PARAMETRIC DESIGN OF AN ADAPTIVE LINE ENHANCER FOR MULTIPLE SWITCHING TONES IN A CORRELATED NOISE ENVIRONMENT

by

Robert D. Ritter

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

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Dr. F. Ricci, Chairman

Dr. J. Knight

Dr. P. Moser

May, 1988

Blacksburg, Virginia
PARAMETRIC DESIGN OF AN ADAPTIVE LINE ENHANCER FOR MULTIPLE SWITCHING TONES IN A CORRELATED NOISE ENVIRONMENT

by

Robert D. Ritter

Committee Chairman: Dr. Fred Ricci

Electrical Engineering

(ABSTRACT)

This thesis demonstrates how a Fast Gradient approximation to a Lattice Filter can be used as an Adaptive Line Enhancer for sampled data consisting of multiple switching tones in correlated noise. A tradeoff analysis is performed with four methods of digital filtering including a conventional Digital Fourier Transform (DFT) algorithm, a Least Mean Squares (LMS) adaptive algorithm, a Fast Recursive Least Squares (Fast RLS) adaptive algorithm, and the Fast Gradient adaptive algorithm. The DFT algorithm is incapable of removing correlations from the incoming noise, and the LMS and Fast RLS algorithms become unstable when a dynamic switching environment is being filtered. The Fast Gradient adaptive algorithm simulated on a computer is robust and capable of converging to an optimal set of FIR filter weights with minimum Mean Squared Error. Parameters for the Fast Gradient algorithm are optimized to provide good filter performance with a minimum number of computations.
Acknowledgements

I would like to thank the members of my advisory committee, Dr. Fred Ricci, Dr. Joe Knight and Dr. Phil Moser for their guidance and suggestions throughout the development of my thesis. I would also like to thank Dr. Gary Jacyna and Dr. Moser for their valuable research on adaptive processes, and AVTEC Systems Inc. for sponsorship of my research. Special thanks goes to Donna for all those countless hours of assistance.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Objective</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Scope</td>
<td>2</td>
</tr>
<tr>
<td>2.0 Problem Statement and Approach</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Generate Incoming Signal</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Process Data</td>
<td>7</td>
</tr>
<tr>
<td>3.0 Critical Literature Survey</td>
<td>9</td>
</tr>
<tr>
<td>3.1 The Simulated Algorithms</td>
<td>10</td>
</tr>
<tr>
<td>3.2 New Material</td>
<td>11</td>
</tr>
<tr>
<td>4.0 Algorithm Theory</td>
<td>12</td>
</tr>
<tr>
<td>4.1 Signal Properties</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Noise</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Digital Fourier Transform Filter</td>
<td>19</td>
</tr>
<tr>
<td>4.4 Adaptive Filter Methods</td>
<td>20</td>
</tr>
<tr>
<td>5.0 Experimental Procedure</td>
<td>37</td>
</tr>
<tr>
<td>5.1 Hardware Requirements</td>
<td>37</td>
</tr>
<tr>
<td>5.2 Software Requirements</td>
<td>38</td>
</tr>
<tr>
<td>5.3 Parameter Specifications</td>
<td>39</td>
</tr>
<tr>
<td>5.4 Digital Signal Processor Simulation</td>
<td>41</td>
</tr>
<tr>
<td>6.0 Results</td>
<td>46</td>
</tr>
<tr>
<td>6.1 The Digital Fourier Transform Filter</td>
<td>46</td>
</tr>
<tr>
<td>6.2 Least Mean Squares Filter</td>
<td>48</td>
</tr>
<tr>
<td>6.3 Fast Recursive Least Squares</td>
<td>52</td>
</tr>
<tr>
<td>6.4 Fast Gradient Filter</td>
<td>55</td>
</tr>
<tr>
<td>7.0 Conclusions</td>
<td>64</td>
</tr>
<tr>
<td>7.1 The Fast Gradient Adaptive Line Enhancer</td>
<td>67</td>
</tr>
<tr>
<td>7.2 Future Study</td>
<td>67</td>
</tr>
<tr>
<td>8.0 References</td>
<td>69</td>
</tr>
<tr>
<td>Appendix A Computer Listing of ALE Simulation</td>
<td>73</td>
</tr>
<tr>
<td>Appendix B ALE Simulation Output Data</td>
<td>90</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

