 5, to give an average Q_γ. To find μ in dB, the logarithm to the base 10 was taken and then multiplied by 10. Thus for a particular SNR, constellation and L=γ, we have

$$\mu_\gamma = 10 \log_{10} \left[\frac{1}{N_c} \sum_{i=1}^{N_c} Q_\gamma(C_i) \right] \qquad 3.41)$$

Finally, results for a sample MMSE test case are given in Appendix A.

	Channel A			Channel B			Channel C			Channel X			Channel Y		
SNR $2N+1$	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB
1	0.5552	0.3160	0.2498	0.3319	0.1987	0.1667	0.3651	0.2226	0.1886	0.4245	0.1948	0.1318	0.3947	0.2046	0.1552
3	0.4600	0.1354	0.0312	0.3135	0.1669	0.1294	0.3647	0.2215	0.1872	0.3925	0.1423	0.0714	0.3445	0.1222	0.0604
5	0.4570	0.1332	0.0284	0.2605	0.0877	0.0399	0.3627	0.2181	0.1832	0.3887	0.1375	0.0664	0.3421	0.1194	0.0575
7	0.4540	0.1285	0.0228	0.2603	0.0846	0.0345	0.3363	0.1760	0.1351	0.3841	0.1330	0.0616	0.3391	0.1167	0.0545
9	0.4525	0.1267	0.0211	0.2536	0.0716	0.0183	0.2937	0.1097	0.0592	0.3696	0.0989	0.0176	0.3250	0.0864	0.0162
11	0.4515	0.1254	0.0197	0.2531	0.0711	0.0175	0.2878	0.1011	0.0495	0.3692	0.0984	0.0170	0.3249	0.0856	0.0149
13	0.4508	0.1245	0.0187	0.2514	0.0680	0.0134	0.2853	0.0976	0.0456	0.3690	0.0980	0.0165	0.3247	0.0851	0.0141
15	0.4502	0.1238	0.0180	0.2509	0.0676	0.0130	0.2818	0.0851	0.0277	0.3689	0.0976	0.0156	0.3245	0.0844	0.0130
17	0.4498	0.1233	0.0175	0.2502	0.0664	0.0116	0.2805	0.0827	0.0249	0.3683	0.0957	0.0130	0.3240	0.0829	0.0109
19	0.4995	0.1228	0.0170	0.2498	0.0661	0.0113	0.2793	0.0805	0.0222	0.3681	0.0955	0.0128	0.3239	0.0827	0.0106
21	0.4492	0.1225	0.0167	0.2494	0.0655	0.0107	0.2773	0.0756	0.0155	0.3680	0.0953	0.0125	0.3238	0.0826	0.0105

Table 3.1: MMSE with L=1, for 4PSK, Squared Minimum Distance = 2.0

	Channel A			Channel B			Channel C			Channel X			Channel Y		
SNR / 2N+1	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB
1	0.5552	0.3160	0.2498	0.3319	0.1987	0.1667	0.3651	0.2226	0.1886	0.4245	0.1948	0.1318	0.3947	0.2046	0.1552
3	0.4600	0.1354	0.0312	0.3135	0.1669	0.1294	0.3647	0.2215	0.1872	0.3925	0.1423	0.0714	0.3445	0.1222	0.0604
5	0.4570	0.1332	0.0284	0.2605	0.0877	0.0399	0.3627	0.2181	0.1832	0.3887	0.1375	0.0664	0.3421	0.1194	0.0575
7	0.4540	0.1285	0.0228	0.2603	0.0846	0.0345	0.3363	0.1760	0.1351	0.3841	0.1330	0.0616	0.3391	0.1167	0.0545
9	0.4525	0.1267	0.0211	0.2536	0.0716	0.0183	0.2937	0.1097	0.0592	0.3696	0.0989	0.0176	0.3250	0.0864	0.0162
11	0.4515	0.1254	0.0197	0.2531	0.0711	0.0175	0.2878	0.1011	0.0495	0.3692	0.0984	0.0173	0.3249	0.0856	0.0149
13	0.4508	0.1245	0.0187	0.2514	0.0680	0.0134	0.2853	0.0976	0.0456	0.3690	0.0980	0.0165	0.3247	0.0851	0.0141
15	0.4502	0.1238	0.0180	0.2509	0.0676	0.0130	0.2818	0.0851	0.0277	0.3689	0.0976	0.0156	0.3245	0.0844	0.0130
17	0.4498	0.1233	0.0175	0.2502	0.0664	0.0116	0.2805	0.0827	0.0249	0.3683	0.0957	0.0130	0.3240	0.0829	0.0109
19	0.4995	0.1228	0.0170	0.2498	0.0661	0.0113	0.2793	0.0805	0.0222	0.3681	0.0955	0.0128	0.3239	0.0827	0.0106
21	0.4492	0.1225	0.0167	0.2494	0.0655	0.0107	0.2773	0.0756	0.0155	0.3680	0.0953	0.0125	0.3238	0.0826	0.0105

Table 3.2: MMSE with L=1, for 8PSK, Squared Minimum Distance = 0.5858

	Channel A			Channel B			Channel C			Channel X			Channel Y		
SNR / 2N+1	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB
1	0.6470	0.3216	0.2499	0.3671	0.2004	0.1667	0.4085	0.2247	0.1886	0.4931	0.1986	0.1318	0.4493	0.2074	0.1552
3	0.5715	0.1424	0.0313	0.3496	0.1686	0.1294	0.4081	0.2237	0.1872	0.4649	0.1463	0.0715	0.4493	0.1253	0.0605
5	0.5691	0.1403	0.0285	0.2994	0.0897	0.0399	0.4063	0.2202	0.1832	0.4616	0.1416	0.0665	0.4014	0.1226	0.0576
7	0.5667	0.1356	0.0229	0.2992	0.0865	0.0345	0.3817	0.1783	0.1351	0.4576	0.1371	0.0617	0.3987	0.1199	0.0546
9	0.5655	0.1339	0.0212	0.2928	0.0736	0.0183	0.3420	0.1121	0.0592	0.4448	0.1032	0.0176	0.3859	0.0896	0.0162
11	0.5647	0.1326	0.0198	0.2924	0.0731	0.0175	0.3365	0.1036	0.0496	0.4444	0.1027	0.0170	0.3858	0.0889	0.0149
13	0.5642	0.1317	0.0188	0.2908	0.0699	0.0134	0.3342	0.1001	0.0457	0.4442	0.1023	0.0165	0.3855	0.0883	0.0142
15	0.5637	0.1310	0.0181	0.2903	0.0695	0.0131	0.3310	0.0876	0.0277	0.4442	0.1019	0.0157	0.3854	0.0877	0.0130
17	0.5634	0.1305	0.0175	0.2897	0.0684	0.0117	0.3297	0.0852	0.0249	0.4436	0.1000	0.0130	0.3850	0.0862	0.0109
19	0.5631	0.1300	0.0171	0.2893	0.0680	0.0113	0.3286	0.0830	0.0222	0.4434	0.0998	0.0129	0.3849	0.0860	0.0106
21	0.5629	0.1297	0.0167	0.2889	0.0675	0.0107	0.3267	0.0781	0.0155	0.4433	0.0996	0.0126	0.3848	0.0859	0.0105

Table 3.3: MMSE with L=1, for 8V29, Squared Minimum Distance = 0.7273

	Channel A			Channel B			Channel C			Channel X			Channel Y		
SNR $2N+1$	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB
1	0.6814	0.3239	0.2499	0.3811	0.2011	0.1668	0.4258	0.2256	0.1886	0.5198	0.2001	0.1318	0.4708	0.2086	0.1552
3	0.6132	0.1453	0.0314	0.3640	0.1693	0.1294	0.4254	0.2245	0.1873	0.4931	0.1480	0.0715	0.4269	0.1266	0.0605
5	0.6110	0.1432	0.0285	0.3150	0.0904	0.0400	0.4236	0.2211	0.1832	0.4900	0.1432	0.0665	0.4247	0.1239	0.0576
7	0.6089	0.1385	0.0229	0.3148	0.0873	0.0345	0.3998	0.1792	0.1351	0.4861	0.1387	0.0617	0.4221	0.1211	0.0546
9	0.6078	0.1368	0.0212	0.3085	0.0743	0.0183	0.3612	0.1131	0.0592	0.4740	0.1049	0.0176	0.4099	0.0910	0.0163
11	0.6071	0.1355	0.0198	0.3081	0.0739	0.0175	0.3559	0.1046	0.0496	0.4737	0.1044	0.0170	0.4098	0.0902	0.0150
13	0.6066	0.1346	0.0188	0.3066	0.0707	0.0134	0.3537	0.0969	0.0457	0.4735	0.1040	0.0165	0.4095	0.0897	0.0142
15	0.6062	0.1339	0.0181	0.3061	0.0703	0.0131	0.3505	0.0886	0.0277	0.4735	0.1036	0.0157	0.4094	0.0890	0.0131
17	0.6059	0.1334	0.0176	0.3055	0.0692	0.0117	0.3493	0.0862	0.0249	0.4729	0.1017	0.0131	0.4090	0.0875	0.0110
19	0.6057	0.1330	0.0171	0.3051	0.0688	0.0113	0.3482	0.0840	0.0222	0.4728	0.1016	0.0129	0.4089	0.0873	0.0106
21	0.6055	0.1326	0.0168	0.3047	0.0683	0.0107	0.3464	0.0791	0.0155	0.4727	0.1013	0.0126	0.4088	0.0872	0.0105

Table 3.4: MMSE with L=1, for 16QAM, Squared Minimum Distance = 0.4

	Channel A			Channel B			Channel C			Channel X			Channel Y		
SNR $2N+1$	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB	8 dB	15 dB	25 dB
1	0.7698	0.3304	0.2500	0.4203	0.2030	0.1668	0.4736	0.2280	0.1886	0.5920	0.2046	0.1319	0.5297	0.2119	0.1552
3	0.7205	0.1535	0.0314	0.4043	0.1713	0.1294	0.4733	0.2270	0.1873	0.5694	0.1527	0.0715	0.4908	0.1303	0.0605
5	0.7189	0.1514	0.0286	0.3584	0.0926	0.0400	0.4716	0.2235	0.1832	0.5667	0.1480	0.0665	0.4888	0.1275	0.0576
7	0.7174	0.1468	0.0230	0.3582	0.0895	0.0345	0.4498	0.1818	0.1352	0.5634	0.1435	0.0618	0.4865	0.1248	0.0546
9	0.7166	0.1451	0.0213	0.3524	0.0766	0.0183	0.4144	0.1159	0.0593	0.5531	0.1099	0.0177	0.4756	0.0948	0.0163
11	0.7161	0.1438	0.0199	0.3519	0.0761	0.0175	0.4095	0.1074	0.0496	0.5528	0.1094	0.0171	0.4756	0.0940	0.0150
13	0.7157	0.1429	0.0189	0.3505	0.0730	0.0135	0.4075	0.1039	0.0457	0.5527	0.1090	0.0166	0.4753	0.0935	0.0142
15	0.7155	0.1422	0.0182	0.3501	0.0726	0.0131	0.4046	0.0915	0.0277	0.5526	0.1085	0.0158	0.4752	0.0928	0.0131
17	0.7153	0.1417	0.0177	0.3495	0.0715	0.0117	0.4035	0.0891	0.0250	0.5522	0.1067	0.0130	0.4748	0.0913	0.0110
19	0.7151	0.1413	0.0172	0.3491	0.0711	0.0113	0.4025	0.0869	0.0222	0.5521	0.1065	0.0129	0.4748	0.0911	0.0107
21	0.7149	0.1409	0.0169	0.3488	0.0705	0.0107	0.4008	0.0820	0.0156	0.5520	0.1063	0.0126	0.4747	0.0910	0.0106

Table 3.5: MMSE with L=1, for 16V29, Squared Minimum Distance = 0.2963

	L=2			L=3			L=5		
SNR Constellation	8dB	15dB	25dB	8dB	15dB	25dB	8dB	15dB	25dB
4PSK	0.375	0.067	0.004	0.449	0.080	0.004	0.489	0.087	0.004
8PSK	0.375	0.067	0.004	0.449	0.080	0.004	0.489	0.087	0.004
8V29	1.012	0.205	0.004	1.173	0.234	0.008	1.243	0.244	0.008
16QAM	1.274	0.266	0.010	1.440	0.297	0.014	1.509	0.307	0.014
16V29	1.833	0.427	0.020	2.062	0.468	0.024	2.148	0.483	0.024

Table 3.6: Average Gain in MMSE dB over (L=1) for 9 Equalizer Taps

3.5 Observations

For each of the tested channels, we observed the following: For a specific number of equalizer taps, the higher the SNR is, the lower is the MMSE. For a reasonably small MMSE, the SNR should be at least 25 dB. For a given SNR, the MMSE decreased monotonically as the number of taps increased. The reduction in MMSE by increasing the number of equalizer taps is larger at higher SNR. Increasing the number of taps above nine does not reduce the MMSE appreciably and thus does not improve the system performance significantly.

For each constellation and fixed value of L, the number of equalizer taps and the SNR were varied and the channels were placed in order of increasing MMSE as shown in Table 3.7. For an SNR of 25 dB, channels A and C have the largest MMSE values. For an SNR of 8 dB, channels A and X have the largest MMSE. Thus, equalization of channel C is more sensitive to the SNR (i e. larger noise enhancement) than channel A. In addition, at an SNR of 25 dB, channel Y has the smallest MMSE and at 8 dB, channel B has the smallest. Thus, Y has the least ISI but the addition of noise degrades the performance of the MMSE equalizer in channel Y more than it

Number of Equalizer Taps	SNR in dB	Channels in Order of Increasing MMSE
9	8	B, C, Y, X, A
	15	B, Y, X, C, A
	25	Y, X, B, A, C
21	8	B, C, Y, X, A
	15	B, C, Y, X, A
	25	Y, B, X, C, A

Table 3.7: Channels in Order of Increasing MMSE.

does in channel B. This shows that channel Y can be better equalized than B, at the expense of a larger noise enhancement.

Reference phase estimation errors are due to channel noise and ISI only since in our analysis, perfect receiver decisions were assumed. In addition, the amplitude of the signal points also affects the reference phase errors since it determines the symbol SNR. As a result of these reference phase errors, the MMSE depends also on the value of L and the size and type of signal constellation. This can be seen from (3.36). The MMSE dependence on these two parameters will now be discussed: From (3.37), the column vector $\underline{\hat{B}}$ differs from perfect phase estimation column vector $\underline{B}$ by a factor which is proportional to the variance of the phase estimation error $\eta[n]$, i.e. $E\left[\eta^2[n]\right]$ (3.36). From the results, a number of observations can be made:

First, for very high SNR, i.e. more than 25 dB, the MMSE results of all signal constellations approach the ideal MMSE results for a coherent receiver regardless of the value of L, since $E\left[\eta^2[n]\right]$ approaches zero for very high SNR (3.36).

Second, the MMSE results were observed to be the same for 4PSK and 8PSK always. This was because, for MPSK, $E\left[\eta^2[n]\right]$ is independent of the constellation size M and inversely proportional to L since $b[n]$ is constant and equal to unity (3.36).

However, although they give the same MMSE results, 4PSK has a smaller probability of error P_e than 8PSK since its minimum distance is larger. Therefore, for the same P_e, the SNR of the 8PSK constellation must be raised to a suitable higher value.

Third, 16V29, 16QAM and 8V29 gave larger MMSE results than MPSK Thus, constellations with signal points of varying amplitudes have degradations in performance, i.e. larger MMSE results, compared to constant amplitude signal constellations. In addition, the 16V29 constellation gave larger MMSE results than both 16QAM and 8V29, since it has signal points with smallest amplitudes. Therefore, constellations with smaller amplitude signal points have larger degradations in MMSE performance.

Using a larger L, the constellations with smaller amplitude symbol points had larger MMSE performance gains, i.e. larger reductions in MMSE. Thus, by increasing L, 16V29, 16QAM, 8V29 and MPSK had performance gains which decreased in that order. As a result, using a larger L reduces the difference in MMSE performance between the V29, QAM, and PSK constellations. Furthermore, by increasing L, the system performance approaches that of combined coherent detection and equalization. In addition, the gain in MMSE(dB), i.e. μ, by using a value of L larger than one, was very significant, especially for low SNR. Also, using L=3 or L=5 gives appreciable gains in performance over L=2. However, larger values of L do not yield appreciable performance gains over L=3. Therefore, three appears to be the best value for L. This is because increasing L increases the SNR of the reference signal from which the phase reference is extracted until it approaches coherent PSK. It appears that the reference phase SNR of the differential detected signal sufficiently approaches that of a coherently detected signal at L=3.

Finally, the difference in MMSE between coherent and differentially coherent detection is smaller here than in [20]. This is due to the way that the reference phase is derived in this work. In [20], an adaptive equalizer was used for differentially coherent reception and the MMSE obtained was about 3 dB more than that obtained in the coherent case. One previous equalizer output was used to generate the reference estimate and its conjugate was used in the decision variable, together with the equalizer output. Thus, errors in the reference estimate caused *both amplitude and phase* errors in the receiver's decisions. In our case, the improved phase reference estimate $\hat{\beta}[n]$, (which can be generated by using more than one past equalizer output to smooth channel noise), is used *only* to phase-shift the current equalizer output. In other words, we process the reference sample by a limiter which removes the amplitude noise. Thus, our reference estimate causes *only phase* errors in the receiver's decisions and therefore, the difference in MMSE between coherent detection and differential detection is less than 3 dB in our case. In addition, increasing L allows the system performance to approach that of combined coherent detection and linear equalization. As a result, our proposed receiver has better system performance which approaches that of combined coherent detection and linear equalization.

Chapter 4

Adaptive Equalization for Unknown Channels

The combination of decision-feedback differentially coherent detection with adaptive equalization is considered in this chapter. In Section 4.1, the conventional Least-Mean-Square (LMS) adaptive algorithm and some fast-converging algorithms, e.g. Kalman are briefly reviewed. Following this, the LMS algorithm, which is used for adapting the linear equalizer, is described. Simulation results (for a specific number of equalizer taps, SNR and L) and graphs which compare average convergence rates and residual MSEs for different test cases (i.e. different constellations, channels and step-sizes.), are presented in Section 4.2. Finally, these results are discussed in Section 4.3.

4.1 The LMS Adaptive Equalizer

For many practical wireless systems, the channel characteristics are usually not known beforehand, and therefore the equalizer must adapt to the unknown channel. In addition, the characteristics of these channels may vary sufficiently with time so that adaptive equalization is also necessary during normal data transmission.

A comprehensive survey on the early days of adaptive equalization can be found in [17]. In 1960, Widrow and Hoff [14] presented the Least-Mean-Squares (LMS) error adaptive filtering scheme which has been used extensively in the last three decades. In addition, key papers [27] and [28] have contributed to the understanding of the convergence of the LMS stochastic update algorithm for transversal equalizers, including the effect of channel characteristics (eigenvalue spread of the auto-correlation matrix) and the number of equalizer taps on the rate of convergence. First, in [27], the assumption of statistical independence for the random equalizer input vectors $\underline{y}[n]$ (from one instant [n] to another instant [n+1]), which direct equalizer convergence, was investigated and it was found that although this assumption is far from true, the results obtained using this assumption are in excellent agreement with the actual performance of the LMS equalizer convergence.

In [28], Ungerboeck considered the MSE criterion instead of the expected tap-gain errors relative to their optimum values (considered by Gersho in [21]). In addition, he assumed the equalizer input vectors $\underline{y}[n]$ at successive instants to be statistically independent and showed that the influence of the number of equalizer taps, and not only the channel characteristics, dominates the speed of convergence. This was opposed to [21], where the speed of convergence (for Gersho's criterion, i.e. the expected tap-gain errors relative to their optimum values) was shown to depend only on the channel characteristics. As a result, Ungerboeck suggested a new criterion for stability, which imposed a much narrower upper bound on the step-size than the one found in [21] and a corresponding optimum initial step-size parameter for LMS adaptive equalization. Finally, he showed the MSE convergence is faster in practice than theoretically predicted and suggested that step-sizes slightly less than the optimum step-size should be chosen, since the assumption of statistical independence of the equalizer input vectors $\underline{y}[n]$ at successive instants is not true in practice.

In our simulations, the LMS algorithm is used to adapt the linear equalizer to the channel because of its simplicity and robustness. However, its main drawback is its slow convergence compared with the more sophisticated algorithms [23]–[25]. In [23], the Kalman filtering algorithm was described. It can be used to estimate the equalizer coefficients vector at each symbol interval and its convergence rate was shown to be proportional to the number of equalizer taps and independent of the eigenvalue spread. However, it requires on the order of N^2 operations per iteration for an equalizer with N taps. In [24], a self-orthogonalizing algorithm was compared to the Kalman algorithm of [23] and the LMS algorithm. The algorithm tries to accelerate the rate of convergence by reducing the eigenvalue spread of the channel-correlation matrix, i.e. by making the eigenvalues equal, since a large eigenvalue spread slows the rate of convergence. It was found that the proposed self-orthogonalizing algorithm, which was less complex than the Kalman, converged much faster than the LMS algorithm but was slower than the Kalman algorithm. The Kalman algorithm of [23] was later recognized as a form of a Recursive-Least-Squares (RLS) algorithm and the idea of fast Kalman filtering was introduced [25]. This algorithm took advantage of the data structure by using the "shifting property" of RLS algorithms and reduced its computational complexity to an order of N operations per iteration for an equalizer with N taps. Therefore, the algorithm performs as well as the one in [23] while avoiding its computational complexity.

