Comparison of Adaptive Algorithms for Cancellation of Harmonic Noise on Distribution Power Lines

by

James Wiggs

CENTER FOR COMMUNICATIONS AND SIGNAL PROCESSING

Department of Computer Studies

CCSP-TR-85/1

Jan 1985

Abstract

WIGGS, JAMES H. Comparison of Adaptive Algorithms for Cancellation of Harmonic Noise on Distribution Power Lines. (Under the direction of Dr. H. J. Trussell.)

Adaptive digital filters have been used for many years in speech processing, echo cancellation, and other areas. The ability of adaptive filters to remove harmonic noise from a contaminated signal, especially when the noise is slowly varying with time, is of special interest in the field of distribution power line carrier communications. This thesis compares the effectiveness of three of the more well-known adaptive digital filter algorithms at removing 60 Hz harmonic noise from an actual distribution power line noise sample: the Widrow-Hopf Least Mean Square (LMS) algorithm, the Least Square Lattice (LSL) algorithm, and the Fast Kalman algorithm. The algorithms are compared in terms of convergence rate, overall noise power reduction, and the ability to reduce the bit error detection rate (BER) of phase-shift-keyed digital data in the noise.

Results indicate that the LMS algorithm, while the slowest to converge, has the best BER performance. It is shown that the performance of the LSL and Fast Kalman algorithms is strongly dependent on the value of the misadjustment parameter; a value of .01 for this parameter causes very poor BER performance, while a value of .1 causes the algorithms to perform almost as well as the LMS algorithm, but with much faster convergence rates.

Acknowledgments

The author wishes to express his appreciation to Dr. H. Joel Trussell for his help and direction in the research for this thesis. His patience, especially in the early part of this endeavor, is also greatly appreciated. The author would also like to thank Jin-Der Wang for his help in generating some of the results, and for the use of his research on the application of the LMS algorithm to distribution power line noise. The generosity of Westinghouse Electric Corporation for their support of a full-time employee's pursuit of this degree is also gratefully acknowledged.

The author would also like to express his appreciation to his wife, Susan Wiggs, for her patience and understanding during the course of this project.

Table of Contents

Table of Contents (Continued)

Chapter I

Introduction

Digital filtering has begun to find widespread use in many applications, ranging from seismic data to speech processing. It has some significant advantages over standard analog filtering, including the capability of analyzing and re-analyzing stored data, and the capability of constructing filters that are not able to be realized by analog means. This does not mean that digital filters are perfect for all applications; on the contrary, analog filters, being much cheaper to realize in the majority of cases, will continue to coexist with digital filters for the foreseeable future. Moreover, analog low-pass filters must be used as front-ends to digital filters to limit the bandwidth of the filtered signal to avoid aliasing effects of sampling. However as digital hardware (especially A/D converters) comes down in price and increases in performance, digital filters will be used more and more for filtering tasks as digital hardware takes over more and more of the traditional analog work.

In this thesis, we will examine the use of adaptive digital filters in the area of noise reduction for power-line carrier communications systems. Power-line communications is implemented by injecting a signal (usually digital) on to the existing transmission or distribution power lines owned by the public utilities. This signal is then detected at a receiver elsewhere on the power system, and the signal is decoded and acted upon. This communication medium is used currently for load management, remote meter reading, and distribution equipment monitoring and control. It is also being used for voice-grade communications in remote housing areas, and in the home for local control and monitoring of appliances.

1.1 Power-Line Carrier Noise Characteristics

Power lines are traditionally a very noisy environment in which to transmit signals, and are divided into 2 broad categories by the utility industry: transmission lines and distribution lines. Transmission lines are those lines which carry power from the generator to the distribution substations, where the power lines branch out to individual homes and are called distribution lines. The noise characteristics of these 2 types of power lines are different.

Because the power line is used to transmit power from the generator to the home, it obviously contains (in this country) 60 Hz signals and its harmonics. All manner of devices are connected to the distribution power lines. Each device injects some signal back on to the power lines. Generally these signals are not very large in relation to the primary 60 Hz voltage, but when it is desired to transmit a control signal on this same transmission medium, the stray signals become significant noise problems, especially when control signals are transmitted over a distance long enough to cause substantial attenuation of the signal, thus decreasing the signal-to-noise ratio. The largest producers of noise on the distribution power line are switching devices, such as light dimmers, which produce voltage/current spikes synchronously with the 60 Hz power [1]. Universal motors and other common household appliances [2] also contribute to the noise on the distribution power lines. These sources of noise can completely mask any communication signal if not removed from the line by some type of filtering technique.

Transmission power lines, however, do not have consumer or industrial switching devices connected to them, because they are very high voltage lines used exclusively to carry power from the generator to the distribution substations. They are therefore much cleaner than distribution power lines, and make a better transmission medium. However, most current interest in power line communications is directed toward the distribution power line, so that the utilities may have access to individual homes and commercial customers.

1.2 Filtering of Power Line Noise

1.2.1 Current Techniques

Currently, the removal of distribution power line noise is accomplished almost exclusively with standard analog filters. Since most communications systems are only interested in a very narrow band of signals, very high Q bandpass filters can be used to filter out all noise outside of the transmission frequency band. This can obviously provide satisfactory performance, since there are systems on the market today which utilize this technique. Note however, that this technique does not remove any noise components in the frequency band of interest. Thus if there is a device producing a large level of noise in the frequency range of interest, this type of filter will not remove that noise.

Another technique used today is to implement the transmission and detection of signals based on the zero-crossings of the 60 Hz power signal. This, however, leads to an absolute maximum data rate of 120 baud, and most systems use much lower data rates.

Since most of the noise on the distribution power line is 60 Hz harmonics, a notch filter can be used which attenuates each 60 Hz harmonic in the frequency band of interest. The drawback to this technique is that the 60 Hz power signal can vary about ± .08% [3]. This is not much of a problem at low frequencies, but at high frequencies the variation becomes significant. For example, at 60 Hz, a ±.08% variation in the 60 Hz harmonics causes the frequency to vary from 59.952 Hz to 60.048 Hz, a bandwidth of only .096 Hz. At the 200th harmonic (12 kHz) however, the frequency varies from 11.99 kHz to 12.01 kHz, a bandwidth of 19.2 Hz.

1.2.2 <u>Digital Techniques</u>

An alternative approach to the current techniques of filtering distribution power line noise is to use some type of digital filtering. This has an intuitive appeal, in that most applications of power line carrier communications use some kind of digital microprocessor as the hardware in the receiver. A very high Q digital filter could be designed to do exactly what its analog counterpart does currently, but the cost of such an approach would almost certainly be higher than the current one, because of the additional A/D converter required and the need for an analog anti-aliasing filter. Without some increase in performance or other added benefit, it is difficult to justify this increased expense.

Another approach involves the use of adaptive digital filters to characterize the noise on the power line before transmission takes place, and then use the resulting filter to clean up the signal at the receiver during reception. The advantage of this approach is that correlated noise <u>in</u> the frequency range of interest can be reduced, thus increasing the effectiveness of the filter and increasing signal-to-noise ratio. Some studies have already been done in this area, notably [1]. The results have been very positive, and this thesis is an extension of the work being done in this area. Adaptive digital filters have been used in a variety of applications including speech processing [4-7], radar/sonar [8], and EKG signal processing [9-10], as well as others.

1.3 <u>Outline of Thesis</u>

This thesis describes the basic theory and implementation of adaptive digital filters for distribution power line carrier noise cancellation. Chapter 2 describes the theory behind three types of adaptive digital filtering techniques - the Widrow-Hopf Least Mean Squares (LMS) algorithm, the Least Squares Lattice (LSL) algorithm, and the Fast Kalman algorithm. Chapter 3 describes the implementation of these algorithms for the current experiments on distribution power line noise, and Chapter 4 presents the results of the experiments for each algorithm. Chapter 5 concludes with a summary of the major results and indicates problem areas and areas of future research.

Chapter II

Theory

2.1 Introduction

2.1.1 System Modeling

The concept of adaptive digital filtering draws heavily from the fields of digital filtering and linear prediction. In the cases of interest here, the system to be modeled by adaptation is assumed to arise from a finite-order discrete-time linear system driven by white noise [11]. That is to say, the signal is assumed to have been generated by passing white noise through a linear filter, either of the all-pole, all-zero, or pole-zero type. Systems so generated are called auto-regressive (AR), moving average (MA), or auto-regressive moving average (ARMA), respectively [12]. These systems can be modeled by all-zero, all-pole, or pole-zero filters of the same order as the filter used to generate the system. It is the goal of modeling to generate the "whitening" filter for a given system of interest, which when applied to the output of the AR process will produce a white noise sequence. In this work we are interested in modeling the power line noise as an AR process plus white noise, as shown in figure 1. This leads to an all-zero whitening filter which can then be used to cancel the correlated signal components of the system.

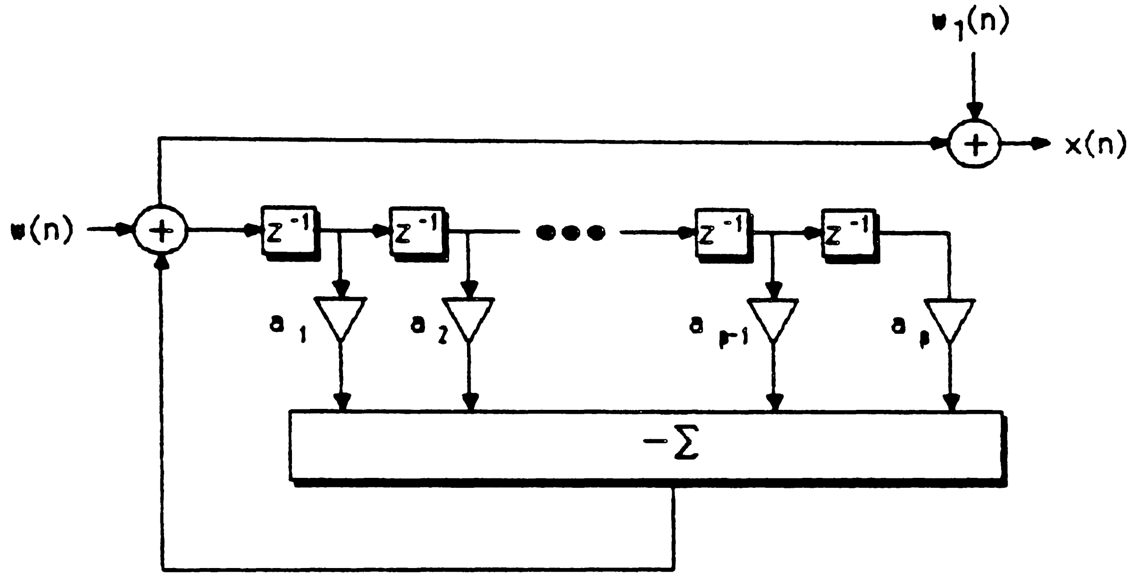

Figure 1

AR Process Generation Model

Note first of all that this assumption of a signal that has been produced by a linear filter implies that some of the signal components in the system are correlated. The best that a whitening filter can hope to do is predict all correlated components of a system; white noise cannot be predicted. Thus an adaptive filter cannot be used to reduce the white noise in the frequency band of interest to power line carrier communications.

The whitening filter takes the form of a linear predictor, which can be implemented as a linear transversal FIR digital filter. A linear predictor has the form

$$y(n) = \sum_{i=1}^{p} a_i \, x(n-i) \tag{2.1}$$

where n is the index corresponding to time, p is the order of the filter, and the a_i are the filter coefficients. The linear predictor tries to predict the value of the desired sequence d(n) from a linear combination of past inputs, x(n). The difference between the predicted value of x(n) and the desired value is called the prediction error, and is defined as

$$e(n) = d(n) - y(n) = d(n) - \sum_{i=1}^{p} a_i \, x(n-i) \tag{2.2}$$

How the desired value d(n) can be obtained is discussed in the next section. This error e(n) is used by all adaptive algorithms to drive the adaptation process. This is shown pictorially in figure 2.

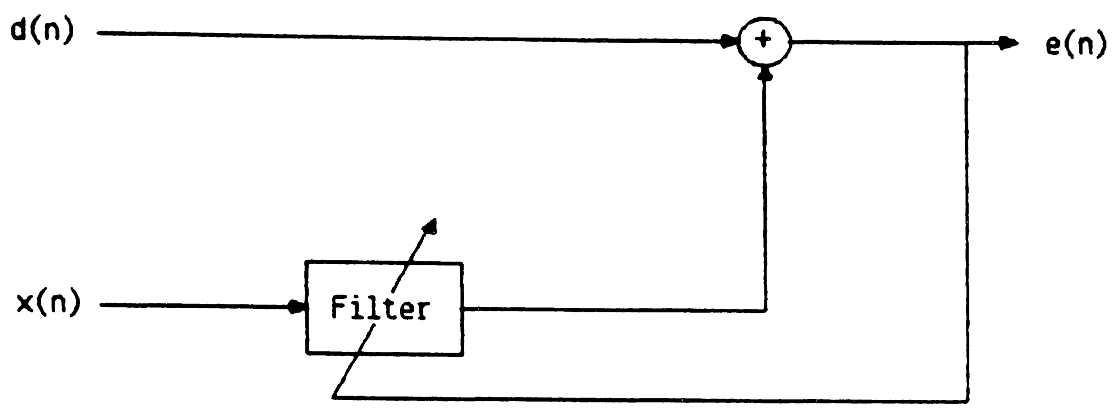

Figure 2

The Adaptive Process

All adaptive modeling techniques seek to "minimize" the error term e(n) in some least-squares sense. The different techniques presented here use different methods and assumptions to minimize the mean square error. These methods and assumptions are discussed in later chapters.

The entire system presented can be shown as in figure 3. A filter is allowed to adapt to the power line noise by one of the adaptive techniques described, and the resulting FIR filter is used to predict the correlated components of the input, which are subtracted out of the original data. As will be mentioned below, a delayed version of the noise input is used as the input x(n), and the current sample of the input is used as d(n). The objective of this thesis is to compare the results of the three different algorithms in terms of convergent rates, reduction of noise power, and bit error rates for phase-shift keyed (PSK) digital data.