1.0 Introduction

In a communications Digital Signal Processing environment, a block of data is received, and decoded for information content. If the signal is corrupted by noise, then it may be incorrectly interpreted, or not even detectable by the system. For general applications, a bandpass filter is placed at the frequency of the received data to increase the Signal to Noise Ratio (SNR). If the frequency of the signal is not known, then a conventional fixed frequency filter is of no use. A filter whose parameters are dynamic is implemented to automatically construct bandpass filter segments around the frequencies of received information. Such a filter is said to adapt its parameters to the received environment.

For a received digital signal consisting of multiple frequency varying tones in correlated noise, a filter algorithm must be chosen that is capable of increasing the overall SNR in an efficient manner. The power spectral density of correlated noise is more heavily weighted in certain regions of the spectrum, and the filter implemented must be capable of attenuating the high level portions of the noise more than the low level noise regions. In a signal environment consisting of variable frequency tones switching on and off, the filter algorithm must be exposed to a large number of samples to ensure that all possible tone frequencies have been seen. If a tone does not appear at one of the frequencies while the filter algorithm is observing the data, then the filter will attenuate that band and information decoding may not be possible.
1.1 Objective

The objective of this thesis is to determine the parameters necessary for development of a digital filter capable of attenuating correlated noise from switching variable frequency tones. Response data is plotted in the frequency domain and compared to the input signal data. Filter quality is based on numerical processing time and acquired SNR improvement. There has been a great deal of research based on fast converging adaptive algorithms for stable signal environments; however, not much work has been conducted on non-stationary switching environments. This report provides application data and examples for a robust adaptive algorithm capable of long term convergence on a random switching signal in correlated noise.

1.2 Scope

The scope of this work is a comparison of capabilities between four digital filtering algorithms. The algorithms are chosen based on a survey of existing research on adaptive signal processing. A trade off is conducted, and filter parameters optimized to provide an efficient Signal to Noise Ratio improvement for the signal processing system (Figure 1.2-1). This system is simulated on a micro-computer workstation that is part of a Local Area Network.
Figure 1.2-1: System Configuration
1.2.1 Environment Simulation

A software system is designed to create the experimental signal environment and to process the signal with the algorithms under test. The signal is generated, filtered, and the input and output data streams transformed into the frequency domain for spectral comparison. Resultant data is graphed and tabulated for observation of SNR improvement and computational complexity is measured in processing time.

1.2.2 Algorithm Limitations

Along with testing the performance of each algorithm, observed limitations of capabilities are examined and documented. Situations sometimes occur in which algorithm response is unsatisfactory or even unstable. Poor results are achieved when an algorithm is not capable of an applied task, or when an algorithm numerical accuracy is bounded and additional computations are forced through.
Chapter 2: Problem Statement and Approach

2.0 Problem Statement and Approach

This work addresses a unique digital signal processing task for improvement in the quality of a received buffer of switching tones in correlated noise. The received data buffer contains 32768 (32k) digitized samples. For the purpose of generating filter weights, a portion of the data called an analysis frame is observed and a best fit to the existing signal properties is computed. The optimal estimate produces the lowest mean squared error between the estimate and the original signal. This estimation is based on statistically regressing the filter weights based on correlations in the received signal [1].

A software simulation of the signal processing system is designed to emulate the digital signal processing functions including environment generation and filtering. The incoming signal is generated to include the selected noise source and incoming signal properties. One of the filter algorithms is chosen and weights are generated to process the received data. The computation times for generation of filter weights is recorded, and the input and output frequency spectrums are plotted and compared. The signal processing system functional flow is pictured in Figure 2.0-1.
FIGURE 2.0-1: Operational Flow Diagram
2.1 Generate Incoming Signal

To test the filter designs, signal environments are developed by software simulation. A signal of interest is generated to simulate the signal that the DSP system is meant to decode. In this report, the signal is characterized by a digital Facsimile (FAX) signal consisting of multiple varying frequency tones [2]. Test conditions are applied to each filter type, and the filtered results compared. The three signal environments that can be generated by the software are the signal of interest (FAX tones) plus:

1. No noise
2. Gaussian noise
3. Correlated noise

The FAX tones are generated by selecting one of the three frequencies that is randomly chosen before each bit transmission. A tone of the chosen frequency is generated for fifty samples (1 bit), and a new frequency is randomly chosen for the next bit. The Gaussian noise is a normalized distribution of uniformly distributed random variables. When used, the Gaussian noise is simply added to the FAX signal. The correlated noise is simulated as an Amplitude Modulated (AM) tone with a Gaussian random variable determining amplitude. The noise is passed through a simple single pole low pass filter which adds correlation to the AM tone (see section 4.2).

2.2 Process Data

The data for each of the three test cases is generated and stored in data files. The stored data is read in by each of the four algorithms being studied and is filtered several
times using altered filter parameters. The parameters for each of the four filters are changed for the purpose of optimizing the output response. The best filter is judged by comparing the obtained output Signal to Noise Ratio, the convergence rate, and the cost in computer CPU time for each ALE algorithm.

The input, output, and filter tap value data sets are all plotted in the frequency domain using two plot routines in the filter programs. The input and output data are transferred to a spectrum plotting subroutine where the time domain data are applied to a Hanning window for reduction of spectral smearing [3]. 50% zero padding is added to reduce circular convolution effects [4], and 50% overlapping between consecutive FFT calculations is included to minimize the effects of data loss due to windowing and to improve resulting accuracy of spectral density averages [4]. The filter tap values are plotted using a Fourier Transform of the tap values.
3.0 Critical Literature Survey

In order to solve the problem of adaptive filtering for a multiple switching tone signal in correlated noise, research must begin with a survey of existing techniques so the optimal filtering method may be utilized. Development of adaptive algorithms started with statistical regression formulae for estimation of data. A classical study was performed in 1927 where correlations in sunspot activity were investigated [5]. In the advent of digital communication technology, applications to the numerical data processing became an important topic. Pioneering work by Wiener [6], Levinson [7], Bode and Shannon [8] began about forty years ago covering the topics of digital data prediction and estimation. Specific theory on adaptive algorithms was developed in the early 1960's, and works by R. Kalman [9] [10] and Bernard Widrow [11] [12] [13] provided usable and efficient means for data adaptation. In the following years and through the present, a great deal of study has been devoted to development of faster adaptive algorithms capable of filtering a wide variety of signals. Tutorials by B. Widrow, et al. [14], and B. Friedlander [15] provide a good background in adaptive processing, and recent books by Haykin [16], Widrow [17], Orfanidis [1], and Treichler [18] give in depth study to the development of these algorithms. Development of Fast adaptive algorithms and Lattice filter methods can be found in papers by Kalman [19], Morf [20], Ljung [21], Falconer [22], Cioffi [23], Marple [24] and several others. Stanford University, CA, is the source of a great deal of information on adaptive filtering research.
3.1 The Simulated Algorithms

The algorithms chosen for study in this report represent a great deal of the presently available means for digital adaptive filtering. The first method of filtering (the DFT approach) is a classical digital filter that has proven capabilities of efficient filtering in wideband Gaussian noise environments [25] [26]. The Least Mean Squares (LMS) adaptive filter, based on the Widrow-Hoff formulation and written for computer by S. L. Marple, Jr. [27], is the first form of adaptive filter algorithms implemented in this thesis [11]. It is a stable formula that is computationally efficient for use on a computer; many LMS iterations can be run with small expense in CPU time.

The fast Recursive Least Squares adaptive filter tested in this report is based on a study by Cioffi and Kailath [28] in 1984. The algorithm was written for use on a computer by S. L. Marple, Jr. in a book containing many useful DSP applications [27]. For the simulation in this report, the incoming data is not complex and the subroutine RLS has been modified to accept real data.