The LMS algorithm will now be discussed. It is similar to the MSEG algorithm (3.41) but uses an instant squared error instead of the mean squared error because the ensemble averages represented by the matrix A and $\underline{\hat{B}}$ are not known in practice. The LMS algorithm is also referred to in the literature as the stochastic gradient (SG) algorithm [13]. Using the LMS algorithm, the filter coefficient vector $\underline{c}$

is updated by

$$\underline{c}[n+1] = \underline{c}[n] - \frac{\lambda}{2}\frac{\partial|\epsilon[n]|^2}{\partial\underline{c}[n]} \tag{4.1}$$

where $\epsilon[n]$ is the error at the n-th iteration, [n] denotes a particular symbol interval (or time instant t=nT), $\underline{c}[n] = [c_{-N}[n],\ldots,c_0[n],\ldots,c_N[n]]$ and λ is the step-size. Using (3.2), the error is given by:

$$\epsilon[n] = z[n]e^{-j\hat{\beta}[n]} - \underbrace{b[n]e^{j\varphi[n]}}_{desired\ signal}$$

where $\hat{\beta}[n]$ is the reference phase estimate. Differentiating the instant squared error $|\epsilon[n]|^2$ with respect to $\underline{c}[n]$, we get :

$$\frac{\partial|\epsilon[n]|^2}{\partial\underline{c}[n]} = 2(z[n]e^{-j\hat{\beta}[n]} - b[n]e^{j\varphi[n]})\underline{y}^*[n]e^{j\hat{\beta}[n]} \tag{4.2}$$

where $z[n] = c^T[n]y[n]$. Therefore, substituting (4.2) in (4.1), we get :

$$\begin{aligned} \underline{c}[n+1] &= \underline{c}[n] - \lambda\underbrace{(z[n]e^{-j\hat{\beta}[n]} - b[n]e^{j\varphi[n]})}_{\epsilon[n]}\underline{y}^*[n]e^{j\hat{\beta}[n]} && (4.3) \\ \underline{c}[n+1] &= \underline{c}[n] - \lambda\epsilon[n]\underline{y}^*[n]e^{j\hat{\beta}[n]} && (4.4) \end{aligned}$$

Thus, each equalizer tap $c_k[n]$ is updated using the error $\epsilon[n]$, the phase reference estimate $\hat{\beta}[n]$ and the received sample $y[n+k]$ for $k = -N,\ldots,0,\ldots,N$.

The algorithm of (4.4) will now be explained referring to Figure 2.4. The equalizer adaptation is driven by the *error* signal $\epsilon[n]$, which indicates to the equalizer in which direction the coefficients $c_k[n]$ must be changed to reduce the squared error $|\epsilon[n]|^2$. Specifically, the input sample to the equalizer, $y[n-k]$ is taken from the output of the same unit delay and is used for multiplication by $c_k[n]$. The resulting product contributes to the summation for z[n], which is then phase-shifted by $\hat{\beta}[n]$ and the data symbol $b[n]e^{j\varphi[n]}$ is subtracted from it to give the error $\epsilon[n]$. The increment of the tap coefficient $c_k[n]$ is $-\lambda\epsilon[n]y^*[n-k]e^{j\hat{\beta}[n]}$, where $y^*[n-k]$ is phase-shifted by $\hat{\beta}[n]$ to compensate for the unknown rotation of these samples.

In wireless communication systems, the adaptive equalizer should be able to track the time-varying multipath characteristics usually encountered. Therefore, the rate of convergence of the adaptive algorithm employed is very important and is determined by the step-size λ. For the LMS algorithm, the best convergence rate and the allowable values of the step-size λ, which guarantee stability of convergence, are dictated by the number of equalizer coefficients $(2N+1)$, and to a lesser extent, by the eigenvalue spread of the matrix A (i.e. which depends on channel characteristics) [28]. From [28], the allowable step-sizes λ are

$$0 < \lambda < \frac{2}{(2N+1)E[|y[n]|^2]} \left(= \frac{2}{\lambda_1 + \ldots + \lambda_{2N+1}}\right) \tag{4.5}$$

where $\lambda_1, \ldots, \lambda_{2N+1}$ are the $2N+1$ eigenvalues of the auto-correlation matrix A and $E[|y[n]|^2]$ is the expected squared amplitude of the equalizer input $y[n]$. Also, the optimum step-size suggested is

$$\lambda_{opt} = \frac{1}{(2N+1)E[|y[n]|^2]} \tag{4.6}$$

The dependence of LMS convergence on λ is as follows: Starting with zero, as we increase λ, the speed of convergence and the residual MSE increases, until we reach the maximum speed at λ_{opt}. Continuing to increase λ, slows the rate of convergence (but the residual MSE still increases) until eventually we reach instability at twice the optimum step-size. Therefore, there is a tradeoff between the rate of convergence and the residual MSE. In fact, for fastest convergence, the residual MSE is twice that of the MMSE [13]. Therefore, if the step-size is too small, the equalizer would not adapt fast enough (i.e. within an agreed time frame or number of symbols) or if it is too large, the equalizer would blow up (not stable) (4.5). Therefore, λ should be chosen such that the rate of convergence is fast yet has a reasonable (not necessarily minimum) residual MSE.

In adaptive equalization, there are two modes. The first mode is the initial acquisition which uses a training sequence which is known to the receiver. This mode is used to initially adapt the equalizer to the channel and, thus uses actual data $be^{j\varphi}$ to generate the error signal. Once the equalizer converges in an specific period of time, the second mode of adaptive equalization can begin. In the second mode, actual receiver decisions are substituted for the known training sequence and normal data transmission occurs. This mode of equalizer adaptation is called the *decision-directed* mode since receiver decisions $\hat{b}e^{j\hat{\varphi}}$ are used to generate the error $\epsilon[n]$, and the phase estimate $\hat{\beta}[n]$. This is seen from Figure 2.4 and equalizer adaptation takes place in a decision-feedback manner. However, this mode of equalizer adaptation cannot track fast variations in the channel characteristics. As a result, it may be necessary to use the first mode to re-adapt the equalizer to the channel.

The adaptive MSE (AMSE) simulation results using the LMS adaptive algorithm for different test cases will now be discussed.

4.2 Simulation Results

Simulations of the equalizer adaptation were performed using the LMS algorithm. Three parameters of the simulations were kept constant:

- Nine (=$2N$+1) equalizer taps were used. Using a larger number of taps increases the delay in the equalizer and does not result in any substantial gain in performance, with the channels that were tested in this work.
- Three equalizer samples used to generate the improved reference phase estimate (i.e. L was chosen to be 3) since the gain in performance over L=2 is substantial and because using any larger value of L e.g. L=5 does not result in an appreciable gain in performance.
- The SNR was set to 25 dB. A lower SNR would require a prohibitive large number of simulations.

The three other parameters in the simulations form the basis for different test cases. These parameters are the signal constellation, the channel and the step-size λ. The choices for each were as follows:

- Four constellations: 8PSK, 8V29, 16QAM and 16V29.
- Two channels: A and X.
- Step-sizes: 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05 and 0.1. For our nine tap equalizer and with $E[|y[n]|^2] \approx 1$, Ungerboeck's optimal step-size is 0.1. Results were examined and the two step-sizes λ=0.005 and λ=0.05 were chosen for presentation since they best summarize the trade-offs in selecting the step size.

For the LMS simulations, training sequences were used, i.e. perfect receiver decisions were assumed. Each sequence had a length of 3220 data symbols to ensure that steady-state convergence had been achieved. All nine equalizer taps were initialized to zero for each trial.

Initially, in our simulations, twenty independent trials were performed for each test case (i.e. choice of constellation, channel and step-size) and an average learning curve was calculated. However, the average learning curves were very noisy due to an insufficient number of trials. At an SNR of 25 dB, it was found that we need approximately sixty trials to get reasonable smooth average learning curves. All simulation results were then examined and are summarized by eight graphs and two tables, shown on the next few pages.

The first four graphs, i.e. Figures 4.1–4, were each derived for a separate test case (i.e. either 8PSK or 16QAM used in either channel A or X, using λ equal to 0.005). Each graph plots the squared error versus the number of iterations and compares sixty trial runs with an average learning curve. It is seen that the number

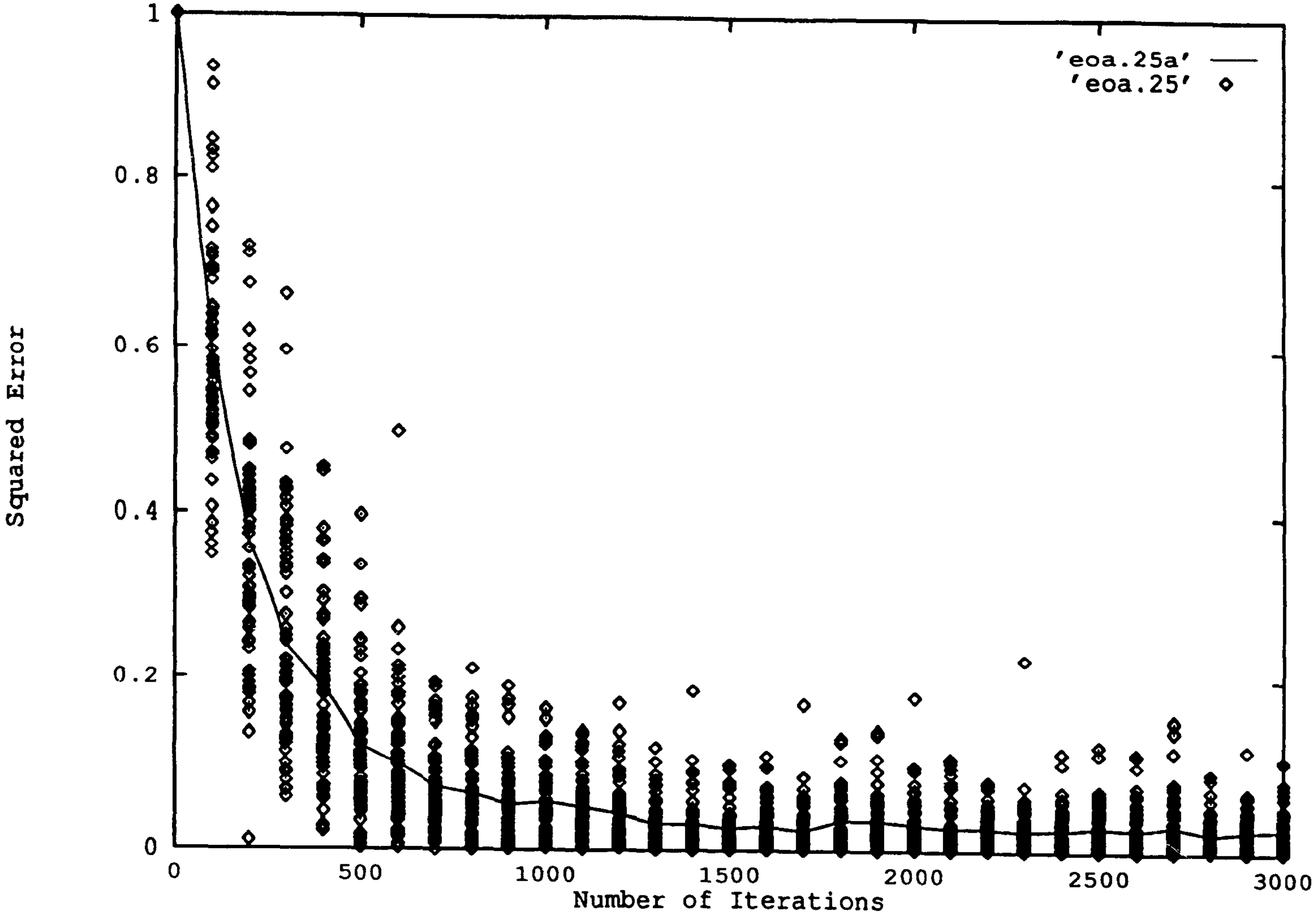

Figure 4.1: Sixty trials and Average Learning Curve for 8PSK, channel A and λ=0.005

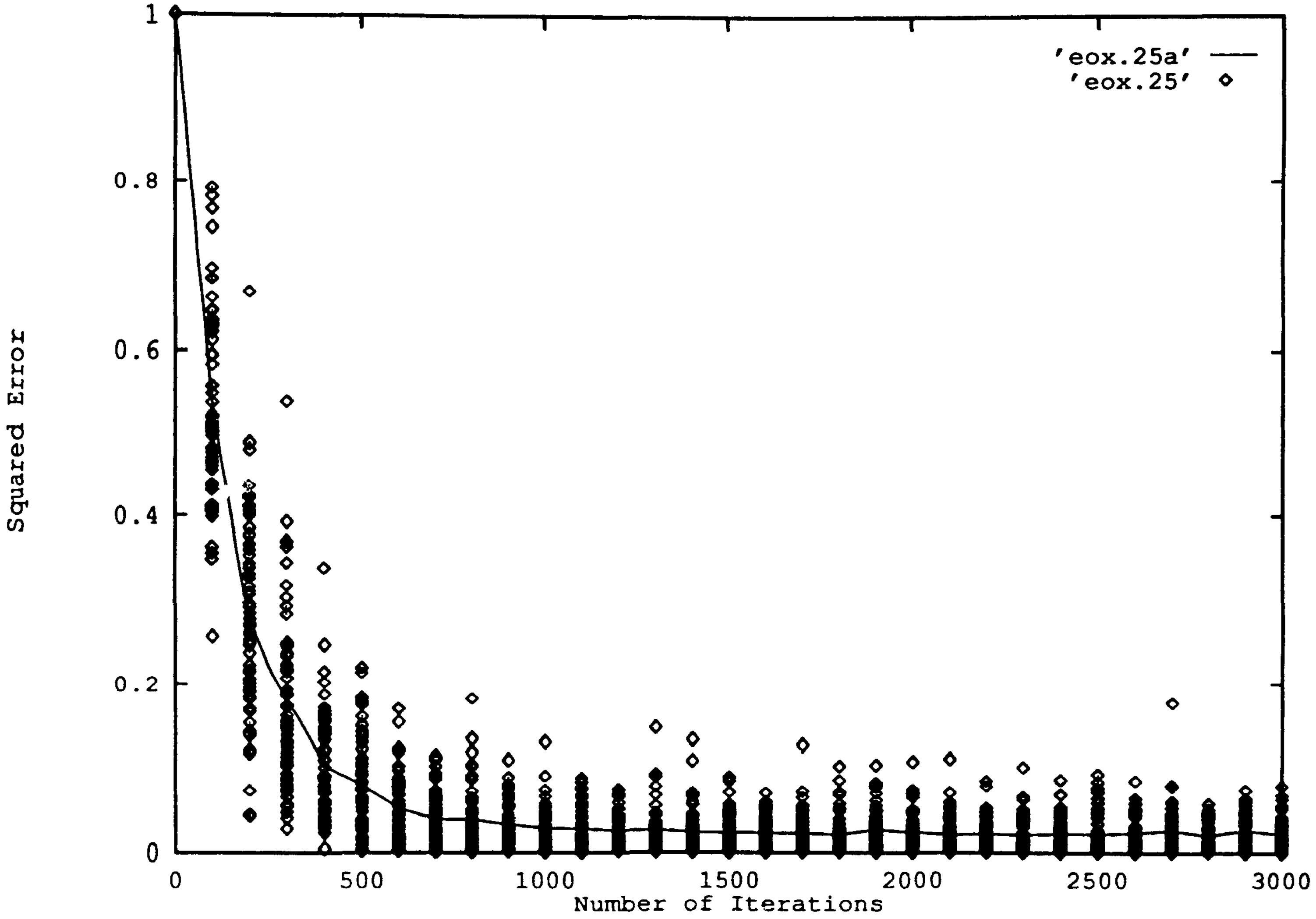

Figure 4.2: Sixty trials and Average Learning Curve for 8PSK, channel X and λ=0.005

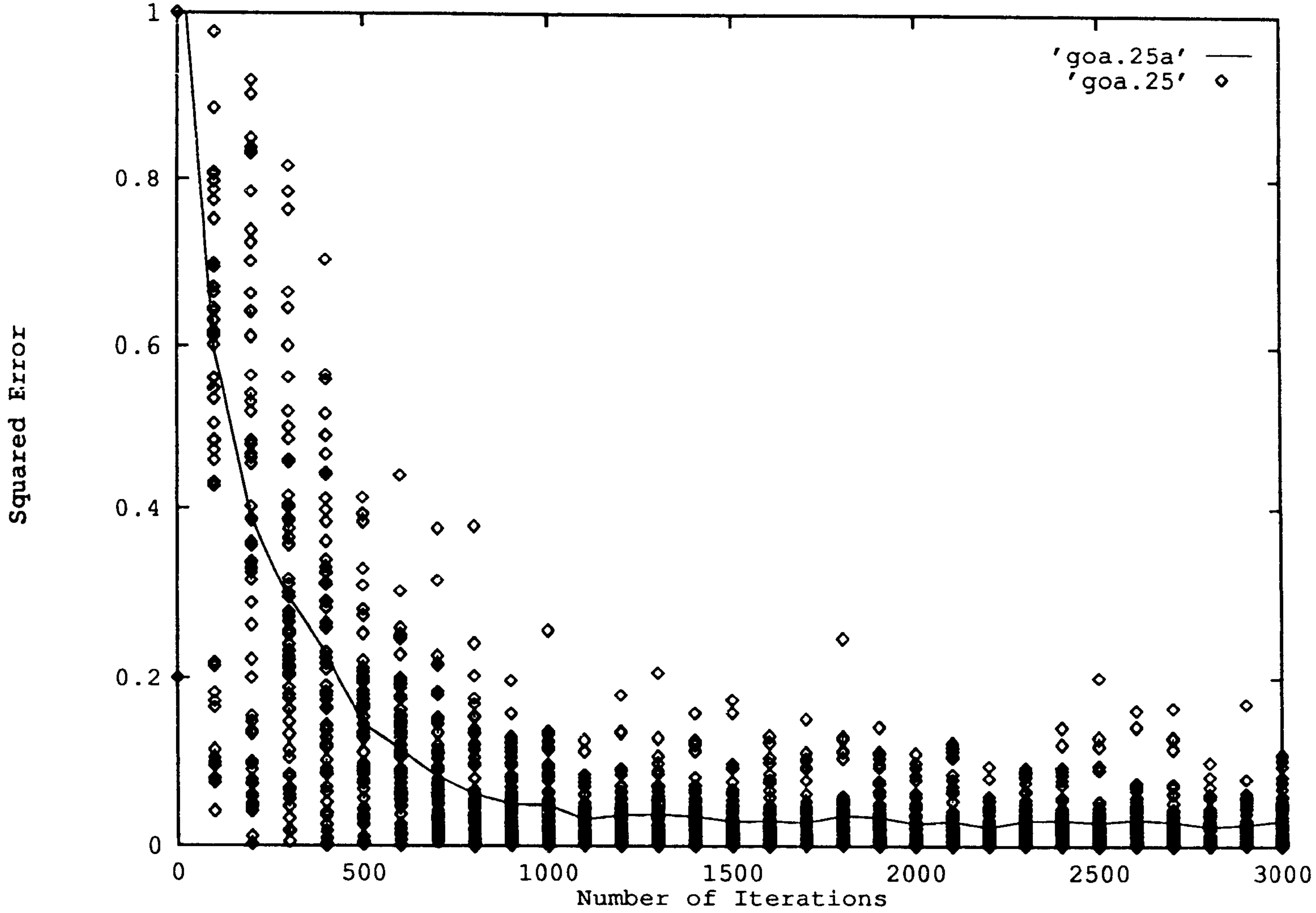

Figure 4.3: Sixty trials and Average Learning Curve for 16QAM, channel A and λ=0.005