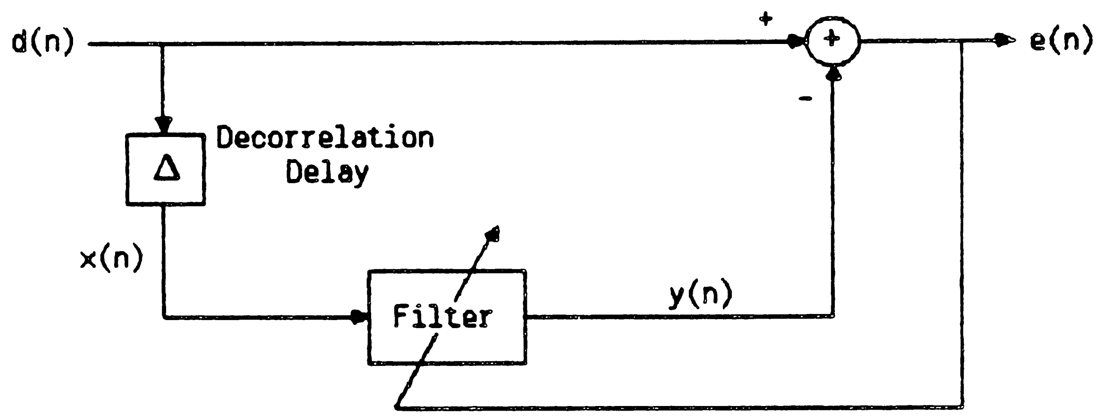

Figure 3

The Adaptive Filter System

2.1.2 Reference Input

Of special interest is the reference input, d(n), which is used to calculate the error of the prediction at time n. This reference input does not have to match point-for-point the "predictable" part of the input; it need only be highly correlated with that part of the input. The degree to which adaptation will occur is directly proportional to the degree of correlation between the reference input d(n) and the input signal. Since the result of all this is a digital filter, the adaptation will produce a filter which "matches" the "predictable" part of the input, and will thus remove those "predictable" parts.

This reference input can be obtained from several sources. In the system of interest, power line carrier communications, the signal is not always present on the power line. A signal is sent (and possibly received), and then the power line is "quiet" for a while. This leads to the concept of "start-stop" adaptation, where the adaptation takes place during periods of "quiet" on the power line, and the filter weights are frozen during periods of transmission. Since all known systems are polled systems, this is easy to implement at the polling site because the transmitter knows when it is going to transmit.

Another approach utilizes the fact that all power line carrier systems operate in a very narrow frequency band. Since all that is needed to cancel the harmonic noise in the transmission frequency band is some input which is correlated with this noise, the fact that the noise outside of the transmission frequency band is very similar to the noise inside the transmission frequency band can be used to advantage. Since the harmonic noise on the power line is a broad-band phenomena, i.e. the 60 Hz harmonics cover the entire frequency range from base-band 60 Hz to the highest frequency of interest, a highly correlated estimate of the "in-band" noise could be gathered from some frequency range which is "out-of-band" with the carrier frequency band. This would allow adaptation to continue at all times, even during transmission of data. This requires a front-end band-pass or low-pass filter to generate this out-of-band reference signal, but this is easily accomplished either by an analog or digital filter. Some results using the LMS algorithm with these different methods of generating the reference noise signal may be found in the paper by Trussell and Wang [1].

There it was discovered that the "start-stop" method always produced good results, and that continuous adaptation using "out-of-band" reference noise was somewhat worse, though still acceptable. Reduction of harmonic noise on the order of 14 to 1 were reported by that study.

2.2 Least Mean Squares (LMS) Adaptive Digital Filter

2.2.1 History

According to Widrow [13], the earliest work on the LMS algorithm grew out of work on noise cancellation. Howells and Applebaum at the General Electric Company worked on a system for antenna sidelobe canceling that used a simple two-weight filter. In 1959 Widrow and Hopf at Stanford University developed what we know today as the LMS algorithm, and used it in a pattern recognition scheme known as Adaline. Other efforts on adaptive filtering were being done independently at Cornell, in the U.S.S.R. and in Britain. In the early to middle 1960's, however, work on adaptive filtering began to take off, with hundreds of papers in the literature devoted to the subject. The best know commercial application of adaptive filtering at that time was the work done by Lucky at Bell Laboratories on high speed modems for digital communications. Since then adaptive filtering has been applied to many problems, including speech processing, radar, sonar, electrocardiography, echo cancellation in phone networks, and geophysics.

2.2.2 Algorithm.

Recall that the output at time n of a linear predictor is given by

$$y(n) = \sum_{i=1}^{p} a_i \, x(n-i) \tag{2.3}$$

which may be written in matrix notation as

$$y(n) = A^T(n) \; X(n) \tag{2.4}$$

where $A(n)$ is defined as $[a_1, a_2, \ldots a_p]^T$ and $X(n)$ is defined as $[x(n-1), x(n-2) \ldots x(n-p)]^T$. The error at time n is given by

$$e(n) = d(n) - y(n) = d(n) - A(n)^T \; X(n) \tag{2.5}$$

The squared error can be written as

$$e(n)^2 = d(n)^2 - 2\, d(n)\, A(n)^T X(n) + A(n)^T X(n) X(n)^T A(n) \tag{2.6}$$

The mean square of the error is found by taking the expectation of both sides of the equation

$$E[e(n)^2] = E[d(n)^2] - 2\, E[d(n) X(n)^T]\, A(n) + A(n)^T E[X(n) X(n)^T]\, A(n) \tag{2.7}$$

Defining $P(n) = E[d(n)X(n)^T]$ and recognizing $E[X(n)X(n)^T]$ as the autocorrelation matrix R gives

$$E[e(n)^2] = E[d(n)^2] - 2P(n)^T A(n) + A(n)^T R(n) A(n) \tag{2.8}$$

It can be noted that the mean square error is a quadratic function of the weights, which can be pictured as a concave hyperparaboloidal surface in p-space [14]. Adjusting the weights to minimize the means square error can be done by "descending" along this surface until the "bottom" of the parabaloid is found. The gradient of the error function can be used to do this, and this method of minimizing the mean square error is known as the gradient method. It is also used in the Gradient Lattice (GL) discussed in section 2.3.1 . The gradient g(n) of the error function is defined as the partial derivative of $E[e(n)^2]$ with respect to the weight vector, which from Eq. 2.8 is

$$g(n) = -2P(n) + 2R(n)A(n) \tag{2.9}$$

The optimal weight vector $A^*(n)$ is found by setting the gradient of the mean square error to zero, or

$$A^*(n) = R^{-1}(n)P(n) \tag{2.10}$$

which is seen to be the Wiener-Hopf equation in matrix form. Since there is no prior knowledge of the autocorrelation matrix R(n) or the correlation matrix P(n), the LMS algorithm tries to approximate the optimal weight vector by iteratively updating a "guess" at the optimal weight vector by taking the present value of the weight vector and making a change to it proportional to the negative of the instantaneous gradient g(n), i. e.,

$$A(n+1) = A(n) - \mu\, g(n) \qquad (2.11)$$

where μ is a parameter that controls stability and rate of convergence, and which will be discussed in more detail later. An estimate, $g(n)'$, of the instantaneous gradient can be found by letting $e(n)^2$ in equation 2.6 be an estimate of the mean-square error and differentiating it with respect to the weight vector $A(n)$. This gives

$$g(n)' = -2\, d(n)\, X(n) + 2\, A(n)^T X(n)\, X(n)^T \qquad (2.12)$$

or

$$g(n)' = 2\, X(n)\, [\, -d(n) + A(n)^T X(n)\,] \qquad (2.13)$$

Noticing that $-d(n) + A(n)^T X(N)$ is $-e(n)$ gives

$$g(n)' = -2\, e(n)\, X(n) \qquad (2.14)$$

Substituting the estimate for the gradient in Eq. 2.11 gives the update formula for the weight vector as

$$A(n+1) = A(n) + 2\,\mu\, e(n)\, X(n) \qquad (2.15)$$

This update formula is attributed to Widrow and Hopf and is known as the Widrow-Hopf Least Mean Squares (LMS) algorithm. This is a very simple update formula, and can be rapidly calculated with 2 multiplications and 2 additions per filter coefficient per input sample. For this reason, the LMS algorithm is widely used in echo cancellation, speech processing, and other applications, despite some of the drawbacks which will be discussed presently.

The parameter μ is the size of the step taken in the opposite direction of the gradient, and is of special importance when evaluating the performance of the LMS algorithm. The rate of convergence of the algorithm has been shown [15] to be proportional to $1/(\mu \lambda_{max})$, where λ_{max} is the maximum eigenvalue of the autocorrelation matrix R of the input data. Actually, each harmonic component of the input, which roughly corresponds to each eigenvalue λ of R, will converge at a rate proportional to $1/(\mu \lambda)$.

Another property of μ is that the LMS algorithm will only converge to the minimum weight vector when [14]

$$1 / \lambda_{max} > \mu > 0 \qquad (2.16)$$

This dependence of the algorithm on the statistical parameters of the input data causes it to be unacceptable in some situations, especially when there are multiple signal components of greatly differing power [16]. The parameter μ also determines the misadjustment M of the algorithm. The misadjustment is the amount of "wander" that the weight vector does around the actual minimum weight vector after it has converged. This misadjustment has been shown to be [14]

$$M = p \mu \lambda_i \qquad (2.17)$$

where λ_i's are the eigenvalues of the input correlation matrix, which correspond roughly to the powers of the orthogonal components of the input. The misadjustment is thus different for each input signal component. Since the rate of convergence is proportional to $1/\mu$ and the level of misadjustment is proportional to μ, the choice of μ is a trade-off between convergence rate and misadjustment after convergence.

2.3 Least Squares Lattice (LSL) Adaptive Digital Filter

2.3.1 History

The Yule-Walker equations are expressed in matrix form as [12]

$$A(n)\ R(n) = E(n) \tag{2.18}$$

where $A(n)$ is $[1, a_1, a_2, \ldots, a_p]$, $E(n)$ is $[e(n)^2, 0, 0, \ldots, 0]$, and $R(n)$ is a matrix of autocorrelation functions

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_p \\ R_{-1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & R_1 \\ R_{-p} & \cdots & R_{-1} & R_0 \end{bmatrix}$$

Each R_i is the autocorrelation with lag i of the input $x(n)$, defined as

$$R_{i-j} = 1/N \sum_{n=1}^{N} x(n+i)\ x(n+j) \tag{2.19}$$

The Yule-Walker equation as expressed above is a means of calculating the optimal Wiener filter coefficients. Solving for A(N) gives

$$A(n) = E(n)\, R^{-1}(n) \tag{2.20}$$

This equation can be solved in several different ways. One way is similar to that shown above in the development of the LMS algorithm. Another method, which iteratively calculates the inverse of the autocorrelation matrix is attributed to Levinson [17] and Durbin [18]. The Levinson-Durbin recursions lead naturally to the representation of the optimal Wiener filter in a lattice form as shown in figure 4, with $K^e = K^r = K$, and where the e_i's and r_i's are known respectively as the forward and backward error residuals at the ith stage of the lattice. This formulation involves the use of the so-called forward and backward prediction error sequences. The forward prediction error e(n) is simply the predictor mentioned in the development of the LMS algorithm, which is

$$e(n) = x(n) - \sum_{i=1}^{p} a_i\, x(n-i) \tag{2.21}$$

where the a_i's are the forward predictor coefficients.

The backward error predictor r(n) is similar to the forward predictor, but works in the reverse (in time) direction, and is given by

$$r(n) = x(n-p) - \sum_{i=0}^{p-1} b_i\, x(n-i) \tag{2.22}$$

where the b_i's are the backward predictor coefficients.

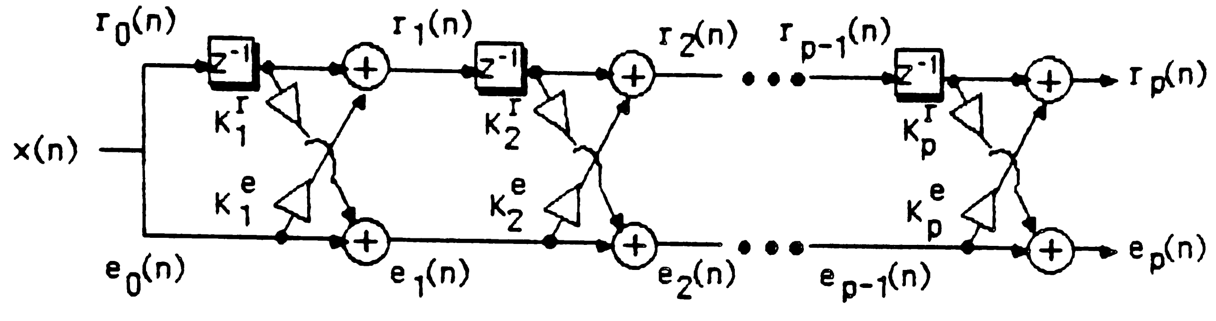

Figure 4

Lattice Filter Structure

This filter implemented as a lattice structure has several advantages over the standard tapped delay line (TDL) filter [19]. First, the p-stage lattice filter generates all outputs which could be generated by p different TDL filters of lengths from 1 to p. This would allow the dynamic assignment of the most effective filter length based on the input at any instant in time. The lattice filter also has the property that larger order filters are built up from smaller ones by simply adding more lattice stages. This should be useful in designing very large scale systems. Another advantage of the lattice structure in general is the lower sensitivity to digital arithmetic roundoff errors [20] than the convention TDL. This is always nice to know, since it means that a given filter might possibly be implemented with fewer bits. Finally, the lattice filter has been shown to have faster convergence properties than the LMS algorithm [16], and the rate of convergence is not related to the eigenvalue spread of the input correlation matrix.

All adaptive lattice algorithms aim to minimize some error condition, and in doing so to calculate the reflection coefficients K^e and K^r. There are two main ways of doing this.

The first method involves finding the set of reflection coefficients that minimizes the sum of the mean squares of the forward and backward error residuals for each stage of the lattice. This is done by a gradient method similar to the LMS algorithm and is known in the literature as the Gradient Lattice (GL). If the input data is assumed to be stationary, the result is reflection coefficients such that $K^e = K^r = K$ [16]. The other method involves finding the set of reflection coefficients that minimizes the sum of the squares of the forward and backward error residuals. Because this method does not make the assumption that the data is stationary, K^e is not necessarily the same as K^r. Actual results with distribution power line data show that K^e and K^r are in fact not equal, though they differed only slightly. Minimizing the sum of squares of the errors is similar to what is done in the Kalman algorithm, and the lattice algorithm which does this is known as the Least Squares Lattice, and is seen to be a special case of the Kalman filter. It is the Least Squares Lattice that is used in the algorithm comparisons in this thesis, and more detail about its derivation is given below. An excellent summary of lattice filter derivations and their histories can be found in [11].