The fast Gradient filter is based on research by Dr. Gary Jacyna of Unisys Corporation, Reston, VA [29], and was developed into a computer simulation by Dr. P. Moser, AVTEC Systems Inc., Fairfax, Virginia and is described in greater detail in section 4.4.3. The fast Gradient filter, based on theory of B. Friedlander [15], is a Gradient lattice approximation designed for use on a computer. It is capable of convergence and is stable in real-time processing environments with limits on numerical precision and hardware constraints.
3.2 New Material

To this date, research has focused primarily on adaptive filters in stable signal environments. Studies have included many applications of filter algorithms on single and multiple tones added to various noise sources. In this report, an unstable environment is created with tones that randomly switch on and off in multiple frequency intervals. In selecting a filter for this application, the four algorithms described are tested under equal conditions and judged on total performance. Judgement is based on a filter's ability to make an unbiased estimate of the signal environment and its ability to remove the unwanted correlated noise. Convergence time to good filter weights is also considered when determining the operational parameters for a filter.
4.0 Algorithm Theory

The purpose of pre-filtering the incoming digitized data is to increase the overall Signal to Noise Ratio (SNR). This increase not only makes the signal more detectable, but also reduces the probability of bit errors in the decoding process. The noisy signal environment generated for this DSP simulator consists of a 32768 (32k) sample block of digitized tones in Gaussian and correlated noise.

The multiple tone varying frequency pulses are generated by the simulation software and stored in a data array for further processing. The menu of choices in the simulation software prompts the user for number of signal tones, beginning frequency, and shift frequency of the information bits. A uniformly distributed random variable determines the frequency of each bit. After 50 samples are generated, a new random sample is taken to produce the tone for the next bit. The process generates a block of 32k data points containing the signal information.

Following generation of the pure signal, user supplied information from the software menus causes the program to generate the appropriate noise source which is added to the signal and stored in the array. If no noise is chosen, the array is left unchanged. When the Gaussian noise is selected, 32k Gaussian distributed random variables are generated and added to the existing signal. Upon the selection of correlated noise, 32k samples are generated and added to the signal.
4.1 Signal Properties

In a Pulse Code Modulation system, the instantaneous transmitted frequency is switched between two or more values. For the experiment in this report, a trinary system is implemented consisting of three possible frequencies for each bit (50 sample points)

\[ \varnothing_0(t) = A \sin(\omega t) \quad (4.1-1) \]

\[ \varnothing_1(t) = A \sin((\omega + \Delta \omega)t) \quad (4.1-2) \]

\[ \varnothing_2(t) = A \sin((\omega + 2\Delta \omega)t) \quad (4.1-3) \]

where \( \omega \) is the user selected first tone frequency, and \( \Delta \omega \) is the frequency difference between tones. The probability of error \( P_e \) for receiving an incorrect bit is based on a Probability Density Function (PDF) for the waveforms (see Figure 4.1-1) [2]. For Gaussian noise and equiprobable zeros ones and twos, the \( P_e \) corresponds to overlap in the PDFs which is a measure of the envelope variance,

\[ P_e = \frac{2(n-1)}{n} \text{Erfc} \left[ \frac{a}{2(\sigma)} \right] \quad (4.1-4) \]

and in terms of the received SNR

\[ P_e = 2 \left( 1 - \frac{1}{n} \right) \text{Erfc} \left( \sqrt{\frac{3}{n^2 - 1}} \frac{S}{N} \right) \quad [2] \quad (4.1-5) \]

As the SNR increases, the probability of error decreases exponentially (Figure 4.1-2)[30].
Figure 4.1-1: Error Probability Density Function for Trinary Switching Tones
Figure 4.1-2: Probability of Error for 2 and 3 Switching Tones versus SNR
4.2 Noise

To simulate the noise received in frequency band of switching sinusoidal tones, three options are easily implemented. The first option is to add no noise to the signal which is similar to having a pure white noise base in the system. This situation is not likely to be seen by an actual receiver and will not be applied to the filter algorithms in this thesis. The second noise option is Gaussian noise. This noise is induced by generating Gaussian distributed random numbers and adding them to the generated signal. The Gaussian distributed random variables are generated using a Box-Jenkins approach [26] that requires only two uniformly distributed random variables

\[ \text{Gauss} = \sqrt{-2 \times \text{ALOG}(\text{RND1})} \times \cos(2\pi \times \text{RND2}), \] (4.2-1)