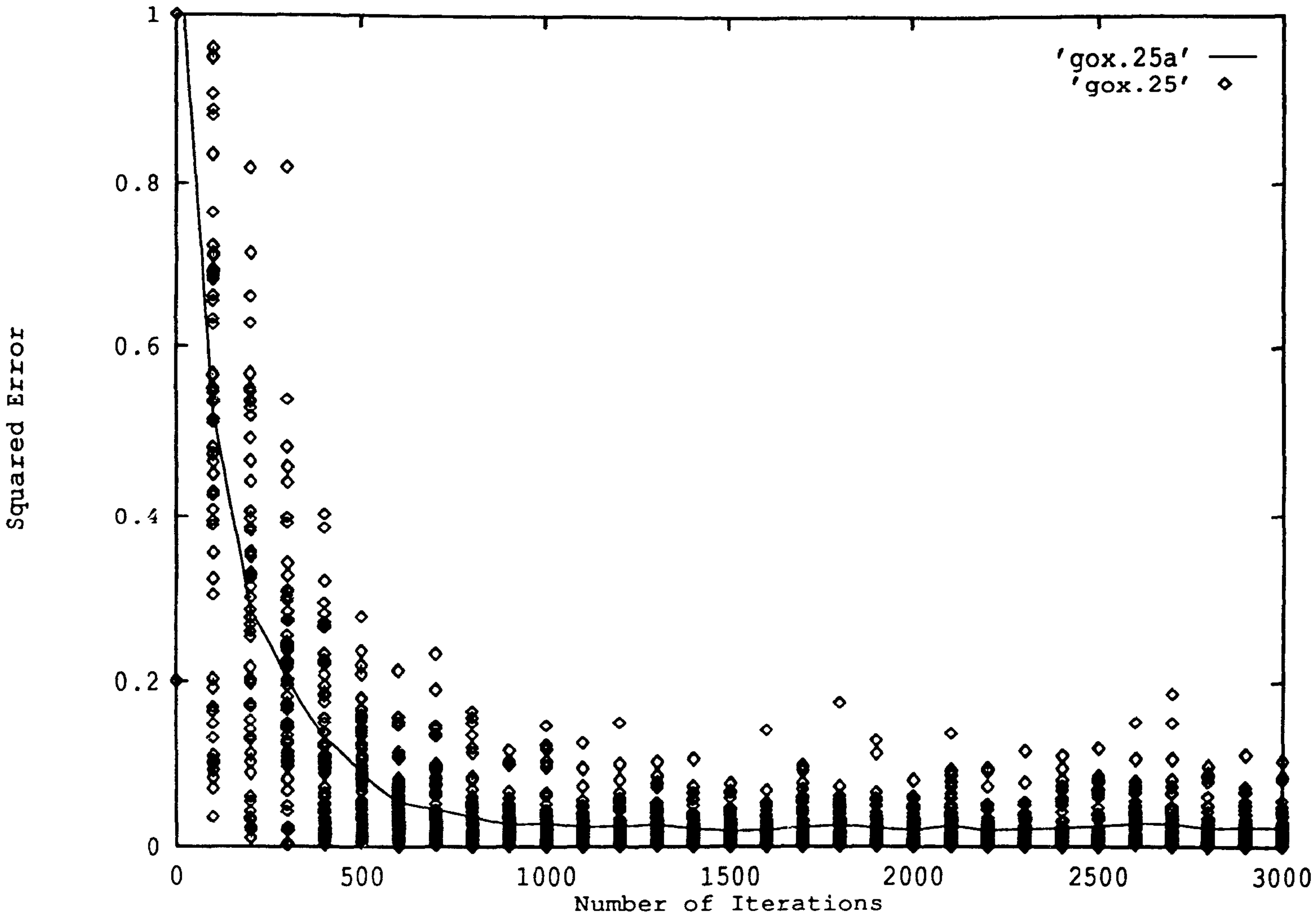

Figure 4.4: Sixty trials and Average Learning Curve for 16QAM, channel X and λ=0.005

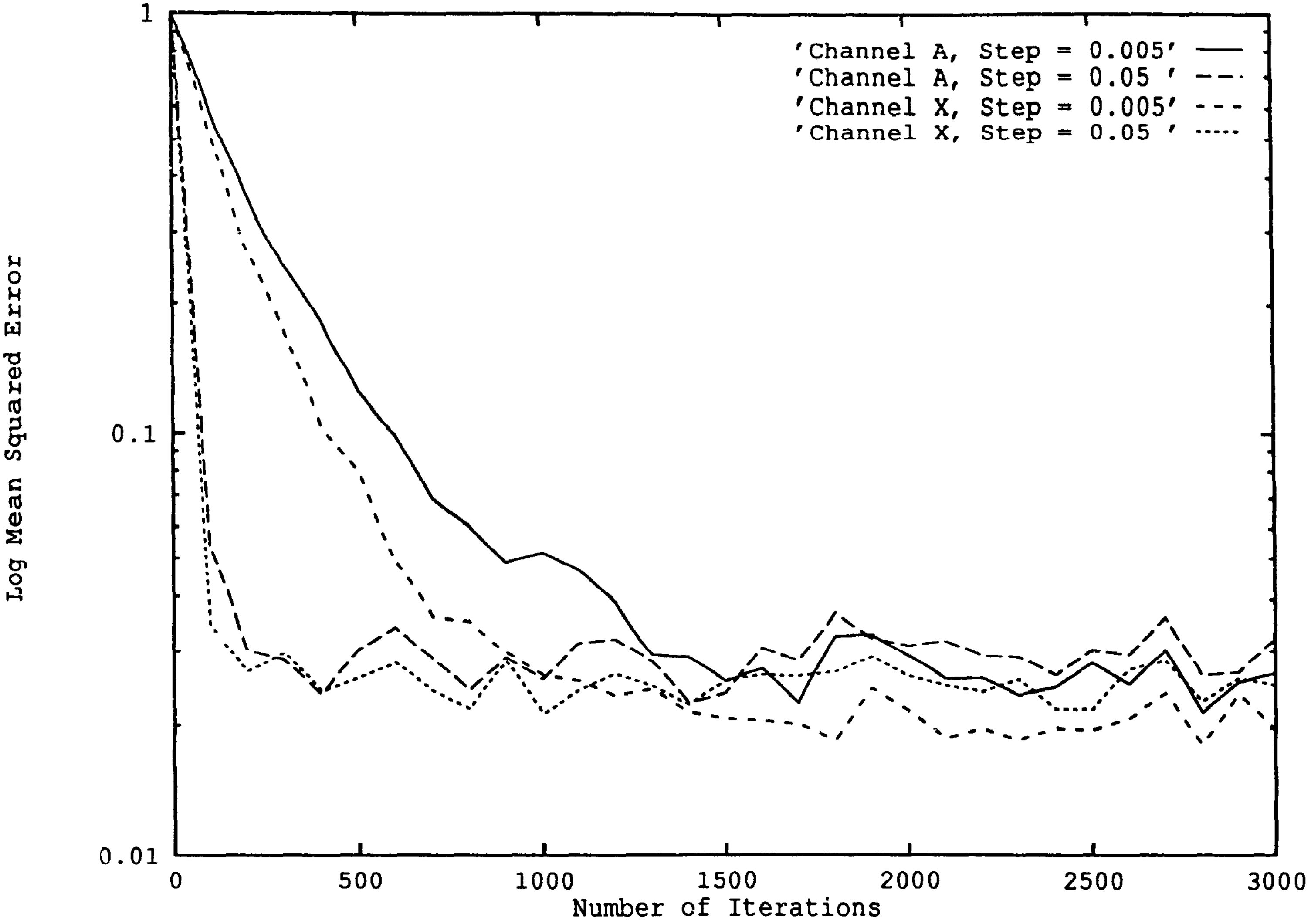

Figure 4.5: Average Learning Curves for 8PSK.

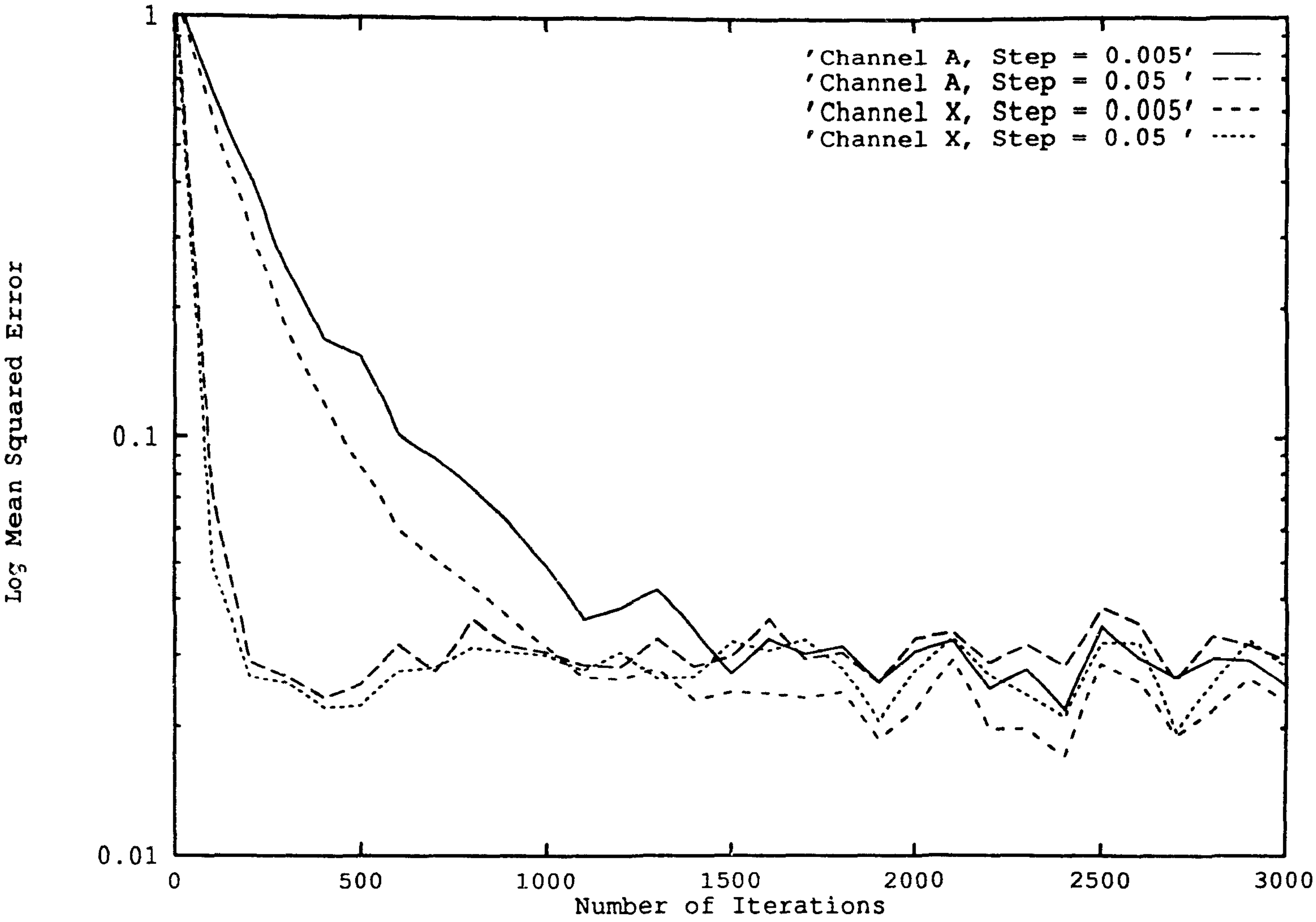

Figure 4.6: Average Learning Curves for 8V29.

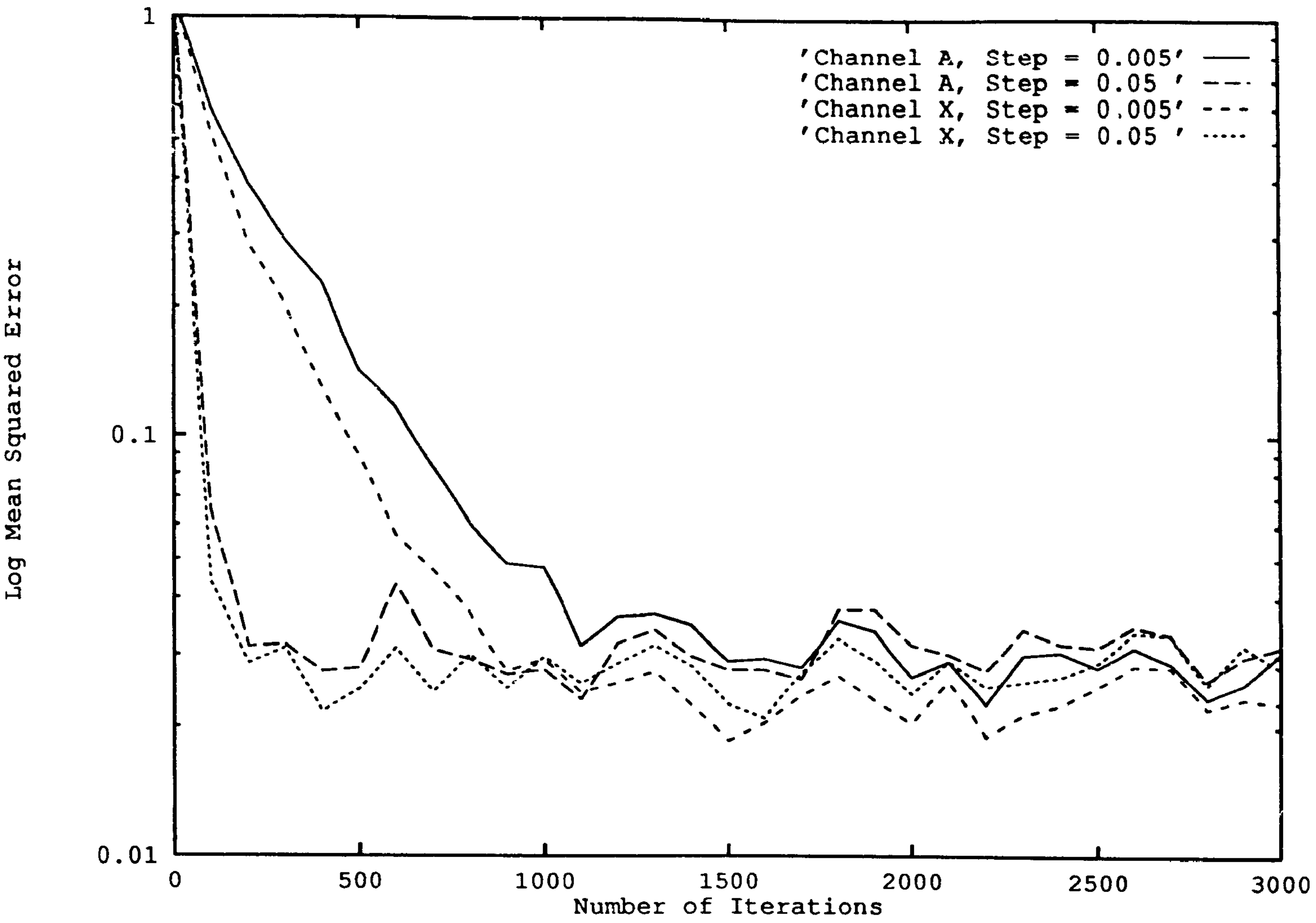

Figure 4.7: Average Learning Curves for 16QAM.

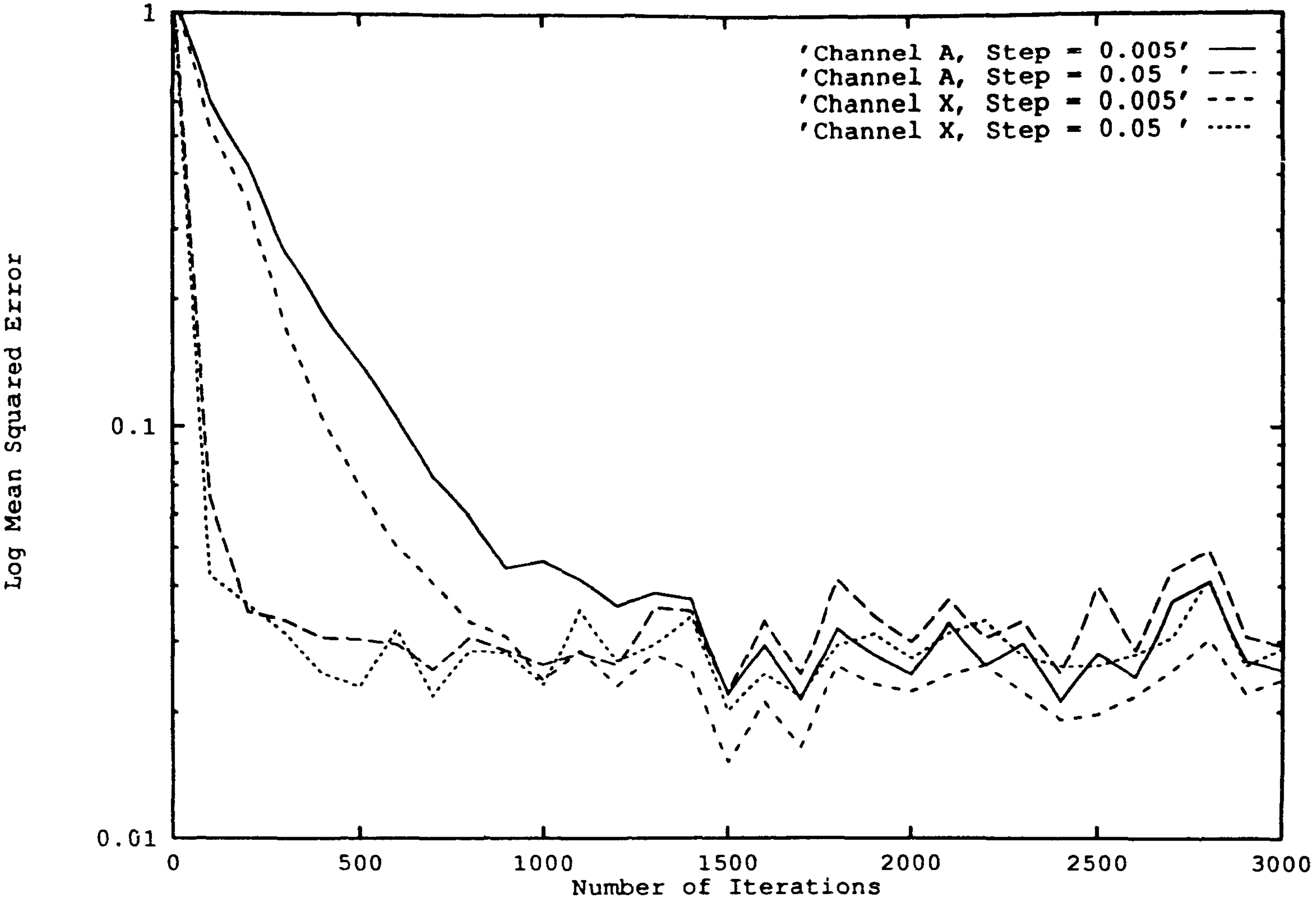

Figure 4.8: Average Learning Curves for 16V29.

	Channel A		Channel X	
Step-Size Constellation	0.005	0.05	0.005	0.05
8PSK	0 026	0.032	0.021	0.026
8V29	0.025	0.029	0.023	0.028
16QAM	0.027	0.032	0.023	0.028
16V29	0.025	0.029	0.024	0.029
MMSE Results	0.0210		0.0176	

Table 4.1: Comparison of Residual MSEs and MMSEs for 25 dB

of independent trials, i.e. 60, used to calculate the curves was sufficient since the dispersion is reasonable.

Consequently, the last four graphs, i.e. Figures 4.5-8, compare the average learning curves of different test cases. Each graph corresponds to a particular constellation, and compares four different test cases (i.e. either λ=0.005 or λ=0.05 used in either channel A or X). Finally, each graph plots the logarithm of the MSE versus the number of iterations so that differences in convergence behaviour between the different test cases can be noticed more easily.

Table 4.1 compares the residual MSEs obtained from Figures 4.5–8 with the calculated MMSE results of Section 3.4. In order to compare the constellations from a probability of error point of view, we used the squared minimum Euclidean distance d^2_{min} normalised to the number of bits per symbol, $\log_2 M$, and the residual MSE, ξ. This quantity in [dB] is given by:

$$\sigma = 10\log_{10}\left[d^2_{min}\,\frac{\log_2 M}{\xi}\right] \tag{4.7}$$

Although, σ may not give an accurate indication to the actual probability of error $P_e \approx K\,\exp\left[d^2_{min}\,\frac{\log_2 M}{\xi}\right]$ which can be arrived at since P_e is proportional to the

Constellation	σ for Channel A			σ for Channel X		
	λ=0.005	λ=0.05	with MMSE	λ=0.005	λ=0.05	with MMSE
8PSK	18.3	17.4	19.2	19.2	18.3	20.0
8V29	19.4	18.8	20.2	19.8	18.9	20.9
16QAM	17.7	17.0	18.8	18.4	17.6	19.6
16V29	16.7	16.1	17.5	16.9	16.1	18.3

Table 4.2: σ for different test cases

exponent of the SNR [13] and $\left[d^2_{min}\frac{\log_2 M}{\xi}\right]$ represents the normalized SNR for any M-ary constellation, it can be used for a benchmark comparison of the different schemes from the probability of error point of view. As a result, we can compare σ for the tested constellations under the same SNR (see Table 4.2). The larger σ is, the smaller the probability of error and the better is the system performance.

Finally, the simulation results for a sample AMSE test case are given in Appendix B.

4.3 Observations

For the first four graphs, i.e. Figures 4.1–4, we see that the average learning curves for sixty independent trials and an SNR of 25 dB are reasonably smooth and give a good indication of the average convergence performance. If the curves were too noisy, more independent trials would have been required to calculate the average learning curves. Finally, we note that the trial runs give a better indication of what should be expected in an actual system implementation.

We will now look at Figures 4.5–8, where each figure compares the average convergence rates for different channels and step-sizes, for a given constellation. A

MSE convergence cutoff point of 0.05 was chosen since the average learning curves passed through this level *only* once before settling down to the residual MSE levels between 0.02 and 0.035. Thus, the average learning curves for different channels and step-sizes were compared for each graph, using this MSE cutoff level of 0.05. The following was observed:

For channel A, the λ=0.05 step-size converged after approximately 125 symbols and the λ=0.005 step-size converged after about 1000 symbols. For channel X, the λ=0.05 step-size converged after approximately 100 symbols and the λ=0.005 step-size converged after about 750 symbols. Thus, for a given λ, the rate of convergence is faster for channel X than for channel A. In addition, the rate of convergence for the λ=0.05 is faster than for λ=0.005.

Table 4.1 shows that the residual MSEs approach but never reach the MMSE results calculated in Section 3.4. This is due to the equalizer coefficients which are never exactly optimum. For a given λ, Table 4.1 shows that the residual MSE is smaller for channel X than for channel A. Also, for a given channel, the residual MSE is larger for λ=0.05 than for λ=0.005 In addition, it was found that the λ=0.05 and λ=0.005 step-sizes have average residual MSEs of 50% and 25% excess MSEs respectively, where excess MSE is the MSE over and above the MMSE possible. As a result, this shows that the larger the step-size λ, the larger the excess MSE, since the equalizer coefficients have a larger variance about the optimum values. For λ=0.05, we see that the proposed receiver adapts very quickly to unknown channels while giving a residual MSE which is only a small percentage larger than that of λ=0.005. Thus, λ=0.05 se ms to be a better choice.