Thus it can be seen that all adaptive filter algorithms mentioned in the literature can be divided into four categories. There are those which calculate the transversal filter coefficients directly and those which calculate the reflection coefficients of the lattice structure directly. In each of these two categories, either the gradient method or the least squares method can be used, and the result is the LMS, Kalman, Gradient Lattice, and Least Squares Lattice, respectively.

2.3.2 Algorithm

As mentioned above, the LSL algorithm, at every time n, seeks to minimize the sum of the squares of the prediction error up to time n. This is done by using the lattice structure as shown in figure 4, and exploiting the orthogonality properties of the forward and backward error sequences e_i and b_i. An exponentially decaying weighting of the squared errors can also be used to cause the LSL algorithm to track nonstationary input. Thus the prediction error to be minimized is

$$E^e(n) = \sum_{k=1}^{n} (1-\alpha)^{n-k} e(n)^2 \tag{2.23}$$

and

$$E^r(N) = \sum (1-\alpha)^{n-k} r(n)^2 \tag{2.24}$$

where $0 < \alpha < 1$ is the "fade factor" of the algorithm.

The LSL algorithm is summarized as follows [16], with the following quantities defined.

r_i	Backward error residual
e_i	Forward error residual
$E^r{}_i$	Backward error power at stage i
$E^e{}_i$	Forward error power at stage i
K^e, K^r	Reflection coefficients
Δ	Cross correlation of forward and backward prediction errors
B	Likelihood variable

<u>LSL Algorithm</u>

Initialization $(i = 0, 1, \ldots, p)$:

$r_i(-1) = 0, \qquad i \neq p$

$E^r{}_i(-1) = \partial, \quad \partial$ positive but close to 0; $i \neq p$

$\Delta_i(-1) = 0, \qquad i \neq 0$

Time Update $(n = 0, 1, \ldots, \infty)$:

$$e_0(n) = r_0(n) = x(n)$$

$$E^e{}_0(n) = E^r{}_0(n) = (1 - \alpha)\, E^r{}_0(n-1) + x(n)^2$$

$$\beta_{-1}(n-1) = 0$$

Order Update $(i = 1, 2, \ldots, p)$

$$\Delta_i(n) = (1 - \alpha)\, \Delta_i(n-1) - [e_{i-1}(n)\, r_{i-1}(n-1) \,/\, (1 - \beta_{i-2}(n-1))\,]$$

$$K^e{}_i(n) = \Delta_i(n) \,/\, E^e{}_{i-1}(n)$$

$$K^r{}_i(n) = \Delta_i(n) \,/\, E^r{}_{i-1}(n-1)$$

$$r_i(n) = r_{i-1}(n-1) + K^e{}_i(n)\, e_{i-1}(n)$$

$$e_i(n) = e_{i-1}(n) + K^r{}_i(n)\, r_{i-1}(n-1)$$

$$E^r{}_i(n) = E^r{}_{i-1}(n-1) - \Delta_i(n)^2\;\; E^e{}_{i-1}(n)$$

$$E^e{}_i(n) = E^e{}_{i-1}(n) - \Delta_i(n)^2 \,/\, E^r{}_{i-1}(n-1)$$

$$\beta_{i-1}(n-1) = \beta_{i-2}(n-1) + r_{i-1}(n-1)^2 \,/\, E^r{}_{i-1}(n-1)$$

2.4 Fast Kalman Adaptive Digital Filter

2.4.1 History

In the early 1960's, Kalman [21] and Kalman and Bucy [22] cast the linear prediction problem as a recursive solution to a set of linear difference equations which defined the system. This so-called Kalman filtering method was applied by Godard [23] to channel equalization, and in that paper he showed that the Kalman algorithm had a faster speed of convergence than the gradient based algorithms in use at the time, but the algorithm required the calculation of a p X p matrix at each time n, and therefore required $O(p^2)$ operations at each iteration. Morf and Ljung [24], and Falconer and Ljung [25] later expanded Kalman's algorithm by using the minimization of the sum of squared errors as the error criterion, and exploited certain shifting properties of the autocorrelation matrix to reduce the number of operations for the Kalman algorithm to O(p). This formulation is known in the literature as the Fast Kalman algorithm, and is mathematically equivalent to the original algorithm proposed by Kalman and Bucy. It is the Fast Kalman algorithm that is used in the comparisons made in this thesis.

2.4.2 Algorithm

As mentioned above, the Fast Kalman algorithm seeks to find a set of filter coefficients at time n which will predict the desired response d(n) of the system from past values of the input x(n) in such a way as to minimize the total prediction error. The prediction error is derived from the known response d(n) and is given by

$$e(n) = d(n) - A_p(n-1)^T x_p(n) \tag{2.25}$$

Here $x_p(n)$ is the vector of p previous inputs (p being the filter order). Subscripts on variables will indicate their dimensions. The Fast Kalman algorithm calculates the coefficient vector $A_p(n)$ at time n which minimizes the cumulative squared error up to that time:

$$\sum_{k=1}^{n} [d(k) - A_p(n)^T x_p(k)]^2 \tag{2.26}$$

As mentioned in the section on Lattice filters, the minimum coefficient vector is the solution to the Wiener-Hopf equation, which is

$$A_p(n) = R_{pp}(n)^{-1} \left[\sum_{k=1}^{n} d(k) \; x_p(k) \right] \tag{2.27}$$

where

$$R_{pp}(n) = \sum_{k=1}^{n} x_p(k) \, x_p(k)^T + \delta \tag{2.28}$$

or the estimated covariance matrix. The parameter δ is a small positive constant which is used in practice to insure the nonsingularity of $R_{pp}(n)$. The Kalman algorithm as shown by Godard [23] then calculates the coefficient vector $A_p(n)$ recursively as

$$A_p(n) = A_p(n-1) + k_p(n) \, e(n) \tag{2.29}$$

where

$$k_p(n) = R_{pp}(n)^{-1} x_p(n) \tag{2.30}$$

The point of the "fast" Kalman algorithm is to calculate the Kalman gain vector $k_p(n)$ recursively, without requiring the inversion of $R_{pp}(n)$. The algorithm takes advantage of the fact that at time n the vector $x_p(n)$ does not get p new elements, but some number much less than p, in this case one. This new element at time n will be designated i(n) and is x(n). At the same time one element is discarded from $x_p(n-1)$ to form $x_p(n)$. This element shifted out will be designated o(n). This shifting property of $x_p(n)$ is used to derive an algorithm which is mathematically equivalent to the equations above, but which recursively calculates the Kalman gain vector $k_p(n)$ and the inverse of the autocorrelation matrix $R_{pp}(n)$.

The equations above assume that the input data is stationary, but in real world situations the data is often slowly varying with time. It is possible to cause the algorithm to track time variations of the statistics of the data by causing it to "forget" errors from the distant past. This is done by introducing an exponentially decaying memory factor α in the squared error calculation as in the lattice method. Thus

$$\sum_{k=1}^{n} \alpha^{n-k} [\, d(k) - A_p(n)^T x_p(k)]^2 \tag{2.31}$$

where α is a positive number less than or equal to 1. The updated coefficient vector then becomes

$$A_p(n) = R_{pp}(n)^{-1} [\, \sum_{k=1}^{n} \alpha^{n-k} d(k)\, x_p(k)] \tag{2.32}$$

where

$$R_{pp}(n) = \sum_{k=1}^{n} {}^{n-k}\, x_p(k)\, x_p(k)^T + \partial \qquad (2.33)$$

and the same formula for the Kalman gain vector k_p above applies.

The fast Kalman algorithm for calculating the Kalman gain vector $k_p(n)$ without requiring the inversion of $R_{pp}(n)$ directly is given below, and the derivation of the algorithm making use of the shifting property of the input can be found in [25]. The subscripts on the variables indicate the size of the matrices.

Fast Kalman Algorithm

Initialization:

$$B_p(0) = D_p(0) = 0_p$$

$$E = \partial$$

$$k_p(1) = 0_p$$

$$x_p(n) = 0 \ \text{ for } n \le 0$$

Time update ($n=1$ to ∞):

$$er(n) = i(n) + B_p(n-1)^T x_p(n)$$

$B_p(n) = B_p(n-1) - k_p(n)\ er(n)$

$er(n)' = i(n) + B_p(n)^T\ x_p(n)$

$E(n) = \alpha E(n-1) + er(n)'er(n)$

k_M' is constructed by making the first element be

$er(n)'/E(n)$

and the last p elements to be

$K_p(n) + B_p(n)er(n)'/E(n)$

The first p elements of k_M' are then taken to be $m_p(n)$ and the last element of k_M' is taken as $\mu(n)$.

$c(n) = o(n) + D_p(n-1)^T\ x_p(n+1)$

$D_p(n) = [D_p(n-1) - m_p(n)\ c(n)] / [1-\mu(n)\ c(n)]$

$k_p(n+1) = m_p(n) - D_p(n)\ \mu(n)$

The filter coefficients are then updated as in Eq. 2.29

$A_p(n+1) = A_p(n) + k_p(n+1)\ e(n+1)$

The complexity of this algorithm is much greater than the LMS algorithm, taking O(10p) multiplications and O(12p) additions per input sample. The advantage of this algorithm over the LMS, however, is that the filter can more rapidly track changes in the statistics of the input x(n). This is advantageous in some situations, but in the case of distribution power line noise, the Fast Kalman algorithm can actually track the input too fast, resulting in false estimation of the harmonic noise in the input. This aspect of the Fast Kalman alogorithm is discussed in the section on results.

Chapter III

Implementation

3.1 Introduction

The three algorithms compared in this thesis were chosen because of their wide representation in the literature. Most results in the literature have been generated from simulated data, with a few reports of implementation on actual speech data. In this thesis the algorithms were run on actual distribution power line noise data, so the results are directly applicable to distribution power line carrier systems.

3.2 Power Line Noise Sample

The data used to test the algorithms was gathered from a distribution substation of a local power company. The data was gathered by low-pass filtering the analog data and then sampling at a rate of 40 kHz. An analog high pass filter was also used to eliminate the 60 Hz power frequency and all harmonics below 1 kHz before sampling. Approximately 1.6 seconds of data (65,536 samples) was gathered by the sampling procedure. This represents a good sample of data that is within the frequency range of all known power line carrier systems (that do not use 60 Hz transmission). It is obviously a very small sample of all possible power line noise; however, it is sufficient to examine the effect of the adaptive algorithms on power line noise.

In all of the experiments run, the original data was first demodulated to bring the 100th 60 Hz harmonic (6 kHz) down to the baseband, and resampled at a ratio of 16-to-1. This allows the length of the filters to be reduced by the same 16-to-1 ratio to produce the same results as filters on the original noise.

3.3 Algorithms

The three algorithms, LMS, LSL, and Fast Kalman, were coded in the C programming language and run on a Vax 11/780 under 4.2 BSD UNIX™. A filter length was chosen (either 128 or 256 taps) and the filters were allowed to adapt to the power line noise for a specified number of samples. A version of the data delayed by m samples was used as the input x(n), and the current sample n as d(n). In the LMS and Fast Kalman cases, the delay m was allowed to be specified, and the significance of its choice will be discussed later. In the LSL case, the algorithm is by definition a 1-step forward linear predictor, so the delay m is fixed by definition at 1. Other algorithm parameters, such as the fade factor and the small constant used to keep the matrices non-singular, were also specified and changed from experiment to experiment. The effect of the different algorithm parameters is discussed in the Results. The final filter weights were then stored for later use. The error output e(n) of the adaptation was plotted for future reference and for inclusion in the section on Results.

The stored filter weights were then use to filter the original (demodulated) noise data, using the delay m to specify the current input point and the value to be predicted, as shown above in figure 3. The power spectrum of the data was plotted before and after the filter was applied, which allowed the reduction in harmonic noise to be seen as the maximum signal power (in the frequency domain) before and after filtering. Comparison was then made of the plots generated from the different algorithms.

Reduction of the noise power alone is not a definitive measure of a filter's performance, since a filter could simply zero the input and thereby reduce the noise power. A better measure of the performance of a filter is how well it can be used to detect digital data buried in the noise. Simulation of phase-shift keyed (PSK) digital data buried in the demodulated power line noise data was used to test each filter.

3.4 Algorithm Parameters

Of particular importance to the correct functioning of all three adaptive algorithms are the various algorithm parameters, such as filter length, decorrelation delay, and allowed misadjustment. The three algorithms fall naturally into 2 groups for the purpose of discussing parameters: the LMS on one hand and the LSL and Fast Kalman on the other. However there are some parameters that are common to all three algorithms, and they will be discussed first.

The parameters common to all three algorithms are the filter length p, the length of data N on which the filter is allowed to learn, and the decorrelation delay Δ. The decorrelation delay functions differently in the 2 groups of algorithms, so it will be discussed separately for each group.

For best results, the number of taps of the filter must be long enough to span at least one period of the 60 Hz spike train, and preferably more than one. The typical number of taps used was either 128 or 256, allowing 3 and 6 periods of the 60 Hz spike train to be spanned. These seemed to work well, even though it can be shown that not all taps are necessary for the canceling of the 60 Hz harmonics, so that some can be constrained to be zero. Investigation of this case is a current research topic.

The length of data N on which the filter is allowed to learn is determined by the length of time needed for the algorithm to converge. This time is different for the three algorithms, and different depending on other parameters as mentioned below. In general, the LMS algorithm needs about 2500 samples (1 second) to converge, and the others need anywhere from 500 (.2 seconds) to 1500 (.6 seconds) samples, depending on other parameters. Many different combinations of parameters were tried with different values of N, and the results are summarized in the next chapter.

The LMS algorithm has two parameters unique to it: μ, the parameter affecting convergence rate and level of misadjustment, and Δ, the decorrelation delay. At very small values of μ, around .001, the LMS algorithm can take many seconds to converge. A value of 1.0 was used for most experiments, which allowed the filter to converge in about 2500 samples (1 second). The decorrelation delay has been found to be important to the LMS algorithm, with delays of more than one 60 Hz period (41 samples) required to adequately cause the LMS algorithm not to adapt to the white noise, which may have some short-term correlations. A decorrelation delay of 82 was used for most experiments.