The third option is correlated noise which is the most difficult to remove from a signal and is the primary focus of this research. Noise characterized by a bandlimited or nonuniform power spectral density is colored noise and consists of wideband correlations [31]. To induce correlation, a noise source can be passed through a network with a bandlimited transfer function. For this report, a Cyclo-Stationary semi-correlated noise source is generated by amplitude modulating a sinewave with a Gaussian distributed random variable [4] (see time response in Figure 4.2-1). The spectral representation is an impulse centered at the sinewave frequency plus and minus random impulses at all of the frequency components of the Gaussian numerical stream. The non-stationary noise is filtered by a single pole lowpass filter with response

\[ H(Z) = \frac{1}{1 + 0.9Z^{-1}}, \] (4.2-2)

which results in an additive correlated noise source (see Figure 4.2-2).
Figure 4.2-1: Unfiltered Cyclo-Stationary Correlated Noise Source
Figure 4.2-2: Cyclo-Stationary Correlated Noise Spectrum
4.3 Digital Fourier Transform Filter

Properties of the Digital Fourier Transform (DFT) provide the means by which the spectral content of a series of digitized data may be represented. If the inverse DFT is applied to the average spectral response of the data, a set of time data is created that contains the average time response of the data block. For this case, the block of 32 k data points is divided into M subsets of data with length \( N \) samples. Taking the inverse DFT of the average of \( M \) DFTs on the signal data results in \( N \) samples that represent a time average of the data block. Since the DFT is a linear transformation \([2]\), the average of \( M \) sections of time data length \( N \) give the same results as by taking the DFTs and inverse DFT.

\[
F^{-1} \left( \text{Avg} \left( F(B_1) + F(B_2) + \ldots + F(B_M) \right) \right) \]

\[
= \text{Avg} \left( f(B_1) + f(B_2) + \ldots + f(B_M) \right) \tag{4.3-3}
\]

where

\[
\begin{array}{cccccc}
B(1) & B(2) & B(3) & \ldots & B(M-1) & B(M) \\
\end{array}
\]

The spectrum of the data averages increases in magnitude where correlations occur between frames. If the time response of this data is used as \( N \) filter weights then the response will be band pass filters with passbands in the regions of correlation. In the DFT filter routine, the entire block of data is divided into \( M \) segments of \( N \) data samples. The \( N \) samples of each segment are averaged in order where the \( N \) filter weights become:
\[ \text{Weight}(1) = \left[ N(1,1) + N(2,1) + \ldots + N(M,1) \right] / M \]
\[ \text{Weight}(2) = \left[ N(1,2) + N(2,2) + \ldots + N(M,2) \right] / M \]
\[ \vdots \]
\[ \text{Weight}(N) = \left[ N(1,N) + N(2,N) + \ldots + N(M,N) \right] / M \] (4.3-4)

and the data is filtered using these calculated weights in an FIR filter.

4.4 Adaptive Filter Methods

The Adaptive filter chosen for use on the receiver simulator is an Adaptive Line Enhancer (ALE) seen in Figure 4.4-1 [14]. There is one source of channel information containing both signal of interest \( S(t) \) plus noise \( N(t) \) that is fed into the filter. The reference signal for the Noise canceler is a delayed version of the primary input. The delayed input is filtered by the present weights, compared to the primary input, and the error is used to update the weights. This process is depicted in Figure 4.4-2 [14] showing to Adaptive linear combiner. The filter weights generally converge to steady-state values after a certain number of sample observations depending on the convergence algorithm selected. In this experiment, a portion of the 32k block of samples (size M) is used for N weight calculations and then the entire 32k block is filtered by the computed set of filter weights (see Figure 4.4-3).
Figure 4.4-1: Adaptive Line Enhancer
FIGURE 4.4-2: Adaptive Linear Combiner
Figure 4.4-3: Adaptive Filter Process
Determining filter weights requires the choice of the best algorithm for the expected data environment. At this time, there are several different algorithms capable of stochastically converging to adaptive filter weights. The different algorithms have varying methods of adapting that affect convergence rates and numerical sensitivity and stability. The three algorithms tested in this report are a good representation of the work conducted on numerical data estimation for signal processing. The Least Mean Square algorithm estimates the parameters of a statistical data model by adjustments based on the Mean Square Error (MSE) between the actual data and the present weights. The Fast Recursive Least Squares algorithm is a recursion based structure for quickly converging to weights by simultaneous calculation of error in forward and backwards time increments. The Fast Gradient algorithm uses a lattice structure and converges based on a numerical estimate of the MSE. The following sections provide more detailed descriptions of the adaptive algorithms.