In addition, the speed and stability of the MSE convergence were compared for the tested constellations. Close examination of the last four graphs and Table 4.1 shows that 16V29 converges slightly faster and has a slightly smaller residual MSE than 16QAM (which is more noticeable in channel A). However, from Table 4.2, 16QAM has a smaller probability of error due to its larger minimum distance d_{min}. Also, 8PSK is better than 8V29 in terms of both convergence speed and residual MSE which was more noticeable in channel X than in channel A. Nevertheless, the minimum distance d_{min} of 8V29 is larger than that of 8PSK, and from Table 4 2, we have that 8V29 has a smaller probability of error than 8PSK, especially for channels with severe ISI e.g. channel A.

Finally, Table 4.2 compares σ for the tested constellations and shows that σ decreases (the probability of error increases) for 8V29, 8PSK, 16QAM, 16V29, in that order, foı the tested channels. In addition, Table 4.2 allows us to calculate the increase ın SNR necessary to achieve the same σ. On the average, for λ=0.005, the SNR must be increased by 0.7 and 1.0 dB for channels A and X respectıvely, to achieve the σ associated with the MMSE. Also, for λ=0.05, the SNR must be ıncreased by an average of about 0.9 dB and 1.1 dB for channels A and X respectively, to achieve the same σ as the λ=0.005 case.

Chapter 5

Conclusions

A combined linear equalization and decision-feedback differentially coherent detection structure for indoor wireless communication channels was proposed. These channels were modeled as multipath channels since multipath propagation is one of the major impairments in wireless communication systems. In these channels, carrier phase tracking is difficult and differentially coherent reception is attractive since it does not require phase-tracking. However, there is a loss in performance compared to coherent detection that approaches 3 dB for MPSK(M>2). Therefore, an improved technique based on decision-feedback differentially coherent detection was used whose performance approaches that of coherent detection. In addition, this differentially coherent scheme can be combined quite easily with known equalization techniques. This is necessary since ISI due to multipath is a major problem in these channels. In this work, the integration of decision-feedback differential detection with linear equalization has been considered. In addition, two-dimensional signal constellations were considered, in the hope of achieving a high data transmission rate, in a given bandwidth.

The MSE criterion was used and MMSE results were calculated for known channels, taking into account reference phase estimation errors. It was seen that the MMSE performance degrades for 16V29, 16QAM, 8V29 and 8PSK in decreasing order, since constellations with signal points of smaller amplitude have a larger degradation. However, using a larger value of L, i.e. number of equalizer outputs used to generate the reference phase, reduces the degradation in MMSE performance, since constellations with smaller amplitude signal points gain more in MMSE performance. Thus, the performance of the V29 and QAM signal constellations approach that of the PSK signal constellation and a high data transmission rate can be achieved in a given bandwidth. Furthermore, increasing L allows the system performance to approach that of combined coherent detection and equalization.

In an adaptive mode, the LMS algorithm was used. The simulations were performed with a 9 tap equalizer, L=3 and an SNR of 25 dB since for known channels, these values were found to be to be sufficient for a reasonably small MMSE. Using a MSE cutoff level of 0.05, the equalizer converges within 125 iterations and has a residual MSE of about 0.029 (50% excess MSE) for λ=0.05. For λ=0.005, the equalizer converges within 1000 iterations with a residual MSE of 0.024 (25% excess MSE). Therefore, λ=0.05 seems to be the better choice. In addition, 8PSK (16V29) converges slightly faster and has a slightly smaller residual MSE than 8V29 (16QAM). However, the difference in MSE convergence performance for the tested constellations is almost negligible for L=3. Finally, at the same E_b, the constellations 8V29, 8PSK, 16QAM and 16V29 have probabilities of error in increasing order.

For a sufficiently large L, e.g. L=3, the combination of the decision-feedback differentially coherent detection structure with linear equalization performs as well as combined coherent detection and linear equalization, with the advantage that it can be used when carrier phase tracking is difficult e.g. fading multipath channels, burst communication. In addition, the receiver shows very small MMSE differences between different two-dimensional constellations. Over practical wireless channels, the proposed rece:ver seems to have significant advantages with respect to conventional coherent receivers and we will now suggest further work in this area of combining equalization with differentially coherent detection.

5.1 Suggestions for Further Work

To simplify the analysis, receiver decisions were assumed error-free with high SNR and actual information phases $\varphi[n-k]$ for $k = 1, \ldots, L-1$ were used in the equalizer adaptation simulations. Therefore, it would be of interest to analyze the effects of decision errors on the adaptation process and on the residual MSE as well. In addition, simulations should also be performed for non-zero excess-bandwidth pulses.

To improve performance, one can use decision-feedback equalizers (DFEs) with the decision-feedback differentially coherent detection structure of [2]. The DFE cancels the dominant postcursor ISI in minimum phase multipath channels without noise enhancement. Therefore, this combination is worth further investigation.

Improving the reference phase estimation for the QAM and V29 constellations may improve results. Therefore, reference phase estimation which is optimized for amplitude and phase signal constellations should be used in conjuncture with differential detection. Also, the reference phase estimation for non-stationary channels can be improved by introducing a forgetting factor. Therefore, past equalizer outputs can be weighted such that the more recent equalizer outputs will have more influence on the reference phase estimate, thus improving the phase tracking capabilities.

Finally, the adaptive equalizer should be tested using faster adaptation algorithms e.g. fast Kalman algorithm [25].

Bibliography

[1] G. D. Forney Jr., R. G. Gallager, G. R. Lang, F. M. Longstaff, and S. U. Qureshi, "Efficient modulation for band-limited channels," *IEEE J. Sel. Areas Commun.*, vol. SAC-2, no. 5, pp. 632–647, Sep. 1984.

[2] H. Leib and S. Pasupathy, "The phase of a vector perturbed by Gaussian noise and differentially coherent receivers," *IEEE Trans. Info. Theory*, vol. 34, no. 6, pp. 1491-1501, Nov. 1988.

[3] S. G. Wilson, J. Freebersyser, and C. Marshall, "Multisymbol detection of DPSK," *GLOBECOM*, Dallas, Texas, pp. 1692–1697, Nov. 27–30, 1989.

[4] D. Divsalar and M. K. Simon, "Differential detection of MPSK," *IEEE Trans. Commun.*, vol. COM–38, no. 3, pp. 300–308, Mar. 1990.

[5] H. Leib and S. Pasupathy, "Noncoherent block demodulation of PSK," *IEEE Vehicular Technology Conf.*, Orlando, Florida, pp. 407–411, May 6-9, 1990.

[6] C. R. Cahn, "Combined digital phase and amplitude modulation communication systems," *IRE Trans. Commun.*, vol.CS–8, no. 3, pp. 150–154, Sep. 1960.

[7] J. C. Hancock and R. W. Lucky, "Performance of combined amplitude and phase-modulated communications systems," *IRE Trans. Commun.*, vol. CS–8, no. 4, pp. 232–237, Dec. 1960.

[8] C. N. Campopiano and B. G. Glazer, "A coherent digital amplitude and phase-modulation scheme," *IRE Trans. Commun.*, vol. CS-10, no. 1, pp. 90–95, Mar. 1962.

[9] G. J. Foschini, R. D. Gitlin, and S. B. Weinstein, "On the selection of a two-dimensional signal constellation in the presence of phase jitter and Gaussian noise," *Bell Sys. Tech. J.*, vol. 52, no. 6, pp. 927–965, Jul.-Aug. 1973.

[10] G. J. Foschini, R. D. Gitlin, and S. B. Weinstein, "Optimization of two-dimensional signal constellations in the presence of Gaussian noise," *IEEE Trans. Commun.*, vol. COM-22, no. 1, pp. 28–37, Jan. 1974.

[11] C. M. Thomas, M. Y. Weidner, and S. H. Durrani, "Digital amplitude phase keying with M-ary alphabets," *IEEE Trans. Commun.*, vol. COM-22, no. 2, pp. 168–180, Feb. 1974.

[12] D. N. Godard, "Self-recovering equalization and carrier tracking in two- dimensional data communication systems," *IEEE Trans. Commun.*, vol. COM-28, no. 11, pp. 1867–1875, Nov. 1980.

[13] E. A. Lee and D. G. Messerschmitt, *Digital Communications.* Boston: Kluwer Academic Publishers, 1988.

[14] B. Widrow and M. E. Hoff Jr., "Adaptive switching circuits," *IRE Wescon Conv. Rec.*, Pt. 4, pp. 96–104, Aug. 1960.

[15] R. D. Gitlin, E. Y. Ho, and J. E. Mazo, "Passband equalization of differentially phase-modulated data signals," *Bell Sys. Tech. J.*, vol. 52, no. 2, pp. 219–238, Feb. 1973.

[16] J. G. Proakis, *Digital Communications.* New York: McGraw-Hill, 1983.

[17] R. W. Lucky, "A survey of the communication theory literature: 1968-1973," *IEEE Trans. Info. Theory*, vol. IT-19, no. 5, pp. 725–739, Nov. 1973.

[18] R. D. Gitlin and K. H. Mueller, "Optimization of digital postdetection filters for PSK differential detectors," *IEEE Trans. Commun.*, vol. COM-24, no. 9, pp. 963–970, Sep. 1976.

[19] B. Farhang-Boroujeny, and L. F. Turner, "An intersymbol interference cancellation equaliser for use in systems employing envelope detection," *IEE Proc., Pt. F, Commun. Radar Signal Process.*, vol. 127, no. 6, pp. 485–496, Dec. 1980.

[20] P. Sehier and G. Kawas Kaleh, "Adaptive equaliser for differentially coherent receiver," *IEE Proc., Pt. I, Commun. Speech Vision*, vol. 137, no. 1, pp. 9–12, Feb. 1990.

[21] A. Gersho, "Adaptive equalization of highly dispersive channels," *Bell Sys. Tech. J.*, vol. 48, no. 1, pp. 55–70, Jan. 1969.

[22] S. U. H. Qureshi, "Adaptive equalization," *IEEE Proc.*, vol. 73, no. 9, pp. 1349–1387, Sep. 1985.

[23] D. N. Godard, "Channel equalization using a Kalman filter for fast data transmission," *IBM J. Res. Develop.*, vol. 18, no. 3, pp. 267-273, May 1974.

[24] R. D. Gitlin and F. R. Magee, "Self-orthogonalizing adaptive equalization algorithms," *IEEE Trans. Commun.*, vol. COM-25, no. 7, pp. 666-672, July 1977.

[25] D. D. Falconer and L. Ljung, "Application of fast Kalman estimation to adaptive equalization," *IEEE Trans. Commun.*, vol. COM-26, no. 10, pp. 1439–1446, Oct. 1978.

[26] H. Sari, S. Moridi, L. Desperben, and P. Vandamme, "Baseband equalization and carrier recovery in digital radio systems," *IEEE Trans. Commun.*, vol. COM-35, no. 3, pp. 319-327, Mar. 1987.

[27] J. E. Mazo, "On the independence theory of equalizer convergence," *Bell Sys. Tech. J.*, vol. 58, no. 5, pp. 963-993, May-Jun. 1979.

[28] G. Ungerboeck, "Theory on the speed of convergence in adaptive equalizers for digital communications," *IBM J. Res. Develop.*, vol. 16, no. 6, pp. 546–555, Nov. 1972.

[29] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge University Press, 1988.

Appendix A

A.1 Program Overview

The computer program was written using the C programming language and ran under the SUN OS. The program consists of seven separate files. Data was read from a specified input file and all results were written to a specified output file. In addition, two other output files were created for ease of plotting the LMS simulation results. One file stored trial results and the other stored the average learning cvrves, i.e. average results of 60 trials.

Input File Data

- Test constellation: PSK, QAM, V29 and any other format.
- Multipath channel Parameters: Number of paths-1, amplitude attenuations, phase-shifts and relative delays
- Roll-off factor of overall desired response. Any number from 0 to 1.
- Step-sizes to be used in LMS simulations.
- Noise Power in dB.
- Number of Equalizer Taps besides the reference tap: called N, i.e. min.,max.,step.
- Number of Equalizer Outputs for phase estimate: L, i.e. min.,max.,step.
- The output data filenames.

Program Files and Functions

The seven files and their functions are as follows:

- EQ.C: Main program file. Reads input file and generates output files. Calls MMSE.C and AMSE.C.

- MMSE.C: Calculates MMSE numerical results. Calls CINV.C
- AMSE.C: Calculates LMS simulation results. Calls RANDOM.C
- CINV.C: Inverts a complex matrix using LU Decomposition.
- RANDOM.C: Generates Uniform and Gaussian distributed random numbers.
- COMPLEX.C: Library of complex arithmetic operations.
- UTIL.C: Utility subroutines.

Program Details

The convolution of the overall pulse response $\tilde{g}[n]$ with the encoded data was limited to 440 terms centered at $\tilde{g}[0]$. It was found that increasing this number to 1000 did not provide any significant differences. In the program, the number of equalizer taps was $N+1$, i.e. the equalizer had $N/2$ taps on both sides of the reference tap $c[0]$ and the maximum number allowed is 41 including the reference tap. In addition, the transmitter and receiver filters were designed such that their overall response $\tilde{g}(t)$ was Nyquist. The pulse shape used was the raised-cosine pulse with a roll-off factor of α. The transmitter and receiver transfer functions were both $\sqrt{\tilde{G}(jw)}$.

The Random and Gaussian number generators used, were provided in [29]. In addition, a complex matrix inversion program to invert the Hermitian matrix A, was developed using the real matrix inversion program in [29]. It was tested rigorously and was very stable. In the LMS simulations, each test case was subjected to 60 independent trials, each of length 3220 and the average learning curve was calculated. In addition, the data for each test case was stored in files coded as "123.456". The codes are shown in table A.1.

Therefore, the file eoa.25a contained the data of the average learning curve for 8PSK, λ=0.005, channel A and 25 dB SNR. Also, the data file grx.25 held the raw data for 60 trials for 16QAM, λ=0.05, channel X and 25 dB SNR.

Code	Parameter	Allowed Symbols and Meaning
1	Constellation	e:8PSK; g:16QAM; h:8V29; j:16V29;
2	Step-Size λ	o:0.005; r:0.05
3	Channel	a:Channel A; x:Channel X;
45	SNR in dB	25:25 dB;
6	File Data Type	a:average; nothing:60 trials;

Table A.1: Data File Code Table

A.2 MMSE Program File and Test Case

MMSE.C

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX       256  /* Max number of signals in constellation       */
#define OFFSET    220  /* Position of Reference Response                */
#define MAXTERMS  440  /* Number of terms in Convolution                */
#define MAXTAPS    40  /* Maximum Number of Taps in Equalizer           */
#define Nm          2  /* Used for Print Display                        */
#define PI        3.141592654
#define ERROR     0.00000001

typedef struct FCOMPLEX {
  double r,i;
  } fcomplex;
/*************************************************************************/
/********* M - Number of signals in constellation             ************/
/********* N+1 - Total number of Equalizer Taps               ************/
/********* SIG_SET - Signal points in Constellation           ************/
/********* g - overall impulse response                       ************/
/********* f - receiver impulse response                      ************/
/********* fp - file pointer to output data file              ************/
/********* val - stores step-sizes that will be used in simulation *****/
/********* N0 - noise power to signal energy power            ************/
/*************************************************************************/
  double MINMEANSQERR(M,N,SIG_SET,g,f,fp,val,N0)
  int      M, N;
  fcomplex SIG_SET[MAX+1], g[MAXTERMS+1], f[MAXTERMS+1];
  FILE     *fp;
  float    val[Nm+1], N0;
  {
/*************************************************************************/
/****************** Complex Operators ************************************/
/*************************************************************************/
    double   Cabs();
    fcomplex Cadd(), Csub(), Cmul(), Cdiv();
    fcomplex Complex(), Conjg(), Arg(), RCmul();
/*************************************************************************/
/********** y - received signal, b - input data signal    ***************/
/********** bd - differentially encoded phase              ***************/
/********** A - y correlation matrix, INV - A inverse      ***************/
/********** B - cross - correlation matrix bet. b and y    ***************/
/********** C - equalizer vector                           ***************/
/********** MSE - mean square error, pdt = 1 - MSE         ***************/
/********** dummy - dummy variable                         ***************/
/********** v - sum of z[n-i]*e[j(sum of angles)]          ***************/
```

```
/**********************************************************************/
    fcomplex A[MAXTAPS+1][MAXTAPS+1], INV[MAXTAPS+1][MAXTAPS+1];
    fcomplex I[MAXTAPS+1][MAXTAPS+1];
    fcomplex B[MAXTAPS+1], C[MAXTAPS+1];
    fcomplex pdt;
    doubl   MSE;
/**********************************************************************/
/********** Cd, Cs vectors calculated during ideal gradient algorithm **/
/********** AC - pdt estimate of A matrix and C matrix *****************/
/********** I - Identity matrix, A1 = I - A            *****************/
/**********************************************************************/
    fcomplex Cd[MAXTAPS+1], Cs[MAXTAPS+1];
    fcomplex AC[MAXTAPS+1];
    fcomplex A1[MAXTAPS+1][MAXTAPS+1];
/**********************************************************************/
/********** i,j,k,k1,kk,n - indices, rn - random # generated    *******/
/********** pos - reference point, jNm = j%Nm                   *******/
/********** Diff - mean square difference without equalizer     *******/
/********** Store, Store2 - intermediate differences            *******/
/********** alpha - step-size                                   *******/
/********** cinv - inverts complex matrix                       *******/
/********** ran1 - generates uniformly distributed r.v          *******/
/********** gasdev - generates gaussian distributed r.v         *******/
/********** noise - additive channel noise components at diff. instants*/
/********** w - noise after passing through receiver            *******/
/**********************************************************************/
    int i, j, jNm, k, n;
    int max_num, chk;
    float alpha;
    double   Diff, Store, Store2;
    void cinv();
/**********************************************************************/
/************************* Get Number of Step-Sizes *******************/
/**********************************************************************/
    chk = 0;
    do
    {
      chk++;
    }
    while (val[chk] != 0.0);
    max_num = chk - 1;
/**********************************************************************/
/******************************** Initialize **************************/
/**********************************************************************/
    for (i=0;i<=MAXTAPS;i++)
    {
      B[i] = Complex(0.0,0.0);
      for (j=0;j<=MAXTAPS;j++) A[i][j] = Complex(0.0,0.0);
    }
    for (i=0;i<=MAXTAPS;i++) C[i] = Complex(0.0,0.0);
```

```
   for (i=0;i<MAXTAPS;i++) g[i] = Complex(0.0,0.0);
   for (i=0;i<MAXTAPS;i++) g[MAXTERMS-i] = Complex(0.0,0.0);
/****************************************************************************/
/******************** Get Auto-Correlation Matrix A ***********************/
/****************************************************************************/
   for (i=1;i<=N+1;i++)
   {
     for (j=1;j<=N+1;j++)
     {
if ((i==j) && (i>1))
  A[i][i] = A[1][1];
else
{
  for (k=0;k<=MAXTERMS;k++)
  {
    if ( ((k-i+j)>=0) && ((k-i+j)<=MAXTERMS) )
    A[i][j]  = Cadd(A[i][j],Cmul(Conjg(g[k]),g[k-i+j]));
  }
  if (i==j) A[i][i].r + = 2 * N0;
}
     }
   }
/****************************************************************************/
/******************** Get cross-correlation Vector B    ******************/
/****************************************************************************/
   for (i=1;i<=N+1;i++)
      B[i] = Conjg(g[i+OFFSET-N/2-1]);
   /***** Inverts A to INV, dim N+1, A unchanged ******/
   cinv(A,INV,N+1);
   fflush(fp);
/****************************************************************************/
/************************** Optimum Equalizer ****************************/
/****************************************************************************/
   pdt = Complex(0.0,0.0);
   for (i=1;i<=N+1;i++)
   {
     C[i-1] = Complex(0.0,0.0);
     for (j=1;j<=N+1;j++)
       C[i-1] = Cadd(C[i-1],Cmul(INV[i][j],B[j]));
     pdt = Cadd(pdt,Cmul(Conjg(B[i]),C[i-1]));
   }
   fflush(fp);
/****************************************************************************/
/***************************** check to see AC = B **********************/
/****************************************************************************/
/*
   for (i=1;i<=N+1;i++)
   {
     AC[i] = Complex(0.0,0.0);
     for (j=1;j<=N+1;j++) I[i][j] = Complex(0.0,0.0);
```