The other two algorithms, LSL and Fast Kalman, have several parameters specific to them. They are α, the fade factor, Δ, the decorrelation delay, and δ, the small constant used to insure non-singularity of the autocorrelation matrix R. The parameter α is defined as α^n for the Fast Kalman algorithm and $(1-\alpha)^n$ for the LSL algorithm. Both will be called α here with no loss of generality.

The fade factor α has been shown in the literature not to be very critical, as long as it is close to 1. Values from .9 to .9999 were used in the experiments, with .999 and .9999 being used most often.

The decorrelation delay Δ was not found to be as critical in the LSL and Fast Kalman algorithms as it is in the LMS algorithm. The LSL algorithm, by definition, only allows a decorrelation delay of 1. The Fast Kalman algorithm, however, can use any decorrelation delay desired, and setting it to 1 caused the Fast Kalman algorithm to behave in the same manner as the LSL algorithm. Setting to something other than 1, such as the 82 used in the

LMS algorithm, did not seem to affect the performance of the Fast Kalman algorithm much, especially not as much as varying δ.

The small constant used to keep R from being non-singular, δ, was found to be one of the most critical parameters of the LSL and Fast Kalman algorithms. This parameter also determines the amount of misadjustment these algorithms will have at convergence. Very small values of δ, in the range of .001, caused the algorithms to converge so fast with so little misadjustment that they were unable to separate the 60 Hz harmonics from the white noise very well; that is, the stop bands were very broad. Figures 5 and 6 show the power spectrums of the LSL and Fast Kalman filters which were allowed to adapt with a δ value of .001. Note the lack of well defined spectral peaks at each 60 Hz harmonic, and the relatively high level of "noise" at the edges of the 60 Hz peaks that do exist.

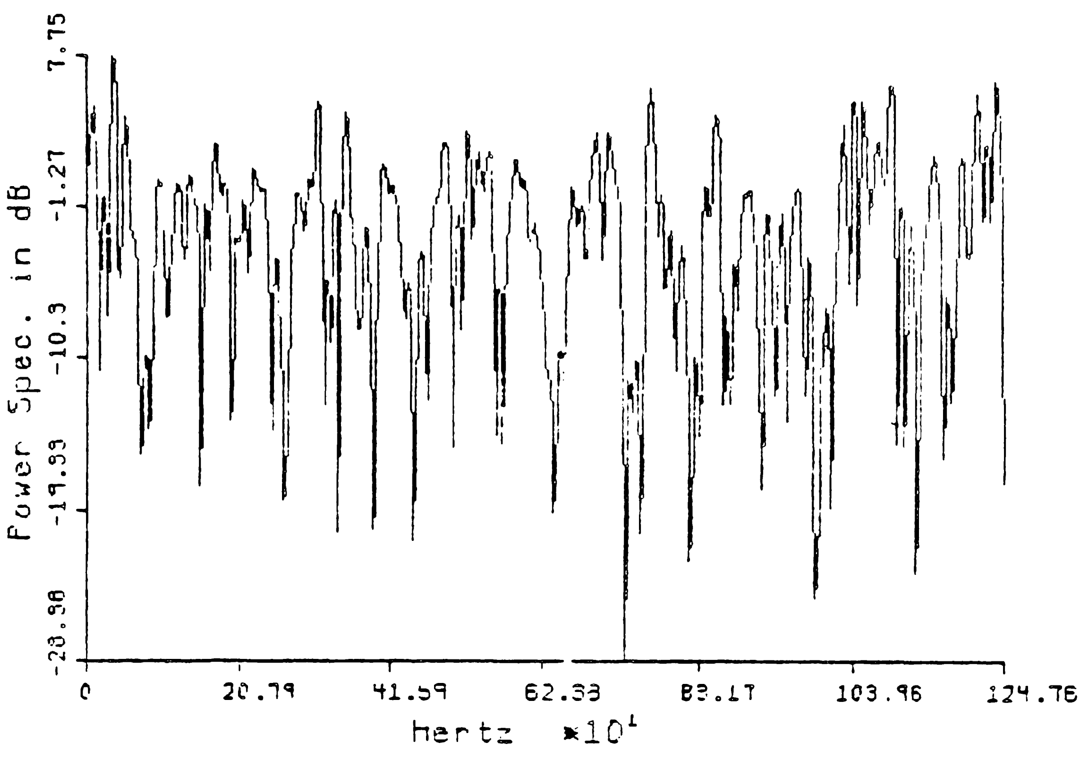

Figure 5

LSL Adaptation with ∂ value of .001

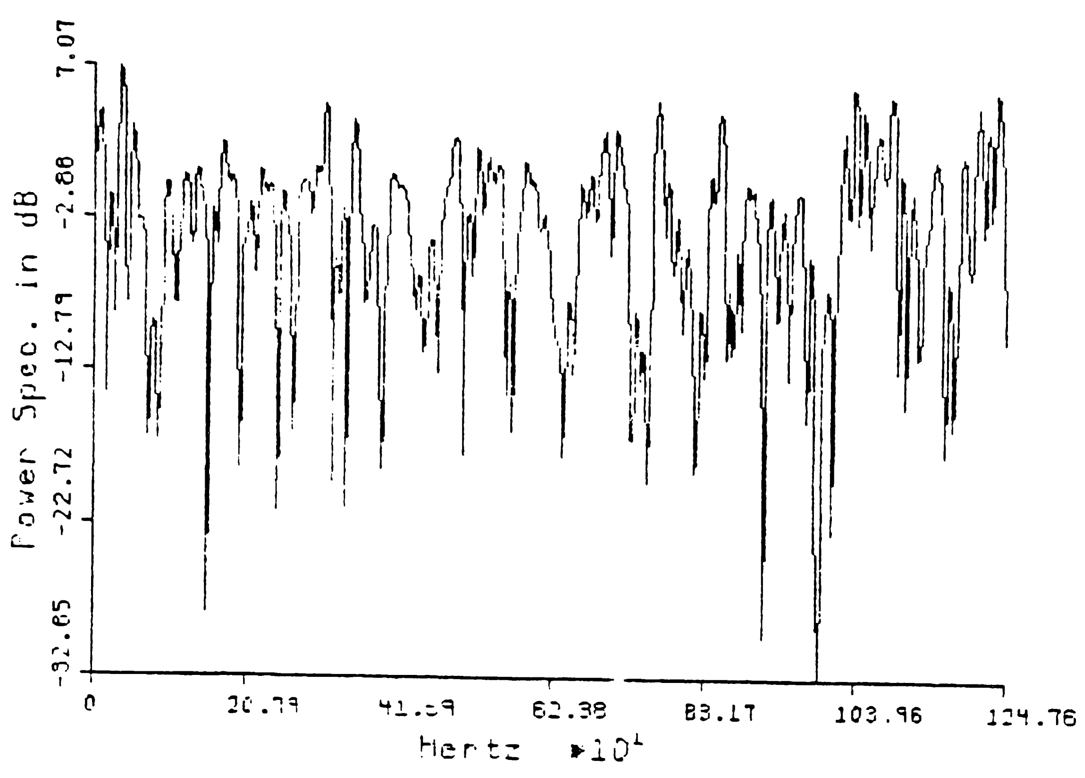

Figure 6

Fast Kalman Adaptation with ∂ value of .001

On the other hand, values of ∂ around 0.1 allowed the algorithms to separate the 60 Hz harmonics from the white noise, while slightly increasing the time of convergence, as will be shown in the Results section on convergence rates. Figures 7 and 8 show the power spectrums of the LSL and Fast Kalman filters which were allowed to adapt with a ∂ value of 0.1. Note the better resolution of the 60 Hz harmonics and the lower levels of "noise" at the edges of the peaks. The results of comparison of bit error rates of these two choices of ∂ are detailed in the Results.

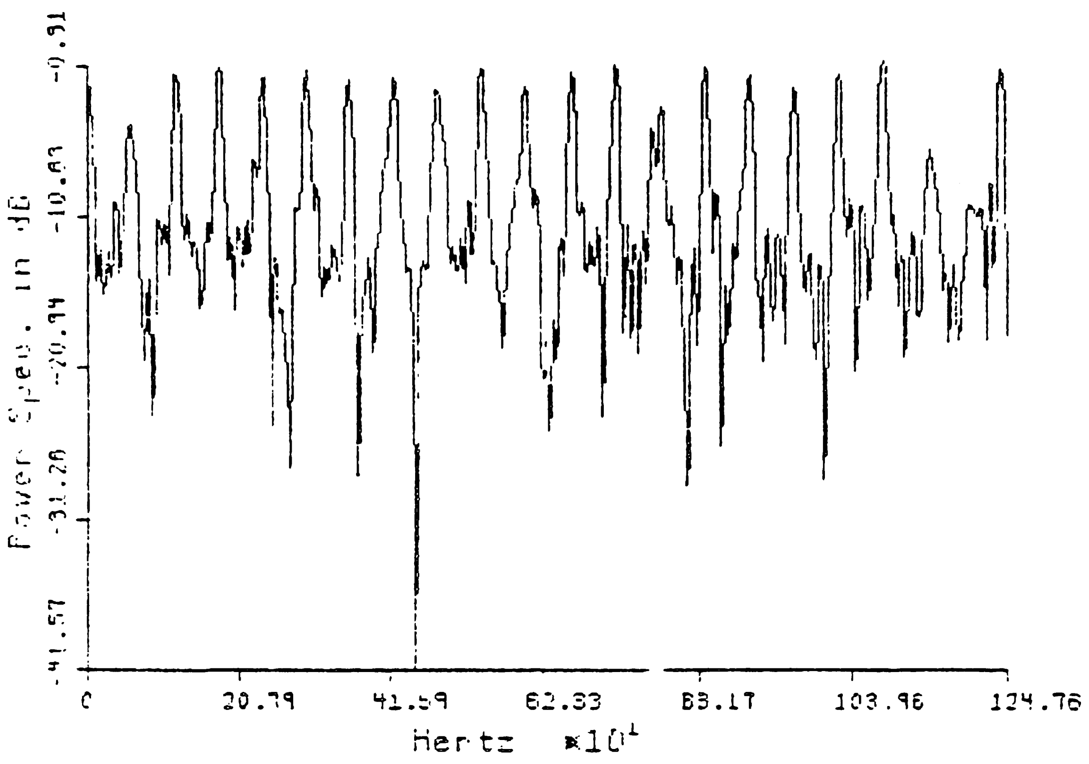

Figure 7

LSL Adaptation with δ value of .1

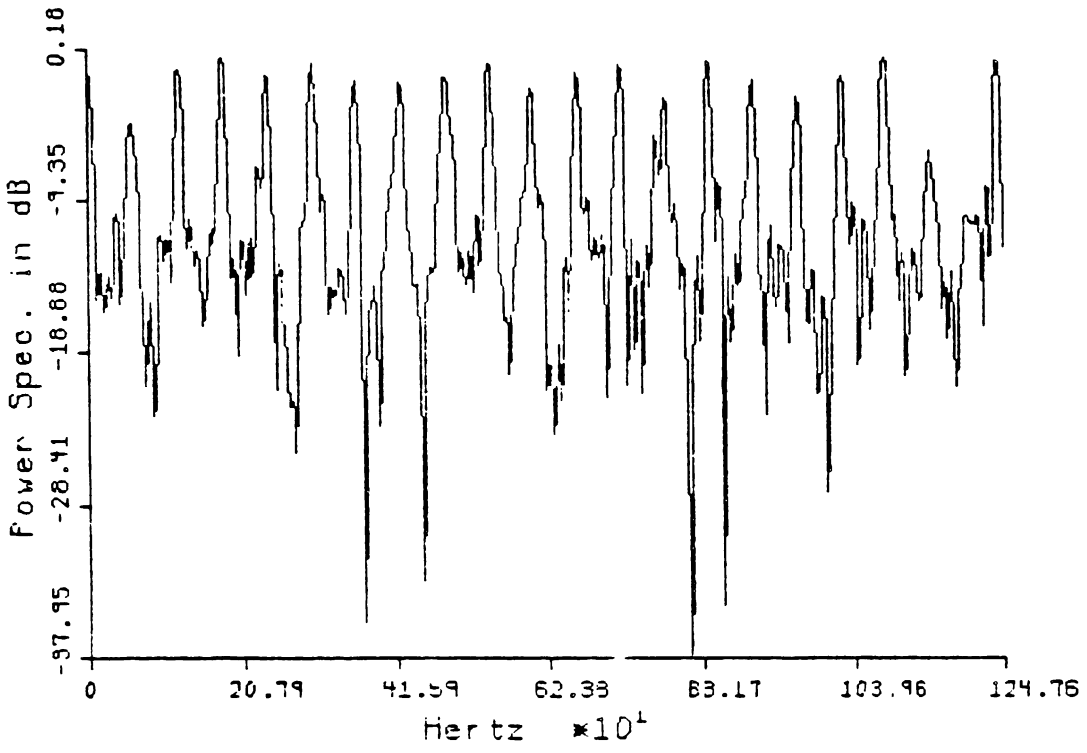

Figure 8

Fast Kalman Adaptation with ∂ value of .1

Chapter IV

Results

4.1 Algorithm Comparison Criteria

4.1.1 Rate of Convergence

Comparison of the rate of convergence of these adaptive algorithms is by nature subjective. The error residual is plotted while the filter is adapting, and by examining the plot, it can be noticed that the error will eventually reach a "steady-state" range of values. The variation in this range is dependent on the level of misadjustment that the algorithm is allowed based on the parameter μ in the LMS algorithm and the parameter ∂ in the LSL and Fast Kalman algorithms. The convergence times mentioned below were chosen by looking at the plots of the error residual and choosing the approximate spot at which the error appeared to reach this steady-state condition.

4.1.2 Reduction of Noise Power

After the algorithm had been allowed to adapt to the power line noise, the filter weights were frozen and then used to filter the original data. The power spectrum of the original data was then compared to the power spectrum of the filtered noise, and the difference in peak power between the two was taken as the total reduction of peak noise power. This reduction in peak power is not a complete measure of the performance of the filters, however, because the filter could behave in such a way as to filter out more than just the 60 Hz harmonics of the noise, in which case the noise power would be reduced, but signal power would also be reduced, which is undesirable. Since in most power line communications systems the signal transmitted is digital data, this reduction of the signal power could result in a higher bit detection error rate. Simulation of this condition is the subject of the next section.