4.4.1 The Least Mean Square Adaptive Algorithm

The method of Least Squares data adaption is based on statistical regression, where an estimate of a data set minimizes the Sum of Squares for Error (SSE) between the real data and the selected weight values. [32] To minimize the error, the SSE function is differentiated and set equal to zero. Equations for the Least Mean Square (LMS) algorithm are discussed using matrix notation [14] (matrix/vectors represented in bold type) from the adaptive linear combiner in Figure 4.4-2; the output shall be written

\[ Y = X^T W = W^T X \]  

(4.4.1-1)
where $X^T$ is the transpose of the $X$ matrix. The error is difference between the output and the reference input (desired response $d$)

$$\epsilon = d - Y = d - X^T W.$$  \hfill (4.4.1-2)

If the error is left in this form, it can range from large negative to large positive values and would be difficult to minimize. By squaring the error and taking its expected value,


a quadratic function of the weights results that represents a bowl shaped surface with a positive minimum value. To minimize this equation, the derivative with respect to $W$ is determined and set equal to zero.

$$\nabla \frac{d E[\epsilon^2]}{d W} = -2 E[dX] + 2 E[XX^T] W$$  \hfill (4.4.1-4)

For $E[dX]$ defined as the cross correlation matrix $P$, and $E[XX^T]$ defined as the input correlation matrix $R$, the gradient

$$\nabla = 2R W - 2P$$  \hfill (4.4.1-5)

The Weiner-Hopf equation represents the optimal weight vector where the gradient of the error is set to zero. \hfill [6]

$$W^* = R^{-1} P$$  \hfill (4.4.1-6)
The Widrow-Hoff LMS algorithm implements an estimate to the gradient and updates the weight vector with a fraction ($\mu$) of the gradient estimate [11]:

$$W_{I+1} = W_I + 2 \mu \varepsilon_I X_I$$ \hspace{1cm} (4.4.1-7)

where the gradient estimate

$$\text{est grad} = -2 \varepsilon_I X_I.$$ \hspace{1cm} (4.4.1-8)

The value chosen for $\mu$ controls the LMS function stability and convergence rate. If $\mu$ is too small the algorithm will converge too slowly, and if $\mu$ is too large the algorithm will become unstable and $W$ will diverge. The value for $\mu$, to guarantee converge, must be in the range

$$0 < \mu < \frac{1}{R}$$ \hspace{1cm} (4.4.1-9)

where $R$ is the magnitude of the input variance [1]

$$R = \mathbb{E}[X_I^2].$$ \hspace{1cm} (4.4.1-10)
4.4.2 Recursive Least Squares Adaption

Following tests of the LMS filter, an adaptive algorithm with a fast convergence rate is implemented and compared. The Recursive Least Squares (RLS) algorithms represent an exact minimization of least square errors for stochastic linear prediction and FIR Wiener filtering [1]. In this report, a Fast RLS algorithm is simulated on the computer which is more efficient than the classical RLS and still provides an exact solution to the data estimate. The fast RLS algorithm updates its parameters using vector operations instead of full matrices which significantly reduces computational complexity. Backward linear prediction coefficients are determined to minimize the backwards SSE and parameter initialization is automated to avoid biasing the adaption procedure.