```
      I[i][i].r = 1.0;
    }
    for (i=1;i<=N+1;i++)
    {
      for (j=1;j<=N+1;j++) AC[i] = Cadd(AC[i],Cmul(A[i][j],C[j-1]));
      if (i%Nm == 1) fprintf(fp,"\n");
      fprintf(fp,"P[%3d]=%8.4f,%8.4fj   ",i,AC[i].r,AC[i].i);
    }
    fflush(fp);
*/
/******************************************************************************/
/***************** Theoretical Interest - Ideal Grad. Alg. ***************/
/****************** C[i] = C[i](I-alpha x A) - alpha x B *****************/
/******************************************************************************/
/*
    fprintf(fp,"\n\n ***** MEAN SQUARE GRADIENT ALGORITHM ****");

    for (chk=1;chk<=max_num;chk++)
    {
      for (i=0;i<=MAXTAPS;i++) C[i] = Complex(0.0,0.0);
      if (N>0) C[N/2+1].r = 1.0;
      else C[0].r = 1.0;
      alpha = val[chk];
      fprintf(fp,"\nalpha=%8.4f\n",alpha);
      if (alpha != 0.0)
      {
        for (i=1;i<=N+1;i++)
          for (j=1;j<=N+1;j++)
            A1[i][j] = Csub(I[i][j],RCmul(alpha,A[i][j]));
        k=0;
        do
        {
          k++;
          for (i=1;i<=N+1;i++)
          {
            AC[i] = Complex(0.0,0.0);
            Cs[i] = C[i];
            for (j=1;j<=N+1;j++)
              AC[i] = Cadd(AC[i],Cmul(A1[i][j],C[j-1]));
          }
          for (i=1;i<=N+1;i++)
            C[i-1] = Cadd(AC[i],RCmul(alpha,B[i]));
          Diff = 0.0;
          for (i=0;i<=N;i++) Cd[i] = Csub(C[i],Cs[i]);
          for (i=0;i<=N;i++) Diff += Cabs(Cd[i]);
        }
        while (Diff > ERROR);

        v = Complex(0.0,0.0);
        for (i=1;i<=N+1;i++)
        {
          if ((i%Nm ==1) && (i != 1)) fprintf (fp,"\n");
          fprintf(fp,"C[%3d]=%8.4f,%8.4fj    ",i-N/2-1,C[i-1].r,C[i-1].i);
          v = Cadd(v,Cmul(Conjg(B[i]),C[i-1]));
        }
        fprintf(fp,"\nTimes in loop=%3d, pdt1=%8.4f,%8.4fj     ",k,v.r,v.i);
      }
    }
*/
/***************************************************************/
    fprintf(fp,"\nMinimum Mean Square Error \n");
    MSE = 1 - Cabs(pdt);
    fprintf(fp,"PDT=%8.4f+%8.4fj",pdt.r,pdt.i);
    fflush(fp);
    return(MSE);
  }
```

Input File

```
ga25.05
1
0.5
180
0.5
q
16
1
0.05
25.0
2
20
2
1
5
1
gra.25
```

Output File

```
Numberofpaths = 1
Path#1Gc_mod = 0.500, Gc_ang = 180.000, delay = 0.500
Integral = 2.1204
Rollfactor = 0.0000

16 - QAM
Qam_Mag = 1.3416, Preq = 0.2500
Qam_Mag = 1.0000, Preq = 0.5000
Qam_Mag = 0.4472, Preq = 0.2500
sqsum[1] = 1.8889
sqsum[2] = 0.3533
sqsum[3] = 0.1429
sqsum[5] = 0.0482
s[1] = -0.9487, -0.9487j s[2] = -0.3162, -0.9487j
s[3] = 0.3162, -0.9487j s[4] = 0.9487, -0.9487j
s[5] = -0.9487, -0.3162j s[6] = -0.3162, -0.3162j
s[7] = 0.3162, -0.3162j s[8] = 0.9487, -0.3162j
s[9] = -0.9487, 0.3162j s[10] = -0.3162, 0.3162j
s[11] = 0.3162, 0.3162j s[12] = 0.9487, 0.3162j
s[13] = -0.9487, 0.9487j s[14] = -0.3162, 0.9487j
s[15] = 0.3162, 0.9487j s[16] = 0.9487, 0.9487j
N0/2 = 25.0000dB
N0/2 = 0.0032

NumberofEqualiserTapsbesidesC[0] = 0
MinimumMeanSquareError
PDT = 0.7502 + 0.0000j
N = 0, MMSE = 0 2498

L = 1MMSE = 0.2499
L = 2MMSE = 0.2498
L = 3MMSE = 0.2498
L = 5MMSE = 0.2498

NumberofEqualiserTapsbesidesC[0] = 2
MinimumMeanSquareError
PDT = 0.9688 + 0.0000j
N = 2, MMSE = 0.0312

L = 1MMSE = 0.0314
L = 2MMSE = 0.0312
L = 3MMSE = 0.0312
L = 5MMSE = 0.0312

NumberofEqualiserTapsbesidesC[0] = 4
MinimumMeanSquareError
PDT = 0.9718 + 0 0000j
N = 4, MMSE = 0.0284

L = 1MMSE = 0.0285
L = 2MMSE = 0.0284
L = 3MMSE = 0 0284
L = 5MMSE = 0 0284

NumberofEqualiserTapsbesidesC[0] = 6
MinimumMeanSquareError
PDT = 0.9773 + 0.0000j
N = 6, MMSE = 0.0227

L = 1MMSE = 0 0229
L = 2MMSE = 0.0228
L = 3MMSE = 0.0228
L = 5MMSE = 0.0228

NumberofEqualiserTapsbesidesC[0] = 8
MinimumMeanSquareError
PDT = 0.9790 + 0.0000j
N = 8, MMSE = 0.0210

L = 1MMSE = 0.0212
L = 2MMSE = 0.0211
L = 3MMSE = 0.0210
L = 5MMSE = 0.0210

NumberofEqualiserTapsbesidesC[0] = 10
MinimumMeanSquareError
PDT = 0.9804 + -0.0000j
N = 10, MMSE = 0.0196

L = 1MMSE = 0.0198
L = 2MMSE = 0.0196
L = 3MMSE = 0.0196
L = 5MMSE = 0.0196

NumberofEqualiserTapsbesidesC[0] = 12
MinimumMeanSquareError
PDT = 0.9813 + 0.0000j
N = 12, MMSE = 0.0187

L = 1MMSE = 0 0188
L = 2MMSE = 0.0187
L = 3MMSE = 0.0187
L = 5MMSE = 0.0187

NumberofEqualiserTapsbesidesC[0] = 14
MinimumMeanSquareError
PDT = 0.9820 + -0.0000j
N = 14, MMSE = 0.0180

L = 1MMSE = 0.0181
L = 2MMSE = 0.0180
L = 3MMSE = 0.0180
L = 5MMSE = 0.0180

NumberofEqualiserTapsbesidesC[0] = 16
MinimumMeanSquareError
PDT = 0.9826 + 0.0000j
N = 16, MMSE = 0.0174

L = 1MMSE = 0.0176
L = 2MMSE = 0.0174
L = 3MMSE = 0.0174
L = 5MMSE = 0.0174

NumberofEqualiserTapsbesidesC[0] = 18
MinimumMeanSquareError
PDT = 0.9830 + 0.0000j
N = 18, MMSE = 0.0170

L = 1MMSE = 0.0171
L = 2MMSE = 0.0170
L = 3MMSE = 0.0170
L = 5MMSE = 0.0170

NumberofEqualiserTapsbesidesC[0] = 20
MinimumMeanSquareError
PDT = 0.9834 + 0.0000j
N = 20, MMSE = 0.0166

L = 1MMSE = 0.0168
L = 2MMSE = 0.0166
L = 3MMSE = 0.0166
L = 5MMSE = 0.0166
```

Appendix B

B.1 AMSE Program File and Test Case

AMSE.C

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX       256          /* Max Number of signals allowed in a set */
#define MAXTERMS  440          /* Max Number of convolution terms        */
#define OFFSET    220          /* Position of reference tap              */
#define RUN      3220          /* Number of signals in a sequence        */
#define MAXTAPS    40          /* Max Number of Complex Taps Allowed     */
#define Nm          2          /* Used for Printed Output                */
#define KKMAX      60          /* Max number of Runs                     */
#dafine PI        3.141592654
#define ERROR     0.000000001
#define CELLSIZE  500
#define MAXCELL    20
#define STEP      100

typedef struct FCOMPLEX {
  double r,i;
  } fcomplex;
/*******************************************************************/
/****************** Uses LMS adaptive algorithm to     *************/
/****************** update equalizer coefficients      *************/
/****************** Parameters just like in MINMEANSQERROR *********/
/*******************************************************************/
  ADAPTIVEMSE(L,M,N,SIG_SET,g,f,fp,f1,f2,val,N0)
  int      L, M, N;
  fcomplex SIG_SET[MAX+1], g[MAXTERMS+1], f[MAXTERMS+1];
  FILE     *fp, *f1;
  float    val[Nm+1], N0;
  {
    double   Cabs();
    fcomplex Cadd(), Csub(), Cmul(), Cdiv();
    fcomplex Complex(), Conjg(), Arg(), RCmul();
    fcomplex y[RUN+1], b[RUN+1], bd[RUN+1];
    fcomplex C[MAXTAPS+1];
/*************************************************************/
/*************** z - equalizer outputs           *************/
/*************** temp - used for phase estimation *************/
/*************** estimate - estimate of data signal *************/
/*************** error = estimate - actual        *************/
/*************** fac - dummy variable for updating *************/
/*************************************************************/
    fcomplex z[RUN+1], b_arg[RUN+1], temp[MAXTAPS+1];
    fcomplex estimate, error, fac[MAXTAPS+1];
    fcomplex v, v_arg, noise[RUN+1], w[RUN+1];
    float    alpha, stdv, ran1();
    double   sum, sum1, Amse[RUN+1];
    double   sum2[MAXCELL+1], sum3[MAXCELL+1], amse;
    float    gasdev(), cnt[MAXTAPS+1];
/***************************************************/
/************** Indices          *******************/
/***************************************************/
    int i, j, jNn, k, k1, kk, rn, n;
    int idum, idum2, max_num, chk;
/********************************************************/
/**************** Determine Number of Step-sizes ***********/
/********************************************************/
    chk = 0;
    do
    {
      chk++;
    }
    while (val[chk] != 0.0);
    max_num = chk - 1;
    stdv = sqrt(NO);
/********************************************************/
    for (chk=1;chk<=max_num;chk++)
    {
      alpha = val[chk];
      if (alpha != 0.0)
      {
fprintf(fp,"\n Step = %6.3f\n",alpha);
for (k=0;k<=RUN;k++) bd[k] = Complex(0.0,0.0);
for (i=0;i<=RUN;i++) Amse[i] = 0.00;
bd[0].r = 1.0; amse = 0.0;
for (i=0;i<=MAXCELL;i++) sum3[i] = 0.0;
/********************************************************/
/********************************************************/
for (kk=1;kk<=KKMAX;kk++)
{
  for (i=0;i<=MAXTAPS;i++) C[i] = Complex(0.0,0.0);
```

```
/******* Different Equalizer Initialization *************/
/*   if (N>0) C[N/2].r = 1.0;
  else C[0].r = 1.0;
*/
/*********************************************************/
  sum = 0.0;
  sum1 = 0.0;
  for (i=0;i<=MAXCELL;i++) sum2[i] = 0.0;
  for (i=1;i<=M;i++) cnt[i] =0.0;

  idum =  kk;
  idum2 = kk;
  fprintf(fp,"\n %2d, ",kk);
  for (k=1;k<= RUN;k++)
  {
    rn = ((int)(ran1(&idum)*M))%M +1;
    for (i=1;i<=M;i++)
    {
      if (rn == i) cnt[i]++;
    }
    /*** idum = k ******/
    b[k] = SIG_SET[rn];
    b_arg[k] = Arg(b[k]);
    bd[k] = Cmul(b_arg[k],bd[k-1]);
    noise[k] = Complex(gasdev(&idum2),gasdev(&idum2));
    noise[k].r *= stdv ;
    noise[k].i *= stdv ;
  }
  fprintf(fp,"\n");
  for (i=1;i<=M;i++)
  {
    cnt[i] /= RUN;
    fprintf(fp,"%8.4f",cnt[i]);
  }
  fflush(fp);

  for (n=1;n<= RUN;n++)
  {
    y[n] = Complex(0.0,0.0);
    w[n] = Complex(0.0,0.0);
    for (k=0;k<=MAXTERMS;k++)
    {
      k1 = k - OFFSET;
      if ((n >k1) && (k1 >= n - RUN))
      {
               y[n] = Cadd(y[n],Cmul(bd[n-k1-1],Cmul(b[n-k1],g[k])));
               w[n] = Cadd(w[n],Cmul(noise[n-k1],f[k]));
      }
    }
    y[n] = Cadd(y[n],w[n]);
```

```
     }
      /*************** LMS ADAPTIVE ALGORITHM ******* */
n = 1;
do
{
  /**** Get Phase encoded estimate for prev. signal ***/
  z[n] = Complex(0.0,0.0);
  sum = 0.0;
  for (i=1;i<=L;i++) temp[i] = Complex(0.0,0.0);
  for (i=0;i<=N;i++)
  {
    if ((n+i)>N/2) z[n] = Cadd(z[n],Cmul(C[i],y[n+i-N/2]));
  }
  for (i=1;i<=L;i++)
  {
    if (n>i) temp[i] = z[n-i];
  }
  for (i=2;i<=L;i++)
  {
    for (j=n-i+1;j<=n-1;j++)
    {
              if (j) temp[i] = Cmul(temp[i],b_arg[j]);
    }
  }
  v = Complex(0.0,0.0);
  for (i=1;i<=L;i++) v = Cadd(v,temp[i]);
  if (Cabs(v) == 0.0) v = Complex(1.0,0.0);
  v_arg    = Arg(v);
  estimate = Cmul(z[n],Conjg(v_arg));
  error    = Csub(estimate,b[n]);
  sum      = Cabs(error) * Cabs(error);
  for (i=0;i<=N;i++)
  {
    fac[i] = Complex(0.0,0.0);
    if ((n+i)>N/2)
    {
             fac[i] = Cmul(Conjg(y[n+i-N/2]),v_arg);
             fac[i] = Cmul(fac[i],error);
    }
    C[i] = Csub(C[i],RCmul(alpha,fac[i]));
  }
  if ((n%STEP == 0) || (n ==1))
  {
    if (n<=RUN-OFFSET) fprintf(f1,"%4d %8.4f\n",n,sum);
    fflush(f1);
  }
  Amse[n] += sum;
  sum1    += sum;

              if ((n>   0)&&(n<= 500)) sum2[1] += sum;
    else if ((n> 500)&&(n<=1000)) sum2[2] += sum;
    else if ((n>1000)&&(n<=1500)) sum2[3] += sum;
    else if ((n>1500)&&(n<=2000)) sum2[4] += sum;
    else if ((n>2000)&&(n<=2500)) sum2[5] += sum;
    else if ((n>2500)&&(n<=3000)) sum2[6] += sum;
    n++;
  }
  while (n<=RUN-OFFSET);

  sum1 /= (RUN - OFFSET);
  amse  += sum1;
  for (i=0;i<=MAXCELL;i++) sum2[i] /= CELLSIZE ;
  fprintf(fp,"\n");
  for (i=1;i<=6;i++) fprintf(fp,"%8.4f",sum2[i]);
  fflush(fp);
  for (i=0;i<=MAXCELL;i++) sum3[i] += sum2[i] ;
  fflush(f1);
  fprintf(f1,"\n");
}
for (n=1;n<=RUN;n++) Amse[n] /= KKMAX;
        fprintf(f2,"   1 %8.4f\n",Amse[1]);
fflush(f2);
for (n=1;n<=RUN;n++)
{
  if ((n%STEP == 0) && (n<=RUN-OFFSET))
  {
    if (n<RUN-OFFSET) fprintf(f2,"%4d %8.4f\n",n,Amse[n]);
    else fprintf(f2,"%4d %8.4f",n,Amse[n]);
    fflush(f2);
  }
}
amse /= KKMAX;
fprintf(fp,"\nStep Size=%8.4f,\tAverage MSE = %8.4f\n",alpha,amse);
fprintf(fp,"%4d\n",KKMAX);
for (i=0;i<=MAXCELL;i++) sum3[i] /= KKMAX ;
for (i=1;i<=6;i++) fprintf(fp,"%10.4f",sum3[i]);
fprintf(fp,"\n");
fflush(fp);
      }
    }
  }
```

Input File

```
gx25.05
2
0.3
180
0.5
0.2
90
3.5
4
16
1
0.05
25.0
2
20
2
1
5
1
grx.25
```

Average Output File

```
   1   1.1200
 100    0.0438
 200    0.0286
 300    0.0312
 400    0.0216
 500    0.0246
 600    0.0312
 700    0.0242
 800    0.0301
 900    0.0247
1000    0 0287
1100    0.0254
1200    0.0286
1300    0.0317
1400    0.0282
1500    0.0225
1600    0.0209
1700    0.0270
1800    0.0328
1900    0.0291
2000    0.0239
2100    0.0290
2200    0.0249
2300    0.0256
2400    0.0264
2500    0.0287
2600    0.0339
2700    0.0334
2800    0.0252
2900    0.0317
3000    0.0278
```

Raw Output File

```
    1  1.8000        1  1.8000        1  1.8000        1  1.0000        1  1.8000        1  1.0000
  100  0.0303      100  0.0247      100  0.0129      100  0.0474      100  0.1145      100  0.0049
  200  0.0163      200  0.0102      200  0.0409      200  0.0490      200  0.0349      200  0.0196
  300  0.0326      300  0.0472      300  0.0151      300  0.0336      300  0.0043      300  0.0266
  400  0.0416      400  0.0015      400  0.0165      400  0.0001      400  0.0022      400  0.0098
  500  0.0780      500  0.0059      500  0.0044      500  0.0038      500  0.0107      500  0.0075
  600  0.0322      600  0.0385      600  0.0107      600  0.0343      600  0.0215      600  0.0366
  700  0.1259      700  0.0515      700  0.0262      700  0.0123      700  0.0315      700  0 0140
  800  0.0115      800  0.0092      800  0.0102      800  0.0092      800  0.0111      800  0 0078
  900  0.0126      900  0.0372      900  0.0418      900  0.0815      900  0.0338      900  0 0055
 1000  0.0045     1000  0.0034     1000  0.0355     1000  0.0401     1000  0.0874     1000  0 0652
 1100  0.0188     1100  0.0025     1100  0.0023     1130  0.0264     1100  0.0212     1100  0.0549
 1200  0.0049     1200  0.0214     1200  0.0192     1200  0.0106     1200  0.0545     1200  0.0325
 1300  0.0393     1300  0.0114     1300  0.0181     1300  0.0029     1300  0.0001     1300  0.0183
 1400  0.0982     1400  0.0149     1400  0.0156     1400  0.0024     1400  0 0074     1400  0.0455
 1500  0.0084     1500  0.0054     1500  0.0302     1500  0.0223     1500  0.0269     1500  0 0031
 1600  0.0016     1600  0.0080     1600  0.0040     1600  0.0019     1600  0.0203     1600  0 0088
 1700  0.0105     1700  0.0019     1700  0.0576     1700  0.0325     1700  0.0914     1700  0 0154
 1800  0.0010     1800  0.0259     1800  0.0307     1800  0.0291     1800  0.0525     1800  0.0232
 1900  0.0097     1900  0.0119     1900  0.0122     1900  0.0788     1960  0.0007     1900  0.0111
 2000  0.0068     2000  0.0297     2000  0.0256     2000  0.0013     2000  0.0034     2000  0.0313
 2100  0.0085     2100  0.0007     2100  0.0123     2100  0.0088     2100  0.0192     2100  0 1019
 2200  0.0030     2200  0.0894     2200  0.0003     2200  0.0264     2200  0.0185     2200  0 0098
 2300  0.0047     2300  0.0130     2300  0.0270     2300  0.0136     2300  0.0701     2300  0 0262
 2400  0.0117     2400  0.0428     2400  0 0218     2400  0.0001     2400  0.0357     2400  0 0564
 2500  0.0222     2500  0.0165     2500  0.0432     2500  0.0012     2500  0 0593     2500  0 0688
 2600  0.0011     2600  0.0312     2600  0.0996     2600  0.0279     2600  0.0379     2600  0 0154
 2700  0.1296     2700  0.0114     2700  0.0023     2700  0.0044     2700  0.0469     2700  u 0272
 2800  0 0093     2800  0 0242     2800  0.0101     2800  0.0095     2800  0 0194     2800  0.0544
 2900  0.0384     2900  0.0313     2900  0 0064     2900  0 0255     2900  0.0297     2900  0 0088
 3000  0.0395     3000  0.0914     3000  0.0199     3000  0.0125     3000  0 0508     3000  0.0466

    1  0.2000        1  1.8000        1  1.0000        1  0.2000        1  1 0000        1  1 8000
  100  0.0073      100  0.0667      100  0.0056      100  0.0178      100  0.0488      100  0 0572
  200  0.0009      200  0 0209      200  0 0022      200  0.0204      200  0 0371      200  0.0171
  300  0.0293      300  0.0014      300  0.0167      300  0.0576      300  0.1076      300  0.0669
  400  0.0520      400  0.0264      400  0.0296      400  0.0198      400  0 0691      400  0.0277
  500  0.0179      500  0.0426      500  0.0182      500  0.0196      500  0.0277      500  0 0166
  600  0.0187      600  0.0220      60u  0.0042      600  0.0554      600  0.0346      600  0 0029
  700  0.0016      700  0.0243      700  0.0013      700  0.0127      700  0 0026      700  0.0411
  800  0.0092      800  0.0046      800  0.0041      800  0.0132      800  0 0286      800  0.0066
  900  0 0075      900  0.0157      900  0.0231      900  0.0460      900  0.0394      900  0 0122
 1000  0.0033     1000  0.0451     1000  0.0377     1000  0.0707     1000  0.0207     1000  0 0320
 1100  0.0044     1100  0.0571     1100  0.0028     1100  0.0032     1100  0.0496     1100  0 0067
 1200  0.0394     1200  0.0381     1200  0.0109     1200  0.0406     1200  0.0117     1200  0 0688
 1300  0.0090     1300  0.0116     1300  0 0212     1300  0.0226     1300  0.0715     1300  0 0263
 1400  0.1023     1400  0.0210     1400  0.0246     1400  0.0018     1400  0.0032     1400  0 0057
 1500  0.0172     1500  0.0241     1500  0.0120     1500  0.0802     1500  0.0443     1500  0.0241
 1600  0.0026     1600  0.0171     1600  0.0112     1600  0.0133     1600  0.0230     1600  0 0120
 1700  0.0183     1700  0.0741     1700  0.0033     1700  0.0218     1700  0.0060     1700  0 0147
 1800  0.0681     1800  0.0136     1800  0.0108     1800  0.0058     1800  0.0210     1800  0 0054
 1900  0.0581     1900  0.0188     1900  0.0085     1900  0.0310     1900  0.0264     1900  0 0094
 2000  0.0814     2000  0.0174     2000  0.0570     2000  0.0184     2000  0.1079     2000  0 0002
 2100  0.0457     2100  0.0137     2100  0.0106     2100  0.02[illegible]     2100  0 0121     2100  0 0099
 2200  0.0006     2200  0.0242     2200  0.0407     2200  [illegible]     2200  0.0066     2200  0 0340
 2300  0.0297     2300  0.0273     2300  0.0040     2300  0.021[illegible]     2300  0 0228     2300  0 0029
 2400  0.0079     2400  0.0171     2400  0 0033     2400  0.0192     2400  0.0147     2400  0.0162
 2500  0.0074     2500  0.0167     2500  0.0079     2500  0.0033     2500  0 0841     2500  0 0006
 2600  0.0329     2600  0.0380     2600  0.0210     2600  0.0276     2600  0 0252     2600  0 0247
 2700  0.0124     2700  0.0462     2700  0.0095     2700  0.0062     2700  0.0030     2700  0.0009
 2800  0.1140     2800  0.0165     2800  0.0184     2800  0.0204     2800  0.0023     2800  0.0373
 2900  0.0210     2900  0.0134     2900  0.0259     2900  0.1068     2900  0.0083     2900  0.0214
 3000  0.0249     3000  0 0117     3000  0.0092     3000  0.0141     3000  0.0228     3000  0.0263
```