4.1.3 Bit Error Rate

Once an algorithm was run on the distribution power line noise and filter coefficients obtained, the coefficients were saved and used as input into a program which simulated a digital signal in PSK format superimposed on the original (demodulated) noise. The program was used to add the PSK signal at the baseband frequency to the noise, and then the signal plus noise was filtered using the saved filter weights. A matched filter was then used to detect the bits after filtering, and the number of errors between the detected bits and the original bits was reported as the bit error rate (BER). This BER is used as a comparison criterion for the different algorithms because it shows how well the resultant adaptive filter attenuated only the harmonic noise, and not the signal. This is the situation of interest in distribution power line carrier communication systems.

4.2 Algorithm Results

Before mentioning the results of the algorithm comparisons, it is necessary to give information about the noise sample used in the experiments. Figure 9 shows the time domain plot of the first 1024 samples of the noise sample as gathered directly from the distribution power line. Note the periodicity of the spikes occurring at 60 Hz intervals (666 samples). Figure 10 is the time domain plot of the original data after demodulation and re-sampling. The 60 Hz spikes can now be seen even more clearly, since they occur only 41 samples apart. Figure 11 shows the power spectrum of the demodulated data. This will be used later in comparison of the reduction of noise power of the filters.

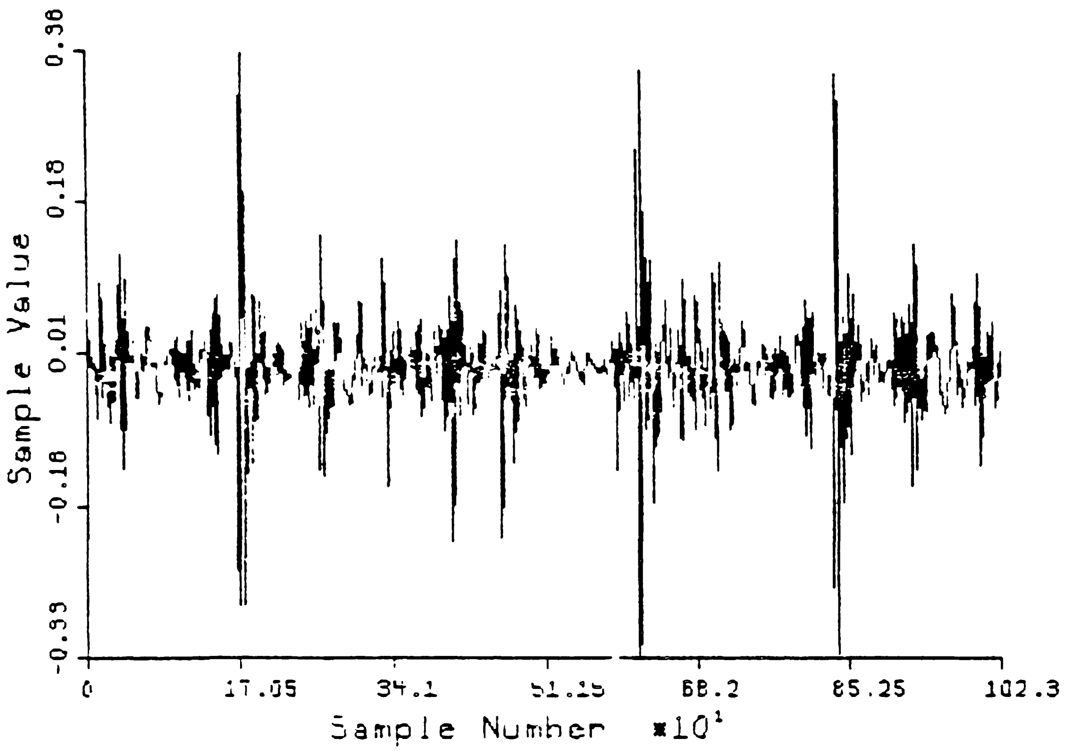

Figure 9

Original Distribution Power Line Noise

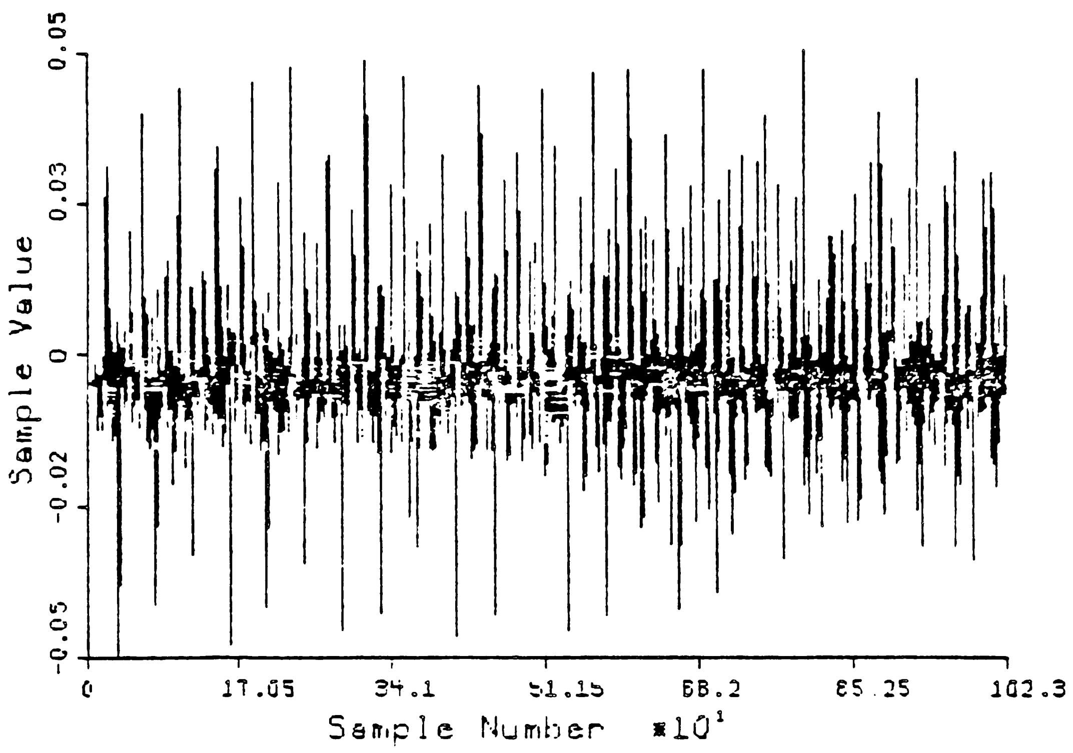

Figure 10

Demodulated and Re-sampled Power Line Noise

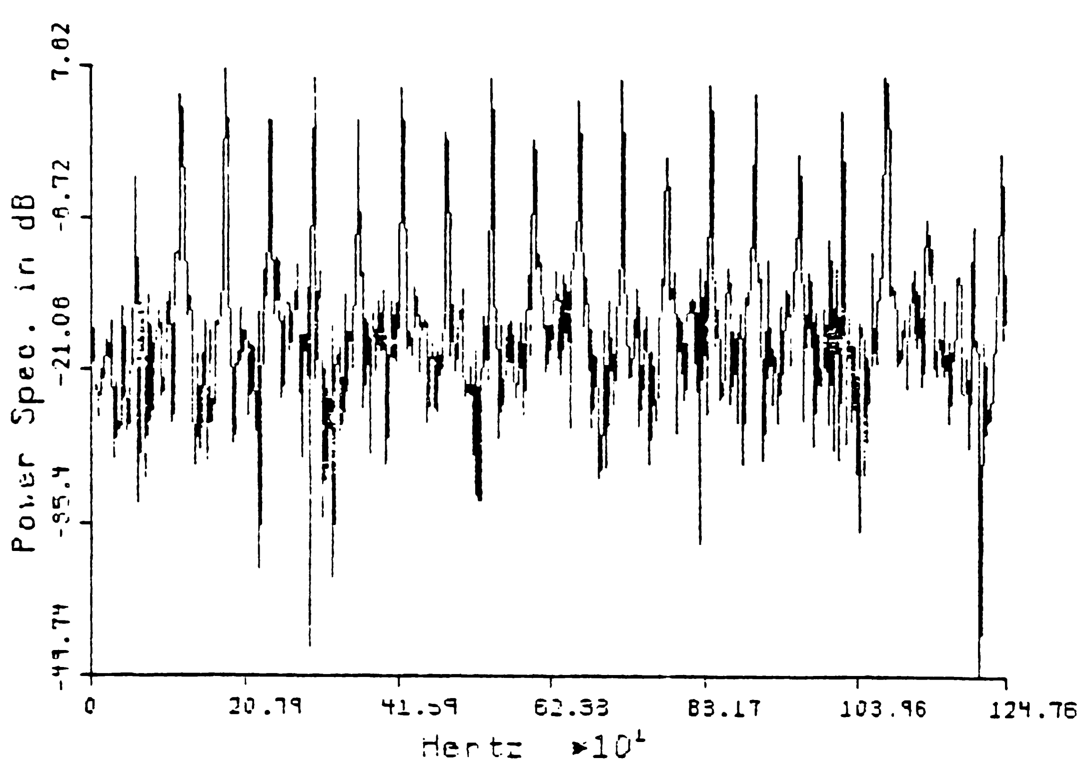

Figure 11

Power Spectrum of Demodulated Noise

4.2.1 Rate of Convergence

Much work has been done on the rate of convergence of the LMS algorithm, and it is well known that it does not converge as fast as the LSL or Fast Kalman algorithms. This known characteristic of the LMS algorithm was confirmed in the experiments run on the power line noise sample. Figures 12-14 show examples of the error residual during adaptation of all three algorithms. The long convergence time, about 2500 samples, of the LMS filter can be noted, as well as the relatively short convergence time of the other two algorithms. The plots shown are for adaptations which produced filters which gave approximately the same bit error rate. As can be seen, the LSL and Fast Kalman algorithms converge in about 30% of the time of the LMS algorithm.