The error minimization from the LMS algorithm (E [e²]) is represented by the RLS as a least squares time average for all N (number of weights) error segments

\[
\varepsilon(n) = \sum_{k=0}^{n} \varepsilon_k^2 = \min . \tag{4.4.2-1}
\]

For emphasis on more recent data, an exponential weighting or "forgetting" factor is included to make the error

\[
\varepsilon(n) = \sum_{k=0}^{n} \lambda^{n-k} \varepsilon_k^2 \tag{4.4.2-2}
\]
where \(0 < \lambda < 1\). The individual error calculation is determined by a present data value \((x_k)\) minus an estimate of \((x'_k)\) based on past data multiplied by present filter weight values or

\[
e_k = x_k - x'_k \tag{4.4.2-3}
\]

where

\[
x'_k = \sum_{m=0}^{N} W_m x_{k-m} \tag{4.4.2-4}
\]

Setting the derivative of the error equal to zero minimizes the error as in the LMS routine

\[
\frac{d \epsilon(n)}{d W} = -2 \sum_{k=0}^{n} \lambda^{n-k} \epsilon_k X(k) = 0 ,
\]

which may be normally represented as

\[
\left[ \sum_{k=0}^{n} \lambda^{n-k} X(k) X^T(k) \right] W = \sum_{k=0}^{n} \lambda^{n-k} x_k X(k) . \tag{4.4.2-5}
\]

The covariance matrix \(R = E [X X^T]\), and the cross-correlation vector \(P = E [x X]\) make the Weiner solution

\[
W^* = R^{-1} P . \tag{4.4.2-6}
\]
The recursive equations become

\[ R(n) = \lambda R(n-1) + X X^T \]  \hspace{1cm} (4.4.2-7)

\[ P(n) = \lambda P(n-1) + X \]  \hspace{1cm} (4.4.2-8)

which lead to

\[ W(n) = W(n-1) + R^{-1}(n) X e_{n/n-1} \]  \hspace{1cm} (4.4.2-9)

where

\[ e_{n/n-1} = x - W^T(n-1) X . \]  \hspace{1cm} (4.4.2-10)

In the fast RLS algorithm, matrix operations are replaced with vectors which reduce the number of adaptive filter computations from \( N^2 \) to \( 5N \) where \( N \) equals number of filter weights. The fast RLS routine initializes its prediction parameters with a least squares solution from the first \( L + 1 \) observations which eliminates classical RLS initialization bias. To increase computational efficiency, the fast RLS algorithm calculates backward linear prediction coefficients as well as the forward calculations by the LMS and classical RLS methods. Having both directions of error prediction allows the determination of error over a wider sample space which decreases the total number of observations needed for convergence. The backwards error prediction parameters are calculated as
where
\[ e_{p,N}^b [N] = x[n-p] + \sum_{k=1}^{P} a_{p,N}^b[k] x[n-p+k]. \] (4.4.2-12)

4.4.3 Fast Gradient Adaptive Filter

The fast Gradient adaptive filter algorithm is a gradient approximation to a least squares lattice filter. In the lattice filter structure, the prediction parameters are derived in the forward and backward cases (see Figure 4.4-4) [29]. To compute filter coefficients, a Levinson algorithm is implemented [15] where the forward prediction vector (A) and reverse prediction vector (B) are determined. From the correlation sequence \( R_{i-j} = E[Y Y^T] \), the mean square error is written as

\[
R_p^c = R_0 + \sum_{i=1}^{p} A_{p,i} R_{-i} \tag{4.4.3-1}
\]

where \( R_{-i} = R_i \) in the basic Yule-Walker equation:

\[
\begin{bmatrix}
R_0 & R_1 & \cdots & R_p \\
R_1 & R_0 & R_1 & \cdots & R_{p-1} \\
\vdots & \vdots & \ddots & \vdots & \ddots \\
R_p & R_{p-1} & \cdots & R_0
\end{bmatrix}
\begin{bmatrix}
I \\
A_{p,1,1} \\
A_{p,1,2} \\
\vdots \\
A_{p,1,p}
\end{bmatrix}
= \begin{bmatrix}
R_p^c \\
0 \\
0 \\
\vdots \\
0
\end{bmatrix}. \tag{4.4.3-2}
\]
Figure 4.4-4: Basic Lattice Structure
In the tutorial on adaptive lattice filters by Benjamin Friedlander [15] the parameters $k_i$ and $\Delta_{p+1}$ are defined as the reflection (partial correlation) coefficients and cross correlation of the bi-directional prediction errors respectively. These two parameters lead to determination of updates on the prediction errors

$$R_{p+1}^f = R_p^f - K_{p+1}^f \Delta_p^f$$

and

$$R_{p+1}^b = R_p^b - K_{p+1}^b \Delta_p^b$$  \hspace{1cm} (4.4.3-3)

from the stationary case where

$$R_{p+1}^s = R_p^s - K_{p+1}^s \Delta_p^s$$  \hspace{1cm} (4.4.3-4)