1	1.0000	1	1.0000	1	1.0000	1	0.2000	1	1.0000	1	1.0000
100	0.0524	100	0.0493	100	0.0470	100	0.0292	100	0.0344	100	0.0608
200	0.0779	200	0.0073	200	0.0541	200	0.0079	200	0.0039	200	0.0230
300	0.0244	300	0.0013	300	0.0099	300	0.0275	300	0.0282	300	0.0109
400	0.0458	400	0.0329	400	0.0184	400	0.0136	400	0.0164	400	0.0129
500	0.0202	500	0.0110	500	0.0325	500	0.0270	500	0.0136	500	0.0067
600	0.0908	600	0.0513	600	0.0176	600	0.0391	600	0.1132	600	0.0591
700	0.0260	700	0.0062	700	0.0004	700	0.0275	700	0.0503	700	0.0049
800	0.0005	800	0.0058	800	0.0078	800	0.1088	800	0.0319	800	0.0203
900	0.0181	900	0.0060	900	0.0036	900	0.0050	900	0.0211	900	0.0029
1000	0.0720	1000	0.0491	1000	0.0059	1000	0.0059	1000	0.0106	1000	0.0079
1100	0.0062	1100	0.0220	1100	0.0131	1100	0.0244	1100	0.0265	1100	0.0350
1200	0.0024	1200	0.0016	1200	0.0104	1200	0.0183	1200	0.0150	1200	0.0024
1300	0.0085	1300	0.1041	1300	0.0176	1300	0.0056	1300	0.0109	1300	0.0375
1400	0.0206	1400	0.0378	1400	0.0008	1400	0.0534	1400	0.0715	1400	0.0334
1500	0.0069	1500	0.0009	1500	0.0109	1500	0.0129	1500	0.0279	1500	0.0177
1600	0.0158	1600	0.0233	1600	0.0001	1600	0.0069	1600	0.0083	1600	0.0324
1700	0.0123	1700	0.0071	1700	0.0006	1700	0.0005	1700	0.0022	1700	0.0044
1800	0.1196	1800	0.0058	1800	0.0041	1800	0.0219	1800	0.0577	1800	0.0206
1900	0.0005	1900	0.0074	1900	0.0370	1900	0.0322	1900	0.0010	1900	0.0054
2000	0.0310	2000	0.0694	2000	0.0029	2000	0.0000	2000	0.0149	2000	0.0272
2100	0.0053	2100	0.1063	2100	0.0505	2100	0.0256	2100	0.0145	2100	0.0018
2200	0.0325	2200	0.0210	2200	0.0168	2200	0.0088	2200	0.0033	2200	0.0231
2300	0.0220	2300	0.0044	2300	0.0028	2300	0.0210	2300	0.0079	2300	0.0006
2400	0.1246	2400	0.0110	2400	0.0566	2400	0.0145	2400	0.0353	2400	0.0003
2500	0.0181	2500	0.0537	2500	0.1058	2500	0.0288	2500	0.0024	2500	0.0165
2600	0.0073	2600	0.0088	2600	0.0350	2600	0.0052	2600	0.0115	2600	0.0484
2700	0.0303	2700	0.0231	2700	0.0093	2700	0.0036	2700	0.0061	2700	0.0522
2800	0.0026	2800	0.0114	2800	0.0270	2800	0.0253	2800	0.0585	2800	0.0047
2900	0.0227	2900	0.0241	2900	0.0077	2900	0.0113	2900	0.0142	2900	0.0014
3000	0.0198	3000	0.0210	3000	0.0058	3000	0.0016	3000	0.0145	3000	0.0133

1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	0.2000	1	0.2000
100	0.0145	100	0.0180	100	0.0057	100	0.0796	100	0.0412	100	0.0574
200	0.0296	200	0.0898	200	0.0148	200	0.0032	200	0.0334	200	0.1065
300	0.0447	300	0.0054	300	0.0782	300	0.0067	300	0.0205	300	0.0081
400	0.0065	400	0.0043	400	0.0089	400	0.0198	400	0.0099	400	0.0170
500	0.0114	500	0.0368	500	0.0306	500	0.0411	500	0.0602	500	0.0445
600	0.0066	600	0.1163	600	0.0093	600	0.0410	600	0.0123	600	0.1021
700	0.0180	700	0.0278	700	0.0045	700	0.0074	700	0.0057	700	0.0294
800	0.0032	800	0.0140	800	0.0116	800	0.0039	800	0.0460	800	0.0097
900	0.0356	900	0.0471	900	0.0062	900	0.0010	900	0.0025	900	0.0081
1000	0.0132	1000	0.0981	1000	0.0055	1000	0.0131	1000	0.0964	1000	0.0181
1100	0.0137	1100	0.0460	1100	0.0211	1100	0.0475	1100	0.0210	1100	0.0071
1200	0.0010	1200	0.0483	1200	0.0013	1200	0.0166	1200	0.0328	1200	0.0144
1300	0.0455	1300	0.0358	1300	0.0394	1300	0.0227	1300	0.1309	1300	0.0682
1400	0.0188	1400	0.0163	1400	[illegible]	1400	0.0029	1400	0.0113	1400	0.0089
1500	0.0077	1500	0.0175	1500	[illegible]	1500	0.0073	1500	0.0491	1500	0.0597
1600	0.0358	1600	0.0154	1600	[illegible]	1600	0.0607	1600	0.0370	1600	[illegible]
1700	0.0259	1700	0.0179	1700	0.017	1700	0.0199	1700	0.0326	1700	[illegible]
1800	0.0204	1800	0.0884	1800	0.0033	1800	0.0091	1800	0.0322	1800	[illegible]
1900	0.0133	1900	0.0662	1900	0.0133	1900	0.0017	1900	0.0006	1900	0.0121
2000	0.0475	2000	0.0223	2000	0.0932	2000	0.0163	2000	0.0311	2000	0.0087
2100	0.1563	2100	0.0011	2100	0.0102	2100	0.0072	2100	0.0253	2100	0.0304
2200	0.0084	2200	0.0248	2200	0.0018	2200	0.0203	2200	0.0196	2200	0.0833
2300	0.0097	2300	0.0019	2300	0.0198	2300	0.0138	2300	0.0046	2300	0.0455
2400	0.0278	2400	0.0069	2400	0.0095	2400	0.0195	2400	0.0419	2400	0.0093
2500	0.0306	2500	0.0036	2500	0.0242	2500	0.0205	2500	0.0399	2500	0.0870
2600	0.0263	2600	0.0084	2600	0.0341	2600	0.0015	2600	0.0248	2600	0.0085
2700	0.0094	2700	0.0127	2700	0.0205	2700	0.1019	2700	0.0035	2700	0.0110
2800	0.0076	2800	0.0207	2800	0.0300	2800	0.0025	2800	0.0130	2800	0.0666
2900	0.0116	2900	0.0511	2900	0.0923	2900	0.0022	2900	0.0448	2900	0.1035
3000	0.0344	3000	0.0185	3000	0.0031	3000	0.0229	3000	0.0199	3000	0.0062

1	1.0000	1	1.0000	1	0.2000	1	1.0000	1	1.0000	1	0.2000
100	0.0656	100	0.1348	100	0.0123	100	0.1548	100	0.1764	100	0.0101
200	0.0867	200	0.0149	200	0.0057	200	0.0079	200	0.0281	200	0.0490
300	0.0094	300	0.0139	300	0.0081	300	0.0332	300	0.0187	300	0.0037
400	0.0544	400	0.0102	400	0.0033	400	0.0589	400	0.0011	400	0.0058
500	0.0734	500	0.0102	500	0.0145	500	0.0304	500	0.0040	500	0.0058
600	0.1085	600	0.0252	600	0.0055	600	0.0220	600	0.0306	600	0.0523
700	0.0194	700	0.0302	700	0.1140	700	0.0003	700	0.0153	700	0.0026
800	0.0146	800	0.0166	600	0.0093	800	0.1623	800	0.0114	800	0.0358
900	0.0000	900	0.0009	900	0.0396	900	0.0128	900	0.0375	900	0.0705
1000	0.0253	1000	0.0030	1000	0.0764	1000	0.0624	1000	0.0638	1000	0.0065
1100	0.0389	1100	0.0081	1100	0.0220	1100	0.0493	1100	0.0063	1100	0.0330
1200	0.0477	1200	0.0316	1200	0.0672	1200	0.1692	1200	0.0453	1200	0.0104
1300	0.0186	1300	0.0052	1300	0.0128	1300	0.0318	1300	0.0031	1300	0.0108
1400	0.0348	1400	0.0266	1400	0.0582	1400	0.0515	1400	0.0396	1400	0.0289
1500	0.0056	1500	0.0298	1500	0.0269	1500	0.0149	1500	0.0592	1500	0.0054
1600	0.0000	1600	0.0393	1600	0.0091	1600	0.0036	1600	0.0776	1600	0.0527
1700	0.0313	1700	0.0621	1700	0.0065	1700	0.0676	1700	0.0184	1700	0.0205
1800	0.0236	1800	0.0117	1800	0.0084	1800	0.0459	1800	0.0607	1800	0.0442
1900	0.0113	1900	0.0116	1900	0.0389	1900	0.0120	1900	0.0097	1900	0.0663
2000	0.0489	2000	0.0054	2000	0.0853	2000	0.0328	2000	0.0106	2000	0.0008
2100	0.0033	2100	0.0315	2100	0.0204	2100	0.0366	2100	0.0286	2100	0.0369
2200	0.0010	2200	0.0061	2200	0.0423	2200	0.0107	2200	0.0256	2200	0.0238
2300	0.0067	2300	0.0248	2300	0.0234	2300	0.0246	2300	0.0471	2300	0.1146
2400	0.0054	2400	0.0028	2400	0.0970	2400	0.0085	2400	0.0287	2400	0.0024
2500	0.0066	2500	0.0025	2500	0.0080	2500	0.0039	2500	0.0075	2500	0.0715
2600	0.1228	2600	0.0370	2600	0.0417	2600	0.0609	2600	0.0693	2600	0.0150
2700	0.0319	2700	0.0110	2700	0.0560	2700	0.0547	2700	0.0602	2700	0.0131
2800	0.0295	2800	0.0966	2800	0.0148	2800	0.0473	2800	0.0160	2800	0.0533
2900	0.3148	2900	0.0036	2900	0.0413	2900	0.0710	2900	0.0325	2900	0.0113
3000	0.0195	3000	0.0284	3000	0.0388	3000	0.0470	3000	0.0020	3000	0.0373

1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	0.2000
100	0.0188	100	0.1438	100	0.1163	100	0.0257	100	0.0567	100	0.0035
200	0.0098	200	0.0302	200	0.0556	200	0.1327	200	0.0031	200	0.0213
300	0.0536	300	0.0123	300	0.0630	300	0.0650	300	0.0613	300	0.0097
400	0.0050	400	0.0010	400	0.0033	400	0.0432	400	0.0244	400	0.0099
500	0.0078	500	0.0029	500	0.0147	500	0.0441	500	0.0208	500	0.0320
600	0.0089	600	0.0164	600	0.0043	600	0.0382	600	0.0096	600	0 0014
700	0.0248	700	0.0173	700	0.0118	700	0.0101	700	0.0305	700	0.0067
800	0.0850	800	0 0198	800	0.1955	800	0.0195	800	0.1032	800	0 0028
900	0.0133	900	0.0025	900	0.0372	900	0.0218	900	0.0340	900	0.0487
1000	0.0241	1000	0.0755	1000	0.0078	1000	0.0386	1000	0.0209	1000	0.0110
1100	0.0168	1100	0.0036	1100	0.0277	1100	0.0015	1100	0.0071	1100	0.0033
1200	0.0014	1200	0.0166	1200	0.0162	1200	0.0339	1200	0.0316	1200	0.0301
1300	0.0051	1300	0.0055	1300	0.0675	1300	0.0406	1300	0.0438	1300	0.0315
1400	0.0781	1400	0.0284	1400	0.0469	1400	0.0491	1400	0.0289	1400	0.0502
1500	0.0432	1500	0.0014	1500	0.0033	1500	0.0055	1500	0.0640	1500	0.0050
1600	0.0158	1600	0.0124	1600	0.0006	1600	0.0037	1600	0.0363	1600	0.0126
1700	0.0195	1700	0.0370	1700	0.0156	1700	0.0445	1700	0.1398	1700	0.0077
1800	0.0025	1800	0.0233	1800	0.0078	1800	0.0629	1800	0.1835	1800	0.0160
1900	0.0670	1900	0.0568	1900	0.0113	1900	0.0079	1900	0.0111	1900	0.0018
2000	0.0085	2000	0.0018	2000	0.0162	2000	0.0273	2000	0.0322	2000	0.0052
2100	0.0535	2100	0.0068	2100	0.0087	2100	0.0580	2100	0.0591	2100	0 0145
2200	0.0053	2200	0.0054	2200	0.0278	2200	0.0532	2200	0.0057	2200	0.1289
2300	0.0016	2300	0.0983	2300	0.0089	2300	0.0327	2300	0.0291	2300	0.0050
2400	0.0592	2400	0.0003	2400	0.0270	2400	0.0024	2400	0.0314	2400	0.0003
2500	0.0021	2500	0.0011	2500	0.0286	2500	0.0231	2500	0.0046	2500	0.0867
2600	0.0390	2600	0.0047	2600	0.0811	2600	0.0070	2600	0.0692	2600	0.0671
2700	0.0364	2700	0.0553	2700	0.1781	2700	0.0490	2700	0.0013	2700	0.0116
2800	0.0537	2800	0.0482	2800	0.0038	2800	0.0117	2800	0.0141	2800	0.0122
2900	0.0004	2900	0.0805	2900	0.0060	2900	0.0934	2900	0.0358	2900	0.0091
3000	0.0071	3000	0.0393	3000	0.0110	3000	0.0093	3000	0.0143	3000	0.0076

1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	1.0000
100	0.0327	100	0.0780	100	0.0213	100	0.0223	100	0.0570	100	0.0208
200	0.0113	200	0.0073	200	0.0981	200	0.0208	200	0.0374	200	0.0034
300	0.0648	300	0.0183	300	0.0251	300	0.0191	300	0.0286	300	0.0214
400	0.0266	400	0.0152	400	0.0052	400	0.0457	400	0 0557	400	0.0009
500	0.0078	500	0.0941	500	0.0039	500	0.0199	500	0.0751	500	0.0276
600	0.0034	600	0.0654	600	0.0266	600	0.0289	600	0.0181	600	0.0015
700	0.0093	700	0.0031	700	0.0045	700	0.0045	700	0 0075	700	0.0380
800	0.0268	800	0.0699	800	0.0310	800	0.0756	800	0 0168	800	0.0519
900	0.0092	900	0.0015	900	0.0477	900	0.0073	900	0.0396	900	0.0230
1000	0.1265	1000	0.0576	1000	0.0242	1000	0.0433	1000	0.0113	1000	0.0004
1100	0.0638	1100	0.0096	1100	0.0360	1100	0.0036	1100	0.0296	1100	0.0199
1200	0.0207	1200	0.0028	1200	0.0166	1200	0.0010	1200	0.0620	1200	0 0808
1300	0.0477	1300	0.0548	1300	0.0165	1300	0.0512	1300	0.0274	1300	0.0380
1400	0.0324	1400	0.0270	1400	0.0551	1400	0.0169	1400	0.0447	1400	0.0284
1500	0.0042	1500	0.0191	1500	0.0105	1500	0.0223	1500	0.0374	1500	0.0161
1600	0.0090	1600	0.0259	1600	0.0091	1600	0.0010	1600	0.0101	1600	0.0000
1700	0.0020	1700	0.0066	1700	0.0120	1700	0.0153	1700	0.0089	1700	0.0768
1800	0.0098	1800	0.0047	1800	0.0747	1800	0.0384	1800	0 0081	1800	0.0770
1900	0.0358	1900	0.0023	1900	0.0165	1900	0.0831	1900	0.0382	1900	0.0366
2000	0.0185	2000	0.0106	2000	0.0310	2000	0.0002	2000	0.0228	2000	0.0223
2100	0.0351	2100	0.0015	2100	0.0079	2100	0.0027	2100	0 0959	2100	0.0097
2200	0.0079	2200	0.0206	2200	0.0058	2200	0.0340	2200	0.0022	2200	0 0210
2300	0.0287	2300	0.0148	2300	0.0097	2300	0.0533	2300	0 0072	2300	0.0238
2400	0.0086	2400	0.0114	2400	0.0043	2400	0.1111	2400	0 0136	2400	0.0130
2500	0.1572	2500	0.0052	2500	0 0227	2500	0.0175	2500	0.0010	2500	0 0895
2600	0.0128	2600	0.0755	2600	0.0173	2600	0.0424	2600	0.0263	2600	0 0239
2700	0.0098	2700	0.0018	2700	0.0348	2700	0.0793	2700	0.0028	2700	0.0556
2800	0.0229	2800	0.0218	2800	0.0016	2800	0.0153	2800	0.0293	2800	0 0007
2900	0.0205	2900	0.0058	2900	0.0226	2900	0.0024	2900	0.0193	2900	0.0010
3000	0.0287	3000	0.0968	3000	0.0567	3000	0.0035	3000	0.0359	3000	0 1046

1	1.0000	1	1.0000	1	1.0000	1	0.2000	1	1.0000	1	1.0000
100	0.0092	100	0.0507	100	0.0413	100	0.0277	100	0.0001	100	0.1139
200	0.0212	200	0.0011	200	0.0083	200	0.0466	200	0.0019	200	0.0066
300	0.0144	300	0.0029	300	0.1228	300	0.0103	300	0.0290	300	0.0351
400	0.0300	400	0.0034	400	0.0104	400	0.0064	400	0.0312	400	0.0058
500	0.0332	500	0.0044	500	0.0057	500	0.0190	500	0.0159	500	0.0488
600	0.0080	600	0.0061	600	0.0092	600	0.1218	600	0.0117	600	0.0729
700	0.0293	700	0.0214	700	0.0764	700	0.0336	700	0.0427	700	0 0035
800	0.0167	800	0.0703	800	0.0067	800	0.0180	800	0.0010	800	0 0461
900	0.0069	900	0.0268	900	0.0127	900	0.0071	900	0.0349	900	0.0191
1000	0.0120	1000	0.0047	1000	0.0010	1000	0.0106	1000	0.0002	1000	0.0081
1100	0.0090	1100	0.0022	1100	0.0262	1100	0.0118	1100	0 1086	1100	0 0048
1200	0.0237	1200	0.0047	1200	0.0061	1200	0.0312	1200	0.0157	1200	0.0263
1300	0.0428	1300	0.0138	1300	0.0010	1300	0.0397	1300	0 0008	1300	0.0080
1400	0.0881	1400	0.0041	1400	0.0156	1400	0.0041	1400	0.0013	1400	0 0123
1500	0.0140	1500	0.0383	1500	0.0069	1500	0.0089	1500	0.0221	1500	0 0036
1600	0.0316	1600	0 0155	1600	0.0322	1600	0.0153	1600	0.0003	1600	0 0153
1700	0.0011	1700	0.0044	1700	0.0529	1700	0.0186	1700	0.0015	1700	0.0094
1800	0.0036	1800	0.0589	1800	0.0027	1800	0.0057	1800	0.0526	1800	0.0319
1900	0.0024	1900	0.0068	1900	0.0089	1900	0 0233	1900	0.0220	1900	0.0305
2000	0.0195	2000	0.0299	2000	0.0083	2000	0.0155	2000	0.0184	2000	0 0035
2100	0.0211	2100	0.0332	2100	0.0172	2100	0.0137	2100	0.0169	2100	0 0220
2200	0.0125	2200	0.0235	2200	0.0116	2200	0.0163	2200	0 0227	2200	0 0077
2300	0.0601	2300	0.0380	2300	0.0039	2300	0.0135	2300	0 0163	2300	0.0303
2400	0.0969	2400	0.0173	2400	0.0964	2400	0.0203	2400	0.0428	2400	0 0059
2500	0.0039	2500	0.0316	2500	0.0818	2500	0.0035	2500	0.0230	2500	0 0189
2600	0.0263	2600	0.0250	2600	0.0418	2600	0.0033	2600	0.0041	2600	0 0494
2700	0.0156	2700	0.0603	2700	0.0596	2700	0.0039	2700	0 0020	2700	0.0432
2800	0.0120	2800	0.0564	2800	0.0033	2800	0.0166	2800	0.0225	2800	0 0001
2900	0.0171	2900	0.0131	2900	0.0617	2900	0.0306	2900	0.0376	2900	0 0113
3000	0.0381	3000	0.0434	3000	0.0323	3000	0.0436	3000	0.0182	3000	0 0069