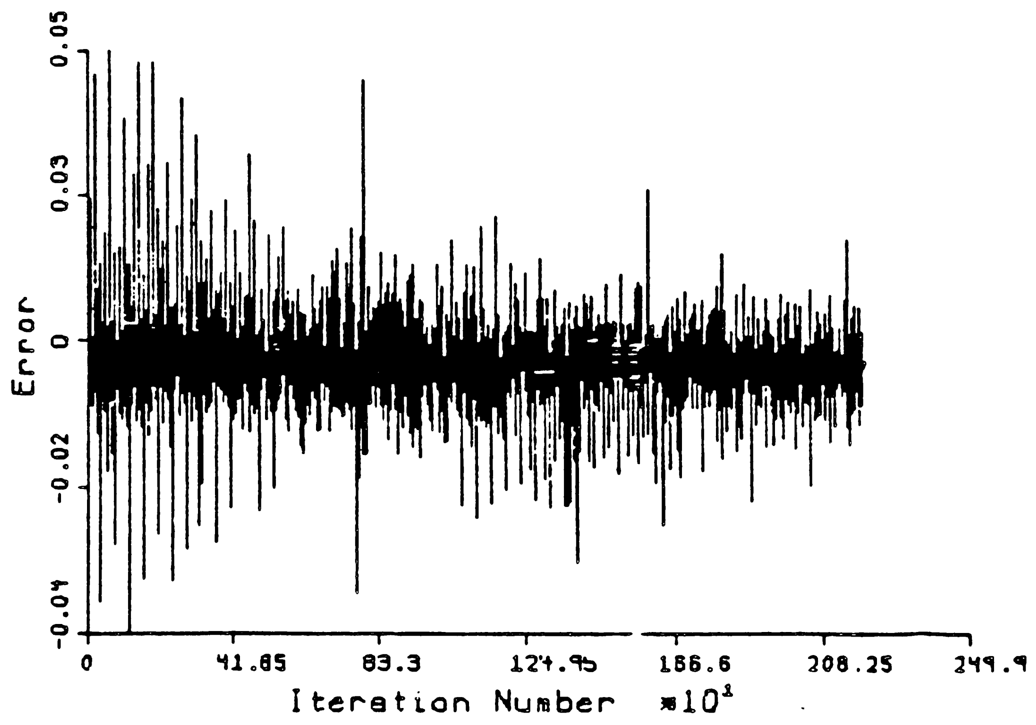

Figure 12

Convergence Plot, LMS

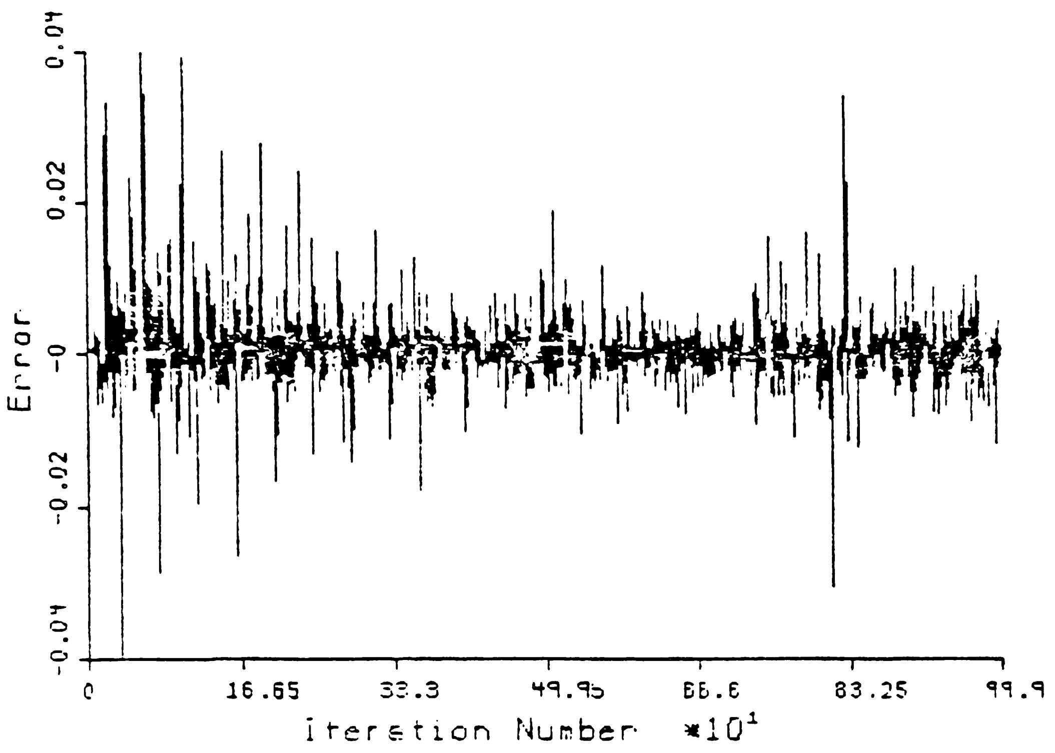

Figure 13

Convergence Plot, LSL, δ = .1

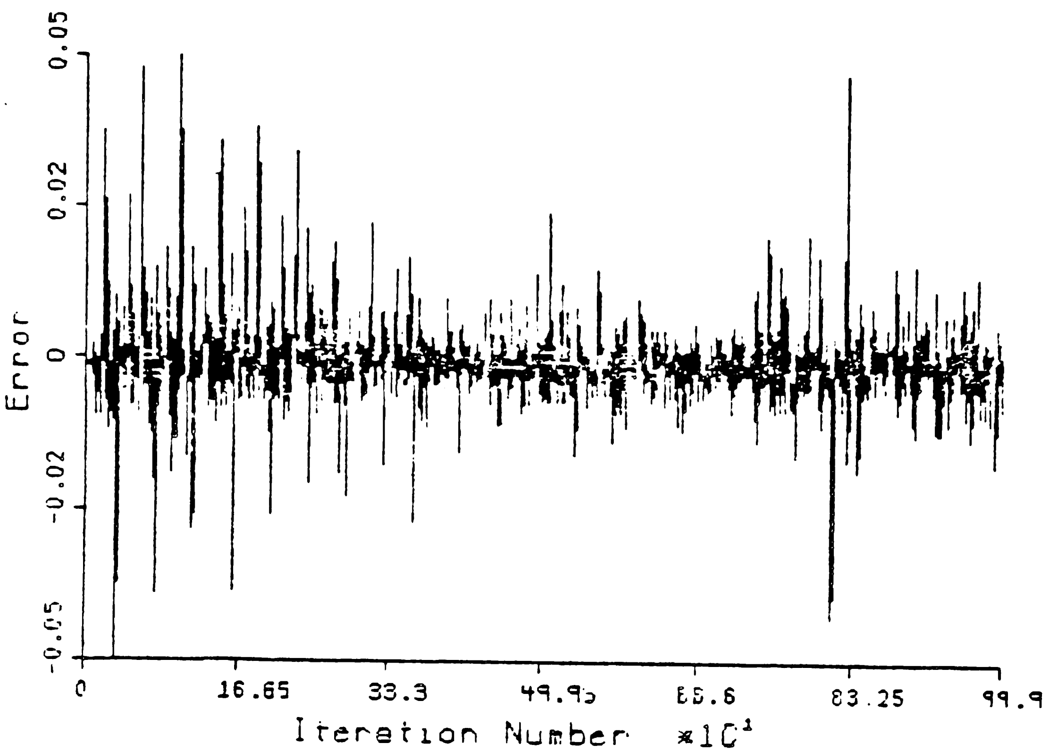

Figure 14

Convergence Plot, Fast Kalman, ∂ = .1

Of particular importance, as mentioned in section 3.4 above, is the change in the rate of convergence when different values of the misadjustment parameter ∂ are used in the LSL and Fast Kalman algorithms. Figures 13 and 14 above show the plot of the error residual for the LSL and Fast Kalman filters with a ∂ value of .1. The gradual "leveling off" of the error can be clearly seen. Figures 15 and 16 show the same filters with a ∂ value of .001. No gradual "leveling off" can be seen; the filter adapts so fast to the harmonic as well as white noise input that the error residual seems to be widely varying continually. As mentioned below, this low value of ∂ also causes the resultant filter not to pick out the 60 Hz harmonics very well, giving a filter that will attenuate any desired signal components, resulting in a higher bit error rate.

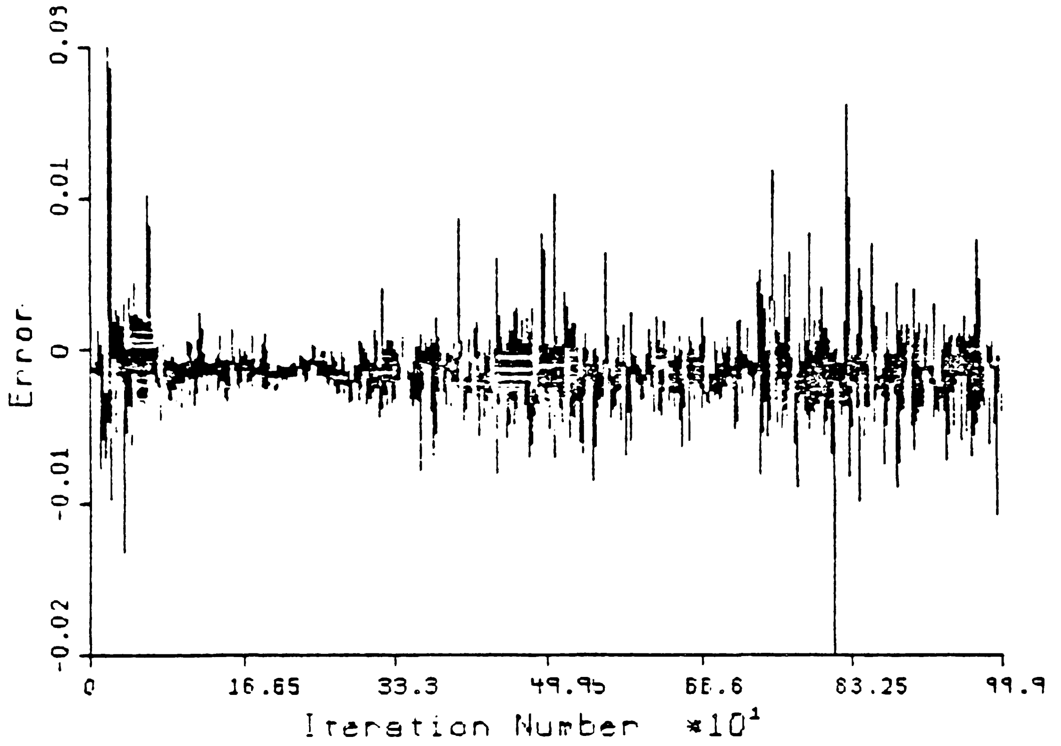

Figure 15

Convergence Plot, LSL, δ = .001

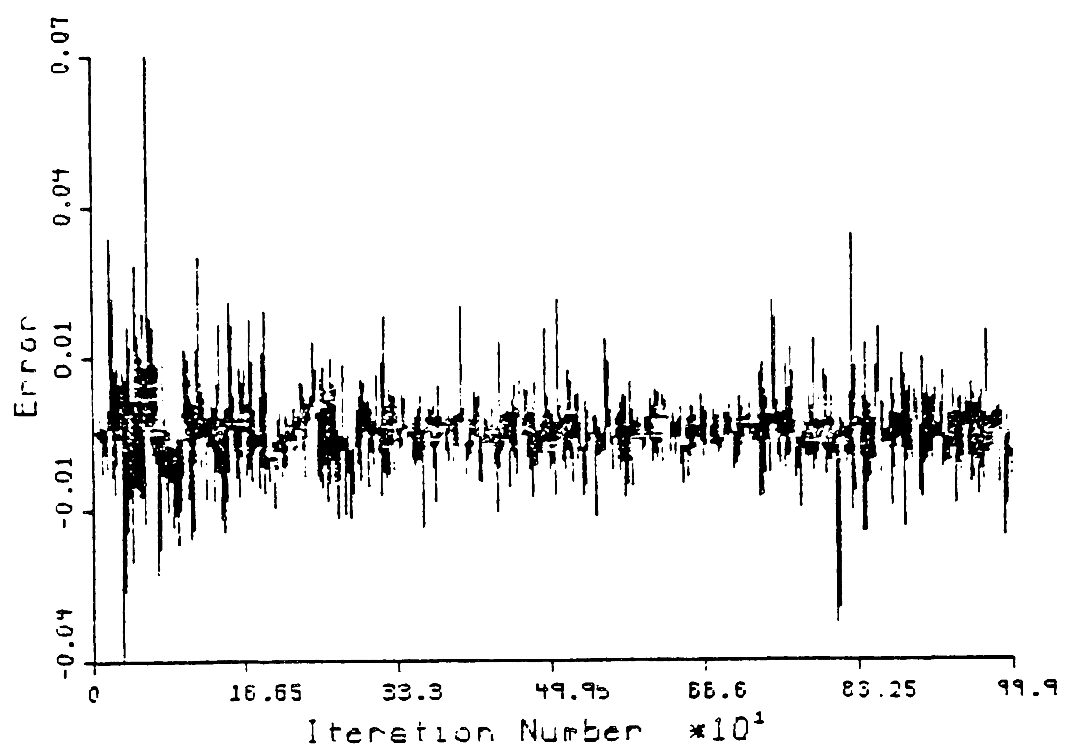

Figure 16

Convergence Plot, Fast Kalman, δ = .001

4.2.2 Reduction of Noise Power

All three algorithms produce approximately the same reduction in noise power when the optimal parameters for each is chosen. Figure 17 shows the power spectrum of the demodulated noise sample after being filtered with the LMS filter weights. The reduction in peak noise power was about 11 dB. Figures 18 and 19 show the same plots for the LSL and Fast Kalman algorithms.

These two filters were able to achieve a reduction of 16 dB and 15 dB, respectively.

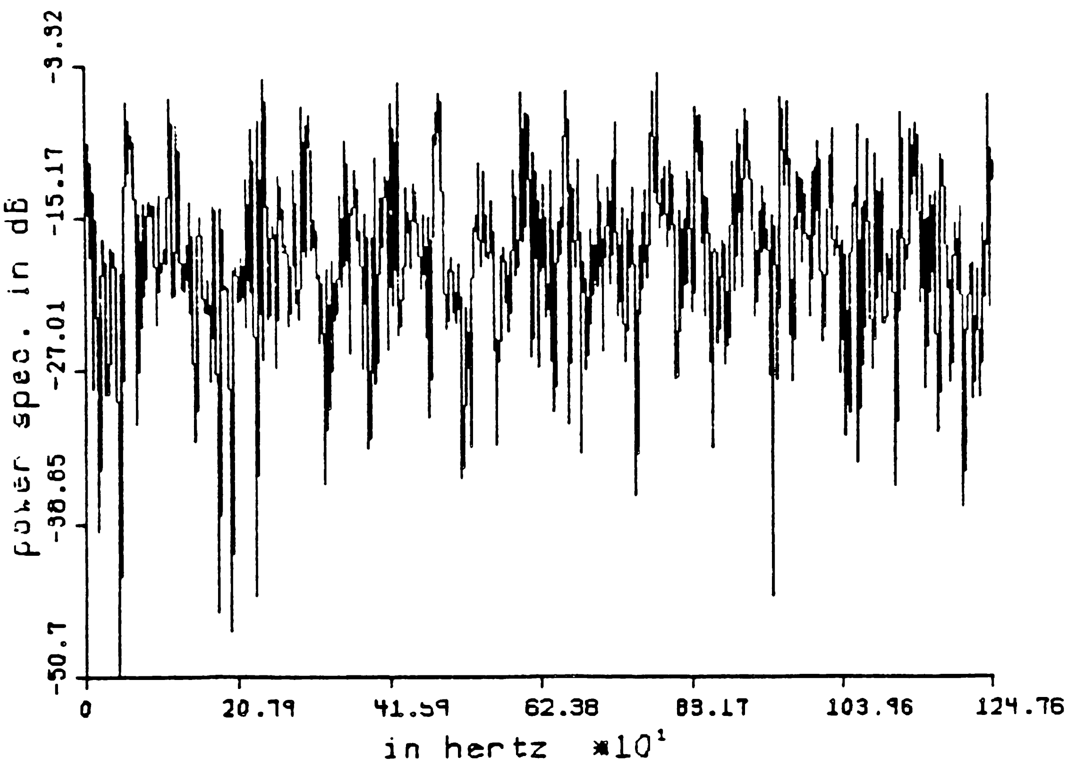

Figure 17

Noise Power Reduction, LMS

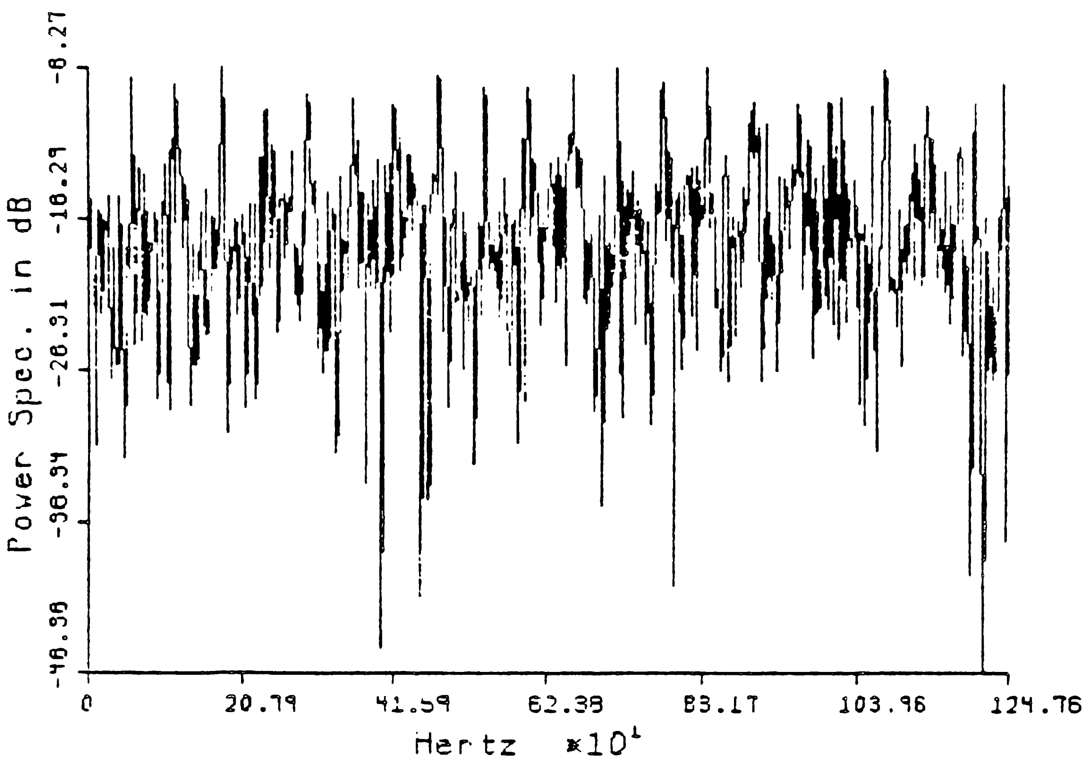

Figure 18

Noise Power Reduction, LSL

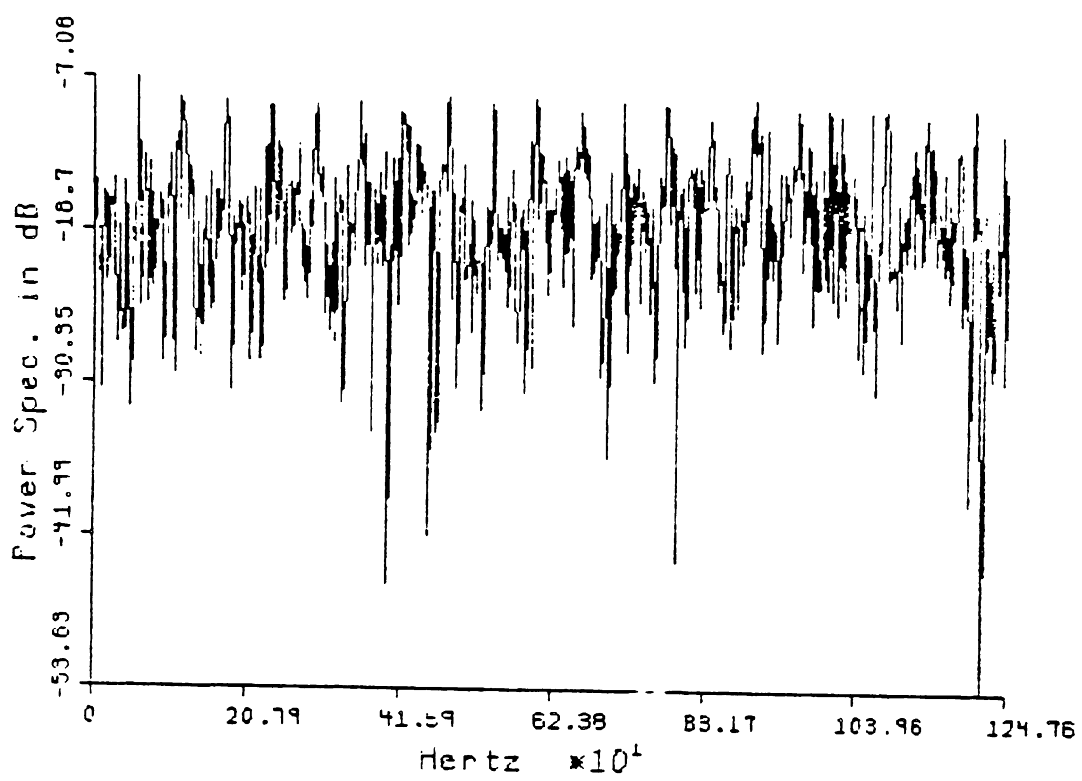

Figure 19

Noise Power Reduction, Fast Kalman

4.2.3 <u>Error Rate</u>

The bit error rate (BER) is perhaps the most significant measure of the performance of any filter on power line noise, either adaptive or fixed, since the objective of power line communications systems is to transmit and receive digital information. As a consequence, many experiments on the BER of the different algorithms were run. First the simulation of digital bit detection was run without any filtering for use as a reference for comparing the absolute performance of the filters. Four different signal-to-noise (SNR) ratios (RMS signal to RMS noise) were used: -24 dB, -27 dB, -30 dB, and -33 dB.

Since much work has been done on the LMS algorithm by Trussell and Wang, as well as others, only the best LMS adaptation was used in comparing the three filters. The parameters of the best LMS adaptation were a filter length of 256, adaptation time of 2500 samples, a decorrelation delay of 82 samples, and a convergence and misadjustment parameter of 1.0.

As mentioned above, the misadjustment parameter ∂ greatly influences the performance of the LSL and Fast Kalman algorithms. BER calculations were performed on LSL and Fast Kalman filters which were adapted using a ∂ of .001 and .1, each with a filter length of 128 and 256. The decorrelation delay used in the LSL algorithm was 1, and in the Fast Kalman algorithm was 82. The fade factor of both algorithms was set at .9999.

The results of all three algorithms, as well as the case of no filtering, are presented in the tables below. It can be easily seen that the misadustment parameter ∂ of the LSL and Fast Kalman algorithms has a great impact on the BER of the filter. It can also be noted that the LSL and Fast Kalman filters give performance very close to the LMS algorithm, yet with much faster convergence, as noted above.

Table 1

BER for PSK signal

No filtering and LMS filtering

SNR (dB)	Algorithm	BER (%)
-24	None	0.0
	LMS	0.0
-27	None	1.6
	LMS	0.0
-30	None	21.6
	LMS	3.8
-33	None	37.3
	LMS	23.9

Table 2

BER for PSK signal

LSL and Fast Kalman filtering, 256 taps

LSL			Kalman		
SNR (dB)	∂	BER (%)	SNR (dB)	∂	BER(%)
-24	.001	10.5	-24	.001	10.5
-27	.001	11.7	-27	.001	10.5
-30	.001	13.8	-30	.001	14.6
-33	.001	27.2	-33	.001	25.5
-24	.01	4.6	-24	.01	3.3
-27	.01	5.0	-27	.01	5.0
-30	.01	8.3	-30	.01	8.8
-33	.01	21.8	-33	.01	23.4
-24	.1	0.0	-24	.1	0.0
-27	.1	0.0	-27	.1	0.4
-30	.1	6.3	-30	.1	6.7
-33	.1	24.3	-33	.1	26.8
-24	1.0	0.0	-24	1.0	0.0
-27	1.0	0.4	-27	1.0	0.4
-30	1.0	10.9	-30	1.0	10.0
-33	1.0	30.1	-33	1.0	30.1

Table 3

BER for PSK signal

LSL and Fast Kalman filtering, 128 taps

LSL			Kalman		
SNR (dB)	∂	BER (%)	SNR (dB)	∂	BER(%)
-24	.001	3.2	-24	.001	3.2
-27	.001	4.0	-27	.001	4.0
-30	.001	8.5	-30	.001	8.5
-33	.001	26.7	-33	.001	25.9
-24	.01	1.6	-24	.01	1.6
-27	.01	3.6	-27	.01	3.6
-30	.01	8.1	-30	.01	7.7
-33	.01	24.7	-33	.01	24.7
-24	.1	0.0	-24	.1	0.0
-27	.1	1.2	-27	.1	0.4
-30	.1	6.1	-30	.1	6.1
-33	.1	24.7	-33	.1	25.5
-24	1.0	0.0	-24	1.0	0.0
-27	1.0	0.4	-27	1.0	0.4
-30	1.0	12.6	-30	1.0	13.0
-33	1.0	30.8	-33	1.0	30.8

Chapter V

Conclusions

5.1 Summary

In this thesis we have attempted to compare the performance of three different adaptive digital filters on distribution power line noise: the Least Mean Square (LMS), Least Square Lattice (LSL), and Fast Kalman algorithms. Studies have been done on the performance of these algorithms in other application areas, but little has been done on these algorithms in the power line noise environment. Comparison of the algorithms was done in three areas: rate of convergence, reduction of noise power, and reduction of bit error rate.

The LMS algorithm has been widely studied, and its characteristics are fairly well know. It was found that the LMS filter performs in a manner that would be expected from the literature in the power line noise environment. The filter is the slowest of the three to converge, but will give slightly better results than the other two if allowed time to converge completely.

The LSL and Fast Kalman algorithms were seen to perform almost as well as the LMS algorithm, while requiring much less time to converge to optimal performance. It was discovered that the parameter governing misadjustment is a critical parameter in the power line noise environment. The algorithms must be allowed a larger misadjustment and thus slower convergence than would be chosen from reading the literature in order to keep the filters from adapting to "perceived" short-term correlations in the white noise. Even with the slower convergence, however, the LSL and Fast Kalman algorithms converge at least twice as fast as the LMS algorithm, while giving similar bit error rate results.

5.2 Problem Areas

One of the major problems in producing a real-time filter using the LSL or Fast Kalman algorithm is the number of multiplications/divisions needed per input sample. The algorithms need O(10p)-O(13p) multiplications/divisions per input sample, and at high data rates this can be prohibitive. Some means of demodulating the input before adaptation could be used, as was done in the simulation here, or some form of multiprocessor implementation could be used to perform the arithmetic in the required amount of time. A low-cost solution to this problem will be necessary before these filters will be able to find widespread use in power line carrier communications.

Another problem area is the effect of arithmetic roundoff errors in the calculation of the filter coefficients. It is conceivable that roundoff errors could cause the filters to become unstable, giving useless results. Results from other sources seem to indicate that the LSL algorithm will be the least sensitive to this problem, but it is not known how well any of the algorithms will perform with limited word-size arithmetic.

5.3 Areas of Future Research

As mentioned above, arithmetic roundoff error is an area that will need research in order to determine it's effect on the stability and performance of the adaptive algorithms. Obviously, the smaller the word size needed to perform the arithmetic, the faster and cheaper the hardware can be. Thus some quantization of the effects of roundoff on the filters would be desirable.

Another area that could be addressed is the use of adaptive filters that assume an autoregressive-moving average (ARMA) model for the input noise. This would lead to an infinite impulse response (IIR) filter rather than an FIR filter, but a smaller number of coefficients could possibly give the same results as the FIR filter if the power line noise is more accurately modeled in this way rather than as an AR process. The adaptation calculations for an IIR filter will be at least as complex as for the FIR filters presented here, and probably more so, but the prospect of better performance with fewer coefficients is worth investigating.

References

1. Trussell, H. J., and Wang, J. D., "Cancellation of Harmonic Noise in Distribution Line Communications." To appear in IEEE Trans. Power App. and Systems, 1985.

2. Vines, Roger, "The Characterization of Residential Impedances and Noise Sources for Power Line Carrier Communications." Master's Thesis, North Carolina State University, Dept. of Elect. Eng., 1983.

3. O'Neal, J. B., Jr., "Modeling the Noise on the Substation Power Distribution Bus at Frequencies from 1-20 kHz." Paper of the Communication and Signal Processing Center, North Carolina State University, CCSP-WP-84/2, 1984.

4. Atal, B. S., and Hanauer, S. L., "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave." Journal of the Acoustical Society of America, vol. 50, no. 2, 1971, pp. 637-655.

5. Itakura, F., and Saito, S., "Analysis Synthesis Telephony Based on the Maximum Likelihood Method." Report of the 6th International Congress on Acoustics, Aug. 1968, pp. C17-C20.

6. Itakura, F., and Saito, S., "A Statistical Method for Estimation of Speech Spectral Density and Formant Frequencies." Electron. Commun. Japan, vol. 53-A, no. 1, 1970, pp. 36-43.

7. Makhoul, J., "Spectral Analysis of Speech by Linear Prediction." IEEE Trans. Audio and Electroacoustics, vol AU-21, June 1973, pp. 140-148.

8. Widrow, B., Mantey, P. E., Griffiths, L., J., and Goode, B. B., "Adaptive Antenna Systems.", Proc. IEEE, vol 55, no. 12, Dec. 1967, pp. 2143-21-59.

9. Bohlin, T., "Comparison of Two Methods of Modeling Stationary EEG Signals." IBM Journ. Res. and Dev., May 1973, pp. 194-205.

10. Gersch, W., "Spectral Analysis of EEG's by Autoregressive Decomposition of Time Series." Math. Biosci., vol. 7, 1970, pp. 205-222.

11. Freidlander, B., "Lattice Filters for Adaptive Processing." Proc. IEEE, vol. 70, no. 8, Aug. 1982, pp. 829-867.

12. Swanson, D. C., "All-Zero and Pole-Zero Least Squares Lattice Filter Structures for Adaptive Processing of Complex Acoustic Data." Masters Thesis, Pennsylvania State University, 1984.

13. Widrow, B., Glover, J. R. Jr., McCool, J. M., Kaunitz, J., Williams, C. S., Hearn, R. H., Zeidler, J. R., Dong, E. Jr., and Goodlin, R. C., "Adaptive Noise Cancelling: Principles and

Applications." Proc. IEEE, Vol. 63, No. 12, Dec. 1975, pp. 1692-1716.

14. McCool, J. M. and Widrow, B., "Principles and Applications of Adaptive Filters: A Tutorial Review." Naval Undersea Center - Dept. of the Navy, NUC TP 530, March 1977.

15. Messerschmitt, D. G., "Echo Cancellation in Speech and Data Transmission." IEEE Jour. Select. Areas Commun., Vol. SAC-2, No. 2, March 1984, pp. 283-297.

16. Hodgkiss, W. S. Jr., and Presley, J. A. Jr., "Adaptive Tracking of Multiple Sinusoids Whose Power Levels are Widely Separated." IEEE Tran. Acous. Speech Sig. Proc., Vol. ASSP-29, No. 3, June 1981, pp. 710-721.

17. Levinson, N., "The Wiener RMS (root-mean-square) Error Criterion in Filter Design and Prediction." Jour. Math. Phys., Vol. 25, 1947, pp. 261-278.

18. Durbin, J., "The Fitting of Time Series Models." Rev. L'Institut Intl. de Statisque, Vol. 28, 1960, pp. 233-243.

19. Satorius, E. H., and Pack, J. D., "Application of Least Squares Lattice Algorithms to Adaptive Equalization." IEEE Trans. Commun., Vol. COM-29, No. 2, Feb. 1981, pp. 136-142.

20. Markel, J. D., and Gray, A. H. Jr., "Roundoff Noise Characteristics of a Class of Orthogonal Polynomial Structures."

IEEE Trans. Acoust., Speech, and Signal Processing, Vol. ASSP-23, Oct. 1975, pp. 473-486.

21. Kalman, R. E., "A New Approach to Linear and Prediction Problems." Jour. Basic Eng., Trans. ASME, Vol. 82, 1960, pp. 35ff.

22. Kalman, R. E., and Bucy, R. S., "New Results in Linear Filtering and Prediction Theory." Journ. Basic Eng., Trans. ASME, Vol. 95, 1961, pp. 107ff.

23. Godard, D., "Channel Equalization Using a Kalman Filter for Fast Data Transmission." IBM Jour. Res. Develop., May 1974, pp. 267-273.