1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	1.0000	1	1.0000
100	0.0069	100	0.0008	100	0.0089	100	0.0237	100	0.0231	100	0 0555
200	0.0078	200	0.0116	200	0.0567	200	0.0222	200	0.0562	200	0.0196
300	0.0237	300	0.0577	300	0.0155	300	0.0385	300	0.0803	300	0.0324
400	0.0617	400	0.0172	400	0.0195	400	0.0049	400	0.0236	400	0.0596
500	0.0291	500	0.0110	500	0.0091	500	0.0461	500	0.0492	500	0.0497
600	0.0136	600	0.0142	600	0.0035	600	0.0075	600	0.0040	600	[illegible]
700	0.0049	700	0.0987	700	0.0023	700	0.0103	700	0.0094	700	[illegible]
800	0.0092	800	0.0424	800	0.0245	800	0.0432	800	0.0027	800	0.0098
900	0.0109	900	0.0351	900	0.0037	900	0.0113	900	0.0102	900	0.0603
1000	0.0162	1000	0.0181	1000	0.0100	1000	0.0067	1000	0.0095	1000	0.0365
1100	0.0247	1100	0.0247	1100	0.0229	1100	0.0251	1100	0.0035	1100	0.0181
1200	0.0046	1300	0.1202	1200	0.0180	1200	0.0230	1200	0.0431	1200	0.0008
1300	0.0128	1300	0.0050	1300	0.0198	1300	0.0592	1300	0.0156	1300	0.0769
1400	0.0281	1400	0.0038	1400	0.0343	1400	0.0070	1400	0.0029	1400	0.0472
1500	0.0200	1500	0.0817	1500	0.0087	1500	0.0083	1500	0.0377	1500	0.0410
1600	0.0030	1600	0.0140	1600	0.0672	1600	0.0985	1600	0.0196	1600	0.0328
1700	0.0983	1700	0.0455	1700	0.0127	1700	0.0325	1700	0.0474	1700	0.0322
1800	0.0642	1800	0.0167	1800	0.0071	1800	0.0096	1800	0.0277	1800	0.0290
1900	0.2163	1900	0.0703	1900	0.0015	1900	0.0428	1900	0.0978	1900	0.0134
2000	0.0702	2000	0.0073	2000	0.0062	2000	0.0318	2000	0.0431	2000	0 0101
2100	0.0036	2100	0.0389	2100	0.0487	2100	0.0008	2100	0.0540	2100	0.0022
2200	0.0190	2200	0.0210	2200	0.0144	2200	0.0195	2200	0.0708	2200	0.0177
2300	0.0082	2300	0.0074	2300	0.0155	2300	0.0301	2300	0.0277	2300	0.0159
2400	0 0064	2400	0.0104	2400	0.0239	2400	0.0152	2400	0.0857	2400	0 0626
2500	0.0189	2500	0.0226	2500	0.0020	2500	0.0143	2500	0.0406	2500	0.0175
2600	0 0442	2600	0.0528	2600	0.0063	2600	0.0258	2600	0.0082	2600	0.0771
2700	0.0436	2700	0.0445	2700	0.0356	2700	0.0271	2700	0.1243	2700	0.0011
2800	0.1045	2800	0.0066	2800	0.0056	2800	0.0153	2800	0.0003	2800	0.0760
2900	0.0130	2900	0.0036	2900	0.0214	2900	0.0048	2900	0.0328	2900	0 0046
3000	0.0344	3000	0.0493	3000	0 0181	3000	0.0097	3000	0.0021	3000	0.0043

1	0.2000	1	1.0000	1	1.0000	1	1.0000	1	0.2000	1	0 2000
100	0.0591	100	0.0536	100	0.0205	100	0.0079	100	0.0016	100	0.0272
200	0.0185	200	0.0006	200	0.0208	200	0.0159	200	0.0484	200	0.0085
300	0.0111	300	0.0345	300	0.0793	300	0.0107	300	0.0372	300	0.0056
400	0 0051	400	0.0400	400	0.0174	400	0.0437	400	0.0062	400	0.0380
500	0.0681	500	0.0186	500	0.0153	500	0.0014	500	0.0053	500	0.0390
600	0.0025	600	0.0703	600	0.0042	600	0.0117	600	0.0329	600	0.0899
700	0.0084	700	0.0246	700	0.0120	700	0.0368	700	0.0875	700	0 0383
800	0.0705	800	0.0218	800	0.0025	800	0.0057	800	0.0488	800	0 0295
900	0.0557	900	0.0735	900	0.0144	900	0.0218	900	0.0905	900	0.0423
1000	0.0110	1000	0.0002	1000	0.0491	1000	0.0461	1000	0.0042	1000	0.0185
1100	0.0757	1100	0.0951	1100	0.0111	1100	0.0096	1100	0.0218	1100	0.1092
1200	0.0134	1200	0.0334	1200	0.0414	1200	0.0487	1200	0.0294	1200	0.0582
1300	0.1425	1300	0.0419	1300	0.0442	1300	0.0114	1300	0.0067	1300	0.0376
1400	0.0230	1400	0.0240	1400	0.0243	1400	0.0387	1400	0.0160	1400	0 0645
1500	0.0111	1500	0.0124	1500	0.0142	1500	0.0094	1500	0.0170	1500	0.0920
1600	0.0008	1600	0.0006	1600	0.0630	1600	0.0371	1600	0.0037	1600	0 0220
1700	0.0325	1700	0.0021	1700	0.0090	1700	0.0555	1700	0.0042	1700	0.0069
1800	0 0111	1800	0.0460	1800	0.0458	1800	0.0054	1800	0.0081	1800	0.0516
1900	0.0376	1900	0.0215	1900	0.0683	1900	0.0162	1900	0.0146	1900	0.0696
2000	0.0171	2000	0.0076	2000	0.0091	2000	0.0025	2000	0.0067	2000	0.0012
2100	0.0051	2100	0.0513	2100	0.0305	2100	0.0333	2100	0.0326	2100	0.1032
2200	0.0140	2200	0.0300	2200	0.0122	2200	0.0123	2200	0.1630	2200	0.0129
2300	0.0081	2300	0.0301	2300	0.1697	2300	0.0434	2300	0.0487	2300	0.0016
2400	0.0075	2400	0.0135	2400	0.0061	2400	0.0190	2400	0.0039	2400	0 0175
2500	0.0068	2500	0.0139	2500	0.0396	2500	0.0190	2500	0.0321	2500	0.0275
2600	0.1660	2600	0.0066	2600	0.0354	2600	0.0036	2600	0.0288	2600	0.0110
2700	0.0043	2700	0.1029	2700	0.0270	2700	0.0256	2700	0.0236	2700	0.0311
2800	0.0012	2800	0.0089	2800	0.0026	2800	0.0594	2800	0 0210	2800	0.0043
2900	0.0260	2900	0.0142	2900	0.0308	2900	0.0858	2900	0.0337	2900	0 0664
3000	0.0030	3000	0.0891	3000	0.0408	3000	0.0159	3000	0.0019	3000	0 0808

B.2 Additional Program Files

EQ.C

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX         256  /* Max # of signals in constellation */
#define MAX_PATHS    10  /* Max # of paths in channel         */
#define PI            3.141592654
#define MAXTERMS    440  /* Number of terms in convolution    */
#define OFFSET      220  /* Position of reference tap         */
#define MAXTAPS      40  /* Max # of taps allowed             */
#define Nm            2  /* Used for display purposes         */
#define SIMP        100  /* No of gaps used by simpson rule   */

typedef struct FCOMPLEX {
  double r,i;
  } fcomplex;

main()
{
  int i, j, k, num_ch;
  float Gc_mod[MAX_PATHS+1], Gc_ang[MAX_PATHS+1], delay[MAX_PATHS+1];
  float roll, val[Nm+1];
  double kdel[MAX_PATHS+1], mag;
  double MINMEANSQERR(), MMSE, LMMSE;
  fcomplex SIG_SET[MAX+1], g[MAXTERMS+1], f[MAXTERMS+1];
  fcomplex Cadd(), Csub(), Cmul(), Cdiv();
  fcomplex Complex(), Conjg(), Arg();
  double Cabs(), Sum;
  FILE  *fp, *f1, *f2;
  int L, M, N, M1, k1;
  int Lmin, Lmax, Lstep, Nmin, Nmax, Nstep;
  int jNm, max_num, chk, l, m;
  void ADAPTIVEMSE();
  char choice, filename[10], file_nl[13], file_nla[14];
  int    n, Diff_Mag, done;
  float  N0, magsum;
  double point[SIMP+1], qam_mag[MAX+1], v29_mag[MAX+1], freq[MAX+1];
  double integral, hvalue[SIMP+1], factor, fac, sqsum[MAXTAPS+1];
  double amt[SIMP+1];
  fcomplex value[SIMP+1], add1, add2, fstore;
/**************************************************************/
  printf("Enter FILENAME to be written to(Max 8 characters) :");
  scanf("%s",filename);
  fp = fopen(filename,"a");
  if (fp == NULL)
  {
    printf("Cannot open File");
    exit(1);
  }
/*******************************************************************/
  printf("ENTER NUMBER OF EXTRA PATHS IN CHANNEL: ");
  scanf("%d", \\num_ch); /* Number of paths in channel */
  fprintf(fp,"\nNumber of paths = %2d\n",num_ch);
  printf("Number of paths = %2d\n",num_ch);
  for (i=1;i<=num_ch;i++)
  {
    printf("\nFor PATH #%d, enter parameters\n",i);
    printf("Path Magnitude Response        - Gc_mod :");
    scanf("%f", \\Gc_mod[i]);
    printf("Path Angle Response in Degrees - Gc_ang :");
    scanf("%f", \\Gc_ang[i]);
    Gc_ang[i] = Gc_ang[i]/180.00;
    printf("Path Time Delay in units of T  - delay  :");
    scanf("%f", \\delay[i]);
  }
  for (i=1;i<=num_ch;i++)
  {
    fprintf(fp,"Path #%2d Gc_mod=%7.3f, Gc_ang=%7.3f,
               delay=%7.3f\n",i,Gc_mod[i],Gc_ang[i]*180,delay[i]);
  }
/*******************************************************************/
/** Using Simpson's Rule to Evaluate an integral ******************/
/*******************************************************************/
 for (n=0;n<=SIMP;n++)
 {
   point[n] = - 1.0/2 + 1.0 * n / SIMP;
 }
 integral = 0.0;
 for (n=0;n<=SIMP;n++)
 {
   hvalue[n] = 0.0;
   value[n] = Complex(0.0,0.0);
   for (i=1;i<=num_ch;i++)
   {
     add1 = Complex(Gc_mod[i]*cos(Gc_ang[i]*PI),
                    Gc_mod[i]*sin(Gc_ang[i]*PI));
     add2 = Complex(cos(2*PI*delay[i]*point[n]),
                    -sin(2*PI*delay[i]*point[n]));
     add1 = Cmul(add1,add2);
     value[n] = Cadd(value[n],add1);
   }
  value[n] = Cadd(value[n],Complex(1.0,0.0));
  hvalue[n] = 1.0 / ( Cabs(value[n]) * Cabs(value[n]) );
  }
  for (n=0;n<=SIMP;n++)
```

```
   integral += hvalue[n] ;
  integral  -= 0.5 * ( hvalue[0] + hvalue[SIMP] );
  integral  /= SIMP;
  fprintf(fp,"\nIntegral    =%8.4f",integral);
  fflush(fp);
/**********Pulse Shape **********************************/
  printf("Enter Roll factor between 0 and 1.");
  scanf("%f", \\roll);
  fprintf(fp,"\nRoll factor =%8.4f",roll);
/*******************************************************/
/* Determine the overall impulse response of          */
/* transmitter, channel and                            */
/* receiver.                                           */
/*******************************************************/
  for (k=0;k<=MAXTERMS;k++)
  {
    g[k] = Complex(0.0,0.0);
    k1 = k - OFFSET;
    g[k].r = sin(PI*k1)/(PI*k1)*
             cos(roll*PI*k1)/(1 -(2*roll*k1)*(2*roll*k1));
    if (k1 ==0) g[OFFSET].r = 1.0000;
    for (i=1;i<=num_ch;i++)
    {
      kdel[i] = k1 - delay[i];
      if ((kdel[i] != 0.0000)  \\ \\ (roll == 0.0))
      {
        g[k].r += Gc_mod[i] * cos(Gc_ang[i]*PI)
                   * sin(PI*kdel[i])/(PI*kdel[i]);
        g[k].i += Gc_mod[i] * sin(Gc_ang[i]*PI)
                   * sin(PI*kdel[i])/(PI*kdel[i]);
      }
      else if ((kdel[i] != 0.0000)  \\ \\ (roll != 0.0))
      {
        g[k].r += Gc_mod[i] * cos(Gc_ang[i]*PI)
             * sin(PI*kdel[i])/(PI*kdel[i])
     * cos(roll*PI*kdel[i])
             / (1 - (2*roll*kdel[i])*(2*roll*kdel[i]));
        g[k].i += Gc_mod[i] * sin(Gc_ang[i]*PI)
             * sin(PI*kdel[i])/(PI*kdel[i])
     * cos(roll*PI*kdel[i])
             / (1 - (2*roll*kdel[i])*(2*roll*kdel[i]));
      }
      else
      {
        g[k].r += Gc_mod[i] * cos(Gc_ang[i]*PI);
        g[k].i += Gc_mod[i] * sin(Gc_ang[i]*PI);
      }
    }
/*
    if ((k%Nm)==1) fprintf(fp,"\n");
  fprintf(fp,"g[%4d]=%8.4f,%8.4fj ",k1,g[k].r,g[k].i);
*/
 }
 fflush(fp);
/***********************************************************************/
/* Determine receiver response for channel additive noise             */
/***********************************************************************/
 for (k=0;k<=MAXTERMS;k++)
 {
  k1 = k - OFFSET;
  f[k] = Complex(0.0,0.0);
  if (roll == 0.0) f[OFFSET].r = 1.0;
  else
  {
   for (i=0;i<=SIMP;i++) amt[i] = 0.0;
   for (i=0;i<=SIMP;i++) amt[i] = sqrt(1-sin(PI*(2*i/SIMP -1)/2));

   /***** even k1 ******/
   if ((k1 != 0)  \\ \\ (k1%2 == 0))
   {
    fstore = Complex(0.0,0.0);
    for (i=0;i<=SIMP;i++)
     fstore.r += 2 * amt[i] * cos(roll*PI*k1*(2*i/SIMP - 1));
    fstore.r -= (amt[0] + amt[SIMP]) * cos(roll*PI*k1);
    fstore.r *= (roll/SIMP/sqrt(2.0));
    f[k].r = sin(k1*PI*(1-roll))/PI/k1 + fstore.r ;

    for (i=0;i<=SIMP;i++)
     fstore.i += 2 * amt[i] * sin(roll*PI*k1*(2*i/SIMP - 1));
    fstore.i -= (amt[SIMP] - amt[0]) * sin(roll*PI*k1);
    fstore.i *= (roll/SIMP/sqrt(2.0));
    f[k].i = fstore.i;
   }
   /***** odd k1 *******/
   else if ((k1 != 0)  \\ \\ (k1%2 != 0))
   {
    fstore = Complex(0.0,0.0);
    for (i=0;i<=SIMP;i++)
     fstore.r += 2 * amt[i] * cos(roll*PI*k1*(2*i/SIMP - 1));
    fstore.r -= (amt[0] + amt[SIMP]) * cos(roll*PI*k1);
    fstore.r *= (roll/SIMP/sqrt(2.0));
    f[k].r = sin(k1*PI*(1-roll))/PI/k1 - fstore.r ;

    for (i=0;i<=SIMP;i++)
     fstore.i += 2 * amt[i] * sin(roll*PI*k1*(2*i/SIMP - 1));
    fstore.i -= (amt[SIMP] - amt[0]) * sin(roll*PI*k1);
    fstore.i *= (roll/SIMP/sqrt(2.0));
    f[k].i -= fstore.i;
   }
   else if (k1 ==0)
```

```
      {
        fstore = Complex(0.0,0.0);
        for (i=0;i<=SIMP;i++)
          fstore.r += 2 * amt[i];
        fstore.r -= (amt[0] + amt[SIMP]);
        fstore.r *= (roll/SIMP/sqrt(2.0));
        f[OFFSET].r = 1 - roll + fstore.r;
      }
    }
  }
  fflush(fp);
/*******************************************************************/
/* Determine the input data signals                                */
/* Either PSK, QAM, V29 or other, M=2,4,8,16,32,64,128,256         */
/*******************************************************************/
  for (i=0;i<=MAX;i++) SIG_SET[i] = Complex(0.0,0.0);
  for (i=0;i<=MAXTAPS;i++) sqsum[i] = 0.0;
  Sum = 0.0;
  printf("\n WHAT SIGNAL CONSTELLATION IS DESIRED?\n");
  printf(" Enter  P or p(PSK), Q or q(QAM),  V or v(V29)");
  printf(" or something else \n\t:");
  scanf("%s", \\choice);

  if ((choice == 'p') || (choice == 'P'))
  {
    printf("PSK Chosen: How Many Points?:   ");
    scanf("%d", \\M);
    fprintf(fp,"\n\n%dPSK",M);
    for (k=1;k<=M;k++)
    {
      SIG_SET[k].r = cos(2*PI*k/M);
      SIG_SET[k].i = sin(2*PI*k/M);
    }
  }
  else if ((choice == 'Q') || (choice == 'q'))
  {
    printf("QAM chosen: How Many Points?:   ");
    scanf("%d", \\M);
    fprintf(fp,"\n\n%2dQAM",M);
    M1 = (int)(float)sqrt(1.00*M);
    for (k=0;k<=M1-1;k++)
    {
      for (k1=1;k1<=M1;k1++)
      {
        SIG_SET[k*M1+k1].r = (2.0*k1 - M1 - 1);
        SIG_SET[k*M1+k1].i = (2.0*k  - M1 + 1);
        Sum = Sum + SIG_SET[k*M1+k1].r * SIG_SET[k*M1+k1].r +
                    SIG_SET[k*M1+k1].i * SIG_SET[k*M1+k1].i;
      }
    }
```

```
printf("\nSum of squares= %8.4f",Sum);
Sum = sqrt(Sum/M);
Diff_Mag = 0;
for (i=1;i<=MAX;i++) freq[i] = 0.0;
for (i=1;i<=MAX;i++) qam_mag[i] = 0.0;
for (k=0;k<=M1-1;k++)
{
  for (k1=1;k1<=M1;k1++)
  {
    SIG_SET[k*M1+k1].r /= Sum;
    SIG_SET[k*M1+k1].i /= Sum;
    mag = Cabs(SIG_SET[k*M1+k1]);
    done = 1;
    for (i=1;i<=Diff_Mag;i++)
    {
      if (qam_mag[i] == mag)
      {
        freq[i] += 1.0/M;
        done = 0;
      }
    }
    if (done == 1)
    {
      Diff_Mag++;
      qam_mag[Diff_Mag] = mag;
      freq[Diff_Mag] += 1.0/M;
    }
  }
}
for (i=1;i<=Diff_Mag;i++) sqsum[1] += freq[i]/qam_mag[i]/qam_mag[i];

for (i=1;i<=Diff_Mag;i++)
  for (j=1;j<=Diff_Mag;j++)
    sqsum[2] += freq[i] * freq[j] /(qam_mag[i]+qam_mag[j])
                                   /(qam_mag[i]+qam_mag[j]);
for (i=1;i<=Diff_Mag;i++)
  for (j=1;j<=Diff_Mag;j++)
    for (k=1;k<=Diff_Mag;k++)
      sqsum[3] += freq[i] * freq[j] * freq[k]
                          / (qam_mag[i]+qam_mag[j]+qam_mag[k])
                          / (qam_mag[i]+qam_mag[j]+qam_mag[k]);
for (i=1;i<=Diff_Mag;i++)
  for (j=1;j<=Diff_Mag;j++)
    for (k=1;k<=Diff_Mag;k++)
      for (l=1;l<=Diff_Mag;l++)
        for (m=1;m<=Diff_Mag;m++)
        sqsum[5] += freq[i] * freq[j] * freq[k] * freq[l] * freq[m]
        /(qam_mag[i]+qam_mag[j]+qam_mag[k]+qam_mag[l]+qam_mag[m])
        /(qam_mag[i]+qam_mag[j]+qam_mag[k]+qam_mag[l]+qam_mag[m]);
for (i=1;i<=Diff_Mag;i++)
```

```
    {
      fprintf(fp,"\nQam_Mag= %8.4f, Freq = %8.4f",qam_mag[i],freq[i]);
    }
    for (i=1;i<=5;i++)
    {
      if (i!=4) fprintf(fp,"\nsqsum[%d]= %8.4f",i,sqsum[i]);
    }
  }
  else if ((choice == 'V') || (choice == 'v'))
  {
    printf("V29 chosen: How Many Points?:   ");
    scanf("%d", \\M);
    printf("%d V29 SIGNAL SET",M);
    fprintf(fp,"\n\n%2dV29",M);
    M1 = M/8;
    printf("\nM1  ==%d",M1);
    mag = 1.0;
    Diff_Mag = M1 * 2;
    for (i=0;i<=MAX;i++) v29_mag[i] = 0.0;
    for (i=0;i<=M1-1;i++)
    {
      j = 8 * i + 1;
      for (k=j;k<=j+3;k++)
      {
        SIG_SET[k].r = mag * cos(PI*(2*k-1)/4);
        SIG_SET[k].i = mag * sin(PI*(2*k-1)/4);
        Sum = Sum + SIG_SET[k].r * SIG_SET[k].r
                  + SIG_SET[k].i * SIG_SET[k].i;
      }
      v29_mag[2*i+1] = mag;
      mag = mag * (2*i+3) / (2*i+1) / sqrt(2.0);
      for (k=j+4;k<=j+7;k++)
      {
        SIG_SET[k].r = mag * cos(PI*k/2);
        SIG_SET[k].i = mag * sin(PI*k/2);
        Sum = Sum + SIG_SET[k].r * SIG_SET[k].r
                  + SIG_SET[k].i * SIG_SET[k].i;
      }
      v29_mag[2*i+2] = mag;
      mag *= sqrt(2.0);
    }
    printf("\nSum of squares= %8.4f",Sum);
    Sum = sqrt(Sum/M);
    for (k=1;k<=M;k++)
    {
      SIG_SET[k].r /= Sum;
      SIG_SET[k].i /= Sum;
    }
    for (i=1;i<=Diff_Mag;i++) v29_mag[i] /= Sum;
  for (i=1;i<=Diff_Mag;i++)
    sqsum[1] += 1.0/(v29_mag[i])/(v29_mag[i]) ;

  for (i=1;i<=Diff_Mag;i++)
    for (j=1;j<=Diff_Mag;j++)
      sqsum[2] += 1.0 /(v29_mag[i]+v29_mag[j])
                     /(v29_mag[i]+v29_mag[j]);
  for (i=1;i<=Diff_Mag;i++)
    for (j=1;j<=Diff_Mag;j++)
      for (k=1;k<=Diff_Mag;k++)
        sqsum[3] += 1.0 /(v29_mag[i]+v29_mag[j]+v29_mag[k])
                        /(v29_mag[i]+v29_mag[j]+v29_mag[k]);
  for (i=1;i<=Diff_Mag;i++)
    for (j=1;j<=Diff_Mag;j++)
      for (k=1;k<=Diff_Mag;k++)
        for (l=1;l<=Diff_Mag;l++)
          for (m=1;m<=Diff_Mag;m++)
            sqsum[5] += 1.0
              /(v29_mag[i]+v29_mag[j]+v29_mag[k]+v29_mag[l]+v29_mag[m])
              /(v29_mag[i]+v29_mag[j]+v29_mag[k]+v29_mag[l]+v29_mag[m]);
  for (i=1;i<=5;i++)
    sqsum[i] /= pow( (double) Diff_Mag, (double) i);
  fprintf(fp,"\n");
  for (i=1;i<=5;i++)
    fprintf(fp,"%8.1f",pow( (double) Diff_Mag, (double) i));
  for (i=1;i<=Diff_Mag;i++)
  {
    fprintf(fp,"\nV29_Mag= %8.4f",v29_mag[i]);
  }
  for (i=1;i<=5;i++)
  {
    if (i!=4) fprintf(fp,"\nsqsum[%d]= %8.4f",i,sqsum[i]);
  }
}
else /* ANY OTHER SET */
{
  printf("\n Enter Number of Points in Signal Constellation:\n");
  scanf("%d", \\M);
  fprintf(fp,"\n\nSignal Set not PSK or QAM or V29");
  for (k=1;k<=M;k++)
  {
    printf("SIG[%d].r =",k);
    scanf("%f",SIG_SET[k].r);
    printf("SIG[%d].i = ",k);
    scanf("%f",SIG_SET[k].i);
    Sum = Sum + SIG_SET[k].r * SIG_SET[k].r +
                SIG_SET[k].i * SIG_SET[k].i;
  }
  printf("\nSum of squares = %8.4f",Sum);
  Sum = sqrt(Sum);
```

```
    for (k=1;k<=M;k++)
    {
      SIG_SET[k].r /= Sum;
      SIG_SET[k].i /= Sum;
    }
  }
/*************************************************************************/
/************************* Print Constellation Set *********************/
/*************************************************************************/
  for (k=1;k<=M;k++)
  {
    if ((k%Nm) == 1) fprintf(fp,"\n");
    fprintf(fp,"s[%3d]=%8.4f,%8.4fj  ",k,SIG_SET[k].r,SIG_SET[k].i);
  }
  fflush(fp);
/*************************************************************************/
/* What are the step-sizes that are used                                 */
/*************************************************************************/
  for (i=0;i<=Nm;i++) val[i]= 0.0;
  printf("\nNumber of ALPHAS to be entered:");
  scanf("%d", \\max_num);
  for (chk=1;chk<=max_num;chk++)
  {
    printf("Enter ALPHA[%2d]:",chk);
    scanf("%f", \\val[chk]);
  }
/**********************************************************************/
/* Determine Noise Power                                              */
/**********************************************************************/
  scanf("%f", \\N0);
  fprintf(fp,"\n\nN0/2= %8.4f dB-',N0);
  N0 /= 10.0;
  N0 = 1.0/pow(10.0,N0);
  fprintf(fp,"\nN0/2= %8.4f ",N0);
  fflush(fp);
/**********************************************************************/
/* Determine L and N ranges and steps                                 */
/**********************************************************************/
  scanf("%d", \\Nmin);
  scanf("%d", \\Nmax);
  scanf("%d", \\Nstep);
  scanf("%d", \\Lmin);
  scanf("%d", \\Lmax);
  scanf("%d", \\Lstep);
/************************************************************************/
/** Perform Simulations                                                */
/************************************************************************/
  for (N=0;N<=Nmax;N=N+Nstep)
  {
    {
```

```
fprintf(fp,"\n\nNumber of Equalizer Taps besides C[0] = %2d",N);
fflush(fp);
MMSE = MINMEANSQERR(M,N,SIG_SET,g,f,fp,val,N0);
fprintf(fp,"\nN=%2d,\tMMSE=%8.4f\n",N,MMSE);
fflush(fp);
for (L=Lmin;L<=Lmax;L=L+Lstep)
{
  if (L != [illegible]
  {
    fprintf(fp,"\nL = %2d    ",L);
    magsum = 0.0; factor = 0.0; fac = 0.0;
    if ((choice == 'p') || (choice == 'P'))
    {
      magsum = 1.0 * L;
      factor = L * N0 * integral / (magsum * magsum);
      fac    = 1 - (0.5 * factor * factor);
      LMMSE  = 1 - (1-MMSE) * fac * fac;
      fprintf(fp,"MMSE=%8.4f",LMMSE);
    }
    else if ((choice == 'q') || (choice == 'Q'))
    {
      factor = L * N0 * integral * sqsum[L] ;
      fac    = 1 - (0.5 * factor * factor);
      LMMSE  = 1 - (1-MMSE) * fac * fac;
      fprintf(fp,"MMSE=%8.4f",LMMSE);
    }
    else if ((choice == 'v') || (choice == 'V ))
    {
      factor = L * N0 * integral * sqsum[L];
      fac    = 1 - (0.5 * factor * factor);
      LMMSE  = 1 - (1-MMSE) * fac * fac;
      fprintf(fp,"MMSE=%8.4f",LMMSE);
    }
    else fprintf(fp,"\n PROGRAM NOT AVAILABLE FOR SIGNAL SET");
    fflush(fp);
    if (N == 8) && (L == 3)
    {
      scanf("%s",file_nl);
      f1 = fopen(file_nl,"w");
      if (f1 == NULL) printf("Cannot open File");  exit(1);
      sprintf(file_nla,"%s%c",file_nl,'a');
      f2 = fopen(file_nla,"w");
      if (f2 == NULL)
      {
        printf("Cannot open File"); exit(1);
      }
      ADAPTIVEMSE(L,M,N,SIG_SET,g,f,fp,f1,f2,val,N0);
      fclose(f1);
      fclose(f2);
    }
```

```
        }
      }
    }
  }
  fclose(fp);
}
```

CINV.C

```
/***********************************************************/
/********** CINV() - Taken from: Numerical Recipes in C ****/
/********** Altered to invert complex matrices          ****/
/********** instead of just real matrices               ****/
/***********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAXTAPS   40
#define TINY 1.0e-20

typedef struct FCOMPLEX {
  double r,i;
  }  fcomplex;

fcomplex Cadd(),Csub(),Cmul(),Cdiv(),RCmul(),Complex();
double   Cabs();
/*********************************************/
/**************** LUBKSB *********************/
/*********************************************/
void lubksb(A,N,indx,b)
fcomplex A[MAXTAPS+1][MAXTAPS+1], b[MAXTAPS+1];
int N, indx[MAXTAPS+1];
{
  int i, ii=0, ip, j;
  fcomplex sum;

  for (i=1;i<=N;i++)
  {
    ip = indx[i];
    sum = b[ip];
    b[ip] = b[i];
    if (ii)
      for (j=ii;j<=i-1;j++) sum = Csub(sum,Cmul(A[i][j],b[j]));
    else if (Cabs(sum)>0.000) ii= i;
    b[i] = sum;
  }
  for (i=N;i>=1;i--)
  {
    sum = b[i];
    for (j=i+1;j<=N;j++) sum = Csub(sum,Cmul(A[i][j],b[j]));
    if (  Cabs(A[i][i]) > 0.00)
    {
       b[i] = Cdiv(sum,A[i][i]);
    }
    else b[i]=sum;
  }
}
/**********************************************/
/******** LUDCMP ******************************/
/**********************************************/
void ludcmp(A,N,indx,d)
int N, indx[MAXTAPS+1];
double *d;
fcomplex A[MAXTAPS+1][MAXTAPS+1];

{
  int i, imax, j, k;
  double big, dum, temp;
  double *vv, *vector();
  fcomplex sum, dum2, dum3;
  void nrerror(), free_vector();

  vv = vector(1,N);
  *d = 1.0;
  for (i=1;i<=N;i++)
  {
    big = 0.0;
    for (j=1;j<=N;j++)
    {
      if ((temp = Cabs(A[i][j])) > big) big = temp;
    }
    if (big == 0.0) nrerror("Singular matrix in routine LUDCMP");
    vv[i] = 1.0/big;
  }

  for (j=1;j<=N;j++)
  {
    for (i=1;i<j;i++)
    {
      sum = A[i][j];
      for (k=1;k<i;k++)
        sum = Csub(sum,Cmul(A[i][k],A[k][j]));
      A[i][j] = sum;
    }
    big = 0.0;
    for (i=j;i<=N;i++)
    {
      sum = A[i][j];
      for (k=1;k<j;k++)
```

```
        sum = Csub(sum,Cmul(A[i][k],A[k][j]));
      A[i][j] = sum;
      if ((dum= vv[i] * Cabs(sum)) >= big)
      {
        big = dum;
        imax = i;
      }
    }
    if (j!=imax)
    {
      for (k=1;k<=N;k++)
      {
        dum2 = A[imax][k];
        A[imax][k] = A[j][k];
        A[j][k] = dum2;
      }
      *d  = - (*d);
      vv[imax]=vv[j];
    }
    indx[j] = imax;
    if (Cabs(A[j][j]) == 0.0)  /***** Question **/
    {
      printf("\nTINYYY%d",j);
      printf("\nA = %8.4f+%8.4fj",A[j][j].r,A[j][j].i);
      A[j][j].r = TINY;
      A[j][j].i = 0.00;
    }
    if (j!=N)
    {
      dum3 = Complex(1.0,0.0);
      dum2 = Cdiv(dum3,A[j][j]);
      for (i=j+1;i<=N;i++)  A[i][j] = Cmul(A[i][j],dum2);
    }
  }
  free_vector(vv,1,N);
}
/**************************************************/
/********** matrix Inversion   program **********/
/**************************************************/
#define Nm 3

void cinv(A,y,N)
fcomplex A[MAXTAPS+1][MAXTAPS+1], y[MAXTAPS+1][MAXTAPS+1];
int N;
{
  int i, j, indx[MAXTAPS+1], k;
  fcomplex ID[MAXTAPS+1][MAXTAPS+1],AA[MAXTAPS+1][MAXTAPS+1];
  fcomplex col[MAXTAPS+1];
  double d;
  int jNm;
  /******** Save Matrix A in AA ******************/
  for (i=1;i<=N;i++)
    for (j=1;j<=N;j++)
    {
       y[i][j] = Complex(0.0,0.0);
      AA[i][j] = A[i][j];
    }
/********************** INVERT MATRIX ***********************/
  ludcmp(A,N,indx, \\d); /* Decompose matrix just once */
  for (j=1;j<=N;j++)
  {
    /* Find inverse by columns */
    for (i=0;i<=N;i++)
    {
      col[i] = Complex(0.0,0.0);
    }
    col[j] = Complex(1.0,0.0);
    lubksb(A,N,indx,col);
    for (i=1;i<=N;i++) y[i][j] = col[i];
  }
/************Recover A matrix from AA*****************/
  for (i=1;i<=N;i++)
     for (j=1;j<=N;j++)
       A[i][j] = AA[i][j];
}
```

RANDOM.C

```
#include <stdlib.h>
#include <math.h>

#define MAXS 98
/*****************************************************************/
/********* Returns uniform r.v from 0.0 to 1.0       *********/
/*********                                           *********/
/********* From: Numerical Recipes in C. Ch.7 pg.207 *********/
/*****************************************************************/
float ran0(idum)
int *idum;
{
  static float y, maxran, v[MAXS];
  float dum;
  static int iff=0;
  int j;
  unsigned int i, k;
  void nrerror();

  if (*idum < 0 || iff ==0)
  {
    iff = 1;
    i   = 2;
    do
    {
      k = i;
      i = i << 1;
    } while (i);
    maxran = k;
    srand(*idum);
    *idum = 1;
    for (j=1;j<MAXS;j++)
      dum = rand();
    for (j=1;j<MAXS;j++)
      v[j] = rand();
  }
  j = 1 + y * (MAXS -1)/maxran;
  if ((j > (MAXS-1)) || (j < 1))
    nrerror("RAN0: THIS CANNOT HAPPEN");
  y = v[j];
  v[j] = rand();
  return(y/maxran);
}

#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349
/*******************************************************************/
/********** returns a uniformly distributed r.v from 0.0 to 1.0 ****/
/********** Set idum to any negative value to initialize or     ****/
/********** reinitialize the sequence.                          ****/
/********** From: Numerical Recipes in C. Ch.7 pg 210           ****/
/*******************************************************************/
float ran1(idum)
int *idum;
{
  static long ix1, ix2, ix3;
  static float r[98];
  float temp;
  static int iff=0;
  int j;
  void nrerror();
  if (*idum < 0 || iff == 0)
  {
    iff = 1;
    ix1 = (IC1-(*idum)) % M1;
    ix1 = (IA1*ix1+IC1) % M1;
    ix2 = ix1 % M2;
    ix3 = ix1 % M3;
    for (j=1;j<=97;j++)
    {
      ix1 = (IA1*ix1+IC1) % M1;
      ix2 = (IA2*ix2+IC2) % M2;
      r[j]= (ix1+ix2*RM2)*RM1;
    }
    *idum = 1;
  }
  ix1 = (IA1*ix1+IC1) % M1;
  ix2 = (IA2*ix2+IC2) % M2;
  ix3 = (IA3*ix3+IC3) % M3;
  j   = 1 + ((97*ix3)/M3) ;
  if (j>97 || j<1) nrerror("RAN1: This cannot happen");
  temp = r[j];
  r[j] = (ix1+ix2*RM2)*RM1;
  return(temp);
}
/*******************************************************/
/**** Returns a normally distributed deviate with  ****/
/**** zero-mean and unit variance, using ran1(idum)****/
/**** as the source of uniform deviates            ****/
/**** From Numerical Recipes in C. Ch 7.3 pp.216-7 ****/
/*******************************************************/
float gasdev(idum)
int *idum;
{
 static int iset = 0;
 static float gset;
 float fac, r, v1, v2;
 float ran1();

 if (iset == 0)
 { /* We don't have an deviate handy so */
   do
   {
     /***************************************************/
     /** pick two uniform numbers in the square ext-  **/
     /** ending from -1 to +1 in each direction       **/
     /** See if they are in the unit circle, if not   **/
     /** try again                                    **/
     /***************************************************/
     v1 = 2.0 * ran1(idum) - 1.0;
     v2 = 2.0 * ran1(idum) - 1.0;
     r  = v1 * v1 + v2 * v2;
   }
   while (r>= 1.0);
   fac = sqrt(-2.0*log(r)/r);
   /****************************************************/
   /** Now make the Box-Muller Transformation to get  **/
   /** two normal deviates. Return one and save the   **/
   /** other for the next time.                       **/
   /****************************************************/
   gset = v1 * fac;
   /****************************************************/
   /** Set flag.                                      **/
   /****************************************************/
   iset = 1;
   return(v2*fac);
 }
 else
 {
   /****************************************************/
   /** We have an extra deviate handy, so unset the   **/
   /** flag, and return the extra deviate.            **/
   /****************************************************/
   iset = 0;
   return(gset);
 }
}
```

COMPLEX.C

```
#include <stdio.h>
#include <math.h>

typedef struct FCOMPLEX {
  double r,i;
  } fcomplex;

fcomplex Cadd(a,b)
fcomplex a,b;
{
  fcomplex c;

  c.r = a.r + b.r;
  c.i = a.i + b.i;
  return(c);
}

fcomplex Csub(a,b)
fcomplex a,b;
{
  fcomplex c;

  c.r = a.r - b.r;
  c.i = a.i - b.i;
  return(c);
}

fcomplex Cmul(a,b)
fcomplex a,b;
{
  fcomplex c;

  c.r = a.r * b.r - a.i * b.i;
  c.i = a.i * b.r + a.r * b.i;
  return(c);
}

fcomplex Cdiv(a,b)
fcomplex a,b;
{
  fcomplex c;
  double r,den;

  if (fabs(b.r) >= fabs(b.i))
  {
    r=b.i/b.r;
    den = b.r + r * b.i;
    c.r = (a.r + r * a.i)/den;
    c.i = (a.i - r * a.r)/den;
  }
  else
  {
    r = b.r/ b.i;
    den = b.i + r * b.r;
    c.r = (a.r * r + a.i)/den;
    c.i = (a.i * r - a.r)/den;
  }
  return(c);
}

fcomplex Complex(re,im)
double re,im;
{
  fcomplex c;

  c.r = re;
  c.i = im;
  return(c);
}
```

```
double Cabs(z)
fcomplex z;
{
  double x, y, ans, temp;

  x = fabs(z.r);
  y = fabs(z.i);
  if (x==0.0) ans = y;
  else if (y==0.0) ans = x;
  else if (x>y)
  {
    temp = y/x;
    ans = x * sqrt(1.0 + temp * temp);
  }
  else
  {
    temp = x/y;
    ans  = y * sqrt(1.0 + temp * temp);
  }
  return(ans);
}

fcomplex Conjg(z)
fcomplex z;
{
  fcomplex c;

  c.r  = z.r;
  c.i = -z.i;
  return(c);
}

fcomplex Csqrt(z)
fcomplex z;
{
  fcomplex c;
  double x, y, w, r;
  if  ((z.r == 0.0)  \\ \\ (z.i == 0.0))
  {
    c.r = c.i = 0.0;
    return(c);
  }
  else
  {
    x = fabs(z.r);
    y = fabs(z.i);
    if (x >= y)
    {
      r = y/x;
      w = sqrt(x) * sqrt(0.5*(1.0 + sqrt(1.0+r * r)));
    }
    else
    {
      r = x/y;
      w = sqrt(y) * sqrt(0.5*(r+sqrt(1.0 + r * r)));
    }
    if (z.r >= 0.0)
    {
      c.r = w;
      c.i =  z.i/(2.0 * w);
    }
    else
    {
      c.i =  (z.i >= 0) ? w :  -w;
      c.r = z.i /(2.0 * c.i);
    }
    return(c);
  }
}

fcomplex RCmul(x,a)
double x;
fcomplex a;
{

  fcomplex c;
  c.r = x * a.r;
  c.i = x * a.i;
  return(c);
}

fcomplex Arg(z)
fcomplex z;
{
  fcomplex c;
  c.r = z.r/Cabs(z);
  c.i = z.i/Cabs(z);
  return(c);
}
```

UTIL.C

```
/************************************************************/
/********* Utility program: Numerical Recipes in C  ********/
/************************************************************/

#include <malloc.h>
#include <stdio.h>
```

```
void nrerror(error_text)
char error_text[];
{
  void exit();

  fprintf(stderr,"Numerical Recipes run-time error..\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}

double *vector(nl,nh)
int nl, nh;
{
  double *v;

  v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
  if (!v) nrerror("allocation failure in vector()");
  return(v-nl);
}

void free_vector(v,nl,nh)
double *v;
int nl, nh;
{
  free((char*) (v+nl));
}
```