24. Morf, M., and Ljung, L., "Fast Algorithms for Recursive Identification." IEEE International Symposium on Info. Theory, June, 1976, Ronneby, Sweden.

25. Falconer, D. D., and Ljung, L., "Application of Fast Kalman Estimation to Adaptive Equalization." IEEE Trans. Commun., Vol. COM-26, No. 10, October 1978, pp. 1439-1446.

Appendix

LSL and Fast Kalman Filters

The following are the listings of the subroutines used to generate the results discussed in this thesis. There is one subroutine for LSL adaptation and one for Fast Kalman. The routines are written in C and were compiled and run on 4.2 BSD Unix™.

LSL Routine

```
/*****************************************************************************
c
c
c
c
c                        Least Squares Lattice
c
c                       Adaptive Digital Filter
c
c                     Constrained or Unconstrained
c
c
c
c  Calling convention:
c
c       lsl_init is called once to set up the data structures and then
c         lsl_pt is called once for each data point.
c
c
c       lsl_init(d, p, alpha, epsilon, tapper, var)
c
c  Parameters:
c
c       d                  :      pointer to lsl structure defined in lsl.h
c
c       p                  :      Filter order; number of weights.  INT
c
```

```
c       alpha           :       Fade factor.  Must be >0.0 but <1.0.
c
c       epsilon         :       Convergence value.  Small FLOAT number.
c
c       tapper          :       Period of taps that are not constrained.
c                                 The first period is assumed to start at
c                                 tap 0. 0 -> no constraining.  INT.
c
c       var             :       Number of taps before AND after the
c                                 unconstrained taps that are not to be
c                                 constrained.  INT.
c
c
c
c
c
c       lsl_pt(data, output, d, wt, freeze)
c
c
c  Parameters:
c
c
c       data            :       Input data point.  FLOAT.
c
c       output          :       Error value at this point.  FLOAT. Output only.
c
c       d               :       "Correct" data value at this point.  FLOAT.
c                               Input only.
c
c       wt              :       Filter weights at this point. P elements long,
c                               OUTPUT ONLY.  FLOAT.
c
c       freeze          :       Flag to indicate when filter weights are to be
c                               calculated. 1-> calculate.  INT.
c
c
c
c       Notation and variable names are taken from the paper by
c       Hodgkiss and Presley, IEEE Transactions on Acoustics, Speech, and
c       Signal Processing, Vol. ASSP-29, No. 3, June 1981  pp.710-721
c
c
c
c
c  Since only the values at time N and N-1 are needed for any variables,
c       N % 2 is used to point to the correct place in the arrays
c
c
c
c
```

```
c  Restrictions:
c
c       filter may not be of order > 512;  all arrays are sized to this
c
c
c  Written by:            James H. Wiggs
c
c
c*******************************************************************************
*/

#include <math.h>
#include <stdio.h>
#include </ncs/jhw/lsl/1pt/lsl.h>



lsl_init(d,p,alpha,epsilon,tapper,var)

        float   alpha,epsilon;
        int     p,tapper,var;
        struct lsl_data *d;
{

        int     i,j,k;


        if(p > MAX_ORDER) return(0);

/*
  store values away for later use
*/
        d->p = p;
        d->alpha = alpha;
        d->n = -1;              /* no input yet */
        d->tapper = tapper;
        d->var = var;

/*
c
c  init; mod(-1,2) = 1
c
*/
        for(i=0;i<p;i++) {
          d->r[i][1] = 0.0 ;
        }

        for(i=0;i<p;i++) {
          d->esupr[i][1] = epsilon ;
        }
        for(i=1;i<=p;i++) {
          d->delta[i] = 0.0 ;
        }

        for(i=1;i<p;i++) {
          for(k=0;k<i;k++) {
            d->b[k][i][1] = 0.0 ;
          }
        }

        return;

}



#define I (i+1)
#define MINUS1 0
        float   a[MAX_ORDER+1][MAX_ORDER+1];


lsl_pt(data, output, d, wt, freeze, pred, ke, kr)

        float   data,    /* input point */
                *output, /* output point */
                wt[],    /* filter coefficients */
                pred;    /* value to predict */
        struct lsl_data *d;  /* lsl data from init call */
        int     freeze; /* flag to indicate weight calculation. 1->calculate */
                        /*  weights; 0 -> do not calculate weights */
        float   ke[],kr[];    /* Reflection coefficients */
{

        double  sum_a, sum_b ;
        int     i, j, k, n, p, place;

        float   e, esupe, alpha, ksupe_sav[MAX_ORDER+1], ksupr_sav[MAX_ORDER+1];
        float   delta[MAX_ORDER+1];
        float   r[MAX_ORDER+1], forw[MAX_ORDER+1];
        float   rn[MAX_ORDER+1], en[MAX_ORDER+1];
        float   ksupe, ksupr;

/*
  use this input value to generate new filter coefficients
*/
        p = d->p;

        d->n++;
        n = d->n;
        d->input[n % p] = data;

/*
  Set up (abs(n-1) % 2)
*/
        place = abs(n-1) % 2;
        alpha = d->alpha;
        for(i=1;i<=p;i++)  delta[i] = d->delta[i];

/*
c
c  init first stage of lattice
*/
          e = data;
          d->r[0][n % 2] = data;
          esupe = (1.0-alpha)*d->esupr[0][place]
                                    +data*data;
          d->esupr[0][n % 2] = esupe ;
          d->gamma[MINUS1][place] = 0.0 ;
/*
c
c  perform update of values of lattice for each stage of the lattice
c
*/
          for(i=1;i<=p;i++) {

            delta[i] = (1.0-alpha)*delta[i]
                                    - ( e * d->r[i-1][place] ) /
                                      (   1.0-d->gamma[I-2][place]   ) ;


/*
  Is this set of reflection coefficients to be constrained to 0?
*/
            if(d->tapper != 0  &&  (((i % d->tapper) > d->var) &&
                                    ((i % d->tapper) <  (d->tapper - d->var)) ) )
            {
              ksupe = 0;
              ksupr = 0;

            }
            else
            {
              if(esupe == 0) {
                ksupe = 0.0;
              }
              else {
                ksupe = delta[i] / esupe;
              }
            if(d->esupr[i-1][place] == 0.0) {
              ksupr = 0.0;
            }
            else {
              ksupr = delta[i] / d->esupr[i-1][place];
            }
          }   /* end of "else if not constrained..." */


          d->r[i][n % 2] = d->r[i-1][place] + ksupe * e;
          e = e + ksupr * d->r[i-1][place];

          if(esupe == 0.0) {
            d->esupr[i][n % 2] = 0.0;
          }
          else {
            d->esupr[i][n % 2] = d->esupr[i-1][place] -
                                 delta[i]*delta[i] /
                                             esupe;
          }
          if(d->esupr[i-1][place] == 0.0) {
            esupe = 0.0;
          }
          else {
            esupe = esupe -((delta[i]*delta[i])/
                      d->esupr[i-1][place] );
          }

          if(d->esupr[i-1][place] == 0.0) {
            d->gamma[I-1][place] = 0.0;
          }
          else {
            d->gamma[I-1][place] = d->gamma[I-2][place] +
                                   d->r[i-1][place]*
                                   d->r[i-1][place] /
                                   d->esupr[i-1][place];
          }


/*
  Save ksupe and ksupr
*/
        ksupe_sav[i] = ke[i-1] = ksupe;
        ksupr_sav[i] = kr[i-1] = ksupr;

        }       /* End of for i=1 to p  */


        *output = e;            /*  Return prediction error  */
        if(freeze == 1)  {

/*
c
c  return the final filter coefficients
c
*/
          r[0] = 1;
          for(n=1;n<=p;n++)  {
            rn[n] = 1;
            for(k=n-1;k>0;k--)  rn[k] = r[k-1] + ksupe_sav[n]*forw[k];
            rn[0] = ksupe_sav[n];

            en[0] = 1;
            for(k=1;k<n;k++)  en[k] = forw[k] + ksupr_sav[n]*r[k-1];
            en[n] = ksupr_sav[n];

            for(j=0;j<=p;j++)  {
              r[j] = rn[j];
              forw[j] = en[j];
            }
          }

          for(k=1;k<=p;k++)  wt[k-1] = forw[k];

        }  /*  end of if freeze...  */


        for(j=1;j<=p;j++) d->delta[j] = delta[j];

} /*  End of LSL algorithm */
```

Fast Kalman Routine

```
/*
        Routines to implement Fast Kalman filtering.

        Calling sequence is:

                Call kalman_init once to initialize the data structures,
                then call kalman_pt once for each data point.

        kalman_init(ptr,N,delta,lambda,delay)

                ptr             pointer to kalman_data structure defined in
                                kalman.h

                N               Number of taps in filter

                delta           Small real number (>0) to be used to make
                                sure the arrays are non-singular

                lambda          Fade factor.  Should be < 1.0 but > 0.0

                delay           NOT USED.


        kalman_pt(ptr,y,e,wt)

                ptr             Pointer to kalman_data structure defined in
                                kalman.h

                y               Current data point.  FLOAT

                e               Current error, returned by the routine. FLOAT

                wt              Array of filter weights returned by the
                                routine for this point.  FLOAT

                dk              Value to use as the "correct" value for this
                                point.  FLOAT


        NOTE:  All notation is taken from the paper by Falconer and Ljung,
                IEEE Transactions on Communications, Vol. COM-26, Oct. 1978.
```

```
*/



#include        </ncs/jhw/lsl/1pt/kalman.h>
#include        <stdio.h>


kalman_init(ptr,N,delta,lambda,delay)

        struct  kalman_data     *ptr;
        int     N,delay;
        float   delta,lambda;
{

        int     j;



        for(j=0;j<N;j++) ptr->AN[j] = ptr->DN[j] = 0.0;
        for(j=0;j<=N;j++) ptr->kn[j] = ptr->xn[j] = ptr->CN[j] = 0.0;

        ptr->Epp = delta;
        ptr->taps = N;
        ptr->n = -1;
        ptr->lambda = lambda;

        return;

}



#define         tap_len         j=0;j<ptr->taps;j++
#define         mn              kn


kalman_pt(ptr,y,e,wt,dk)

        struct  kalman_data     *ptr;
        float   y,*e,wt[],dk;

{

        float   temp,epsilon,epsilon_prime,rho,mu,eta;
        int     j,taps;
        taps = ptr->taps;

/*
  Return newest error and filter weights
*/

        temp = 0.0;
        for(tap_len) temp = temp + ptr->CN[j] * ptr->xn[j];
        *e = dk - temp; /*  error at pt. n */
/*
  Calculate new weights
*/
        for(tap_len) ptr->CN[j] = wt[j] = ptr->CN[j] + ptr->kn[j] * *e;

/*
  Start update of kalman gain vector kn
*/

/*  epsilon  */
        temp = 0.0;
        for(tap_len) temp = temp + ptr->AN[j] * ptr->xn[j];
        epsilon = y + temp;

/*  AN array  */
        for(tap_len) ptr->AN[j] = ptr->AN[j] - ptr->kn[j] * epsilon;

/*  epsilon prime  */
        temp = 0.0;
        for(tap_len) temp = temp + ptr->AN[j] * ptr->xn[j];
        epsilon_prime = y + temp;

/*  Epp array  */
        ptr->Epp = ptr->lambda * ptr->Epp + epsilon * epsilon_prime;


/*
  calculate km vector that will be partioned into mn and mu
*/
        if(ptr->Epp != 0)
        {
          for(j=taps;j>0;j--) ptr->kn[j] = ptr->kn[j-1] +
                                   ptr->AN[j-1] * epsilon_prime / ptr->Epp;
            ptr->kn[0] = epsilon_prime / ptr->Epp;
        }
        else
        {
          for(j=taps;j>0;j--) ptr->kn[j] = ptr->kn[j-1];
          ptr->kn[0] = 0.0;
        }
/*
  Partition km (kn) into mu and mn.  Pull out mu, leaving kn[0...] as mn
*/
        mu = ptr->kn[taps];

/*  Shift xn, adding new point as xn[0]  */
        rho = ptr->xn[taps-1];
        for(j=taps-1;j>0;j--) ptr->xn[j] = ptr->xn[j-1];
        ptr->xn[0] = y;

/*  eta  */
        temp = 0.0;
        for(tap_len) temp = temp + ptr->DN[j] * ptr->xn[j];
        eta = rho + temp;

/*  Update DN array  */
        for(tap_len) ptr->DN[j] = (ptr->DN[j] - ptr->mn[j] * eta) /
                                                      (1.0 - mu * eta);

/*  Update kalman gain vector for next point  */
        for(tap_len) ptr->kn[j] = ptr->mn[j] - ptr->DN[j] * mu;

        ptr->n++;


        return;

}
```

Kalman Filter Common File

```
#define MAX_KALMAN 1024

struct  kalman_data {

        float   AN[MAX_KALMAN],         /* temporary recursion array */
                DN[MAX_KALMAN],         /* temporary recursion array */
                CN[MAX_KALMAN],         /* tap weight array          */
                kn[MAX_KALMAN],         /* Kalman gain array         */
                xn[MAX_KALMAN],         /* the last N inputs         */
                lambda,                 /* fade factor               */
                Epp;                    /* temporary recursive error */

        int     n,                      /* number of points          */
                taps,                   /* number of taps in filter  */
                delay,
                tapper,                 /* period of taps not to be
                                           constrained               */
                var;                    /* number of taps on each
                                           side of tapper not to be
                                           constrained               */
};
```

LSL Filter Common File

```
#define MAX_ORDER 256

struct lsl_data  {
        float   r[MAX_ORDER+1][2],      /* reverse linear predictor value */
                e,                      /* forward linear predictor value */
                esupr[MAX_ORDER+1][2],  /* (E super r) */
                esupe,                  /* (E super e) */
                delta[MAX_ORDER+1],             /* intermeditate values */
                gamma[MAX_ORDER+1][2],
                b[MAX_ORDER+1][MAX_ORDER+1][2], /*backward filter coefficients*/
                a[MAX_ORDER+1][MAX_ORDER+1],    /*forward filter coefficients */
                ksupe,
                ksupr ;

        float   input[MAX_ORDER],       /* last p inputs */
                alpha;                  /* fade factor */
        int     n,       /* current data point (mod p) */
                p,  /* filter length (number of weights) */
                tapper,                 /*  Tap period that is not to be
                                            constrained  */
                var;                    /*  Number of taps on each side of
                                            tapper that are not to be
                                            constrained  */
} ;  /* end of lsl data structure */
```