The update to the prediction coefficients is performed as follows from the recursion relationships:

$$A_{p+1,i} = A_{p,i} - K_{p+1}^r B_{p,p+1-i}$$  \hspace{1cm} (4.4.3-5)

$$B_{p+1,p+1-i} = B_{p,p+1-i} - K_{p+1}^e A_{p,i}$$  \hspace{1cm} (4.4.3-6)

In the scalar case, the backward predictor is a reverse of the forward, and using the Anderson recursion relation,

$$A_{p+1,i} = A_{p,i} - K_{p+1} A_{p,p+1;i} \quad ; \quad i=1,2,...,p$$

$$A_{p+1,p+1} = -K_{p+1}$$  \hspace{1cm} (4.4.3-7)
The next step is to estimate the reflection coefficients of the lattice $K_{p+1}$. Using a Yule-Walker approach, a direct realization is

$$
K_{p+1} = \frac{R_{p+1} + \sum_{i=1}^{p} A_{p,i} R_{p+1-i}}{R_{i} + \sum_{i=1}^{p} A_{p,i} R_{i}}
$$

(4.4.3-8)

for \textit{a priori} knowledge of $R_i$. In an indirect form, making use of a recursion on the errors, the reflection coefficient estimate

$$
K_{p+1,k} = \frac{2 \sum_{i=p+1}^{k} \varepsilon_{p,i} r_{p,i-1}}{\sum_{i=p+1}^{k} (\varepsilon_{p,i}^2 + r_{p,i-1}^2)}
$$

(4.4.3-9)

is the Burg forward-backward technique which is stable since $|K_{p+1}| \leq 1$. By writing the correlation and cross correlation sequences as follows:

$$
R_{p,k}^2 = \sum_{i=p+1}^{k} (\varepsilon_{p,i}^2 + r_{p,i-1}^2)
$$

(4.4.3-10)

$$
d_{p+1,k} = \sum_{i=p+1}^{k} \varepsilon_{p,i} r_{p,k-1}
$$

(4.4.3-11)
the reflection (PARtial CORrelation or PARCOR) coefficients become

\[ K_{p+1,k} = \frac{d_{p+1,k}}{R_{p,k}} \]  

(4.4.3-12)

In the gradient update equations, as described in section 4.4-1, the weight update parameter is written as

\[ A_t(k) = A_t(k-1) - 2\mu E_k Y_{k-i} \]  

(4.4.3-13)

where \( A \) is the weight vector and \( \mu \) is the convergence rate. For the LMS algorithm, \( \mu \) is fixed. In the fast gradient algorithm, \( \mu \) is adaptively updated in the following manner:

\[ \mu_k = \frac{1}{P_R} \]  

(4.4.3-14)

where

\[ P_R = \lambda P_{R-1} + Y_k^2 \]  

(4.4.3-15)

and \( \lambda \) is called the "forgetting factor," a value between 0.0 and 1.0.

In defining the Fortran code for the fast Gradient algorithm, the recursion relationships developed in this section are summarized as follows:

\[ d_{p+1,k} = \lambda d_{p+1,k-1} + 2 e^T_{p,k} r_{p,k-1} \]  

(4.4.3-16)
\[ d^x_{p+1,k} = \lambda_d d_{p+1,k-1} + 2 \varepsilon_{p,k}^x r_{p,k-1} \]  
(4.4.3-17)

\[ R^2_{p,k} = \lambda R^2_{p,k-1} + |\varepsilon_{p,k}|^2 + |r_{p,k-1}|^2 \]  
(4.4.3-18)

\[ K_{p+1,k} = \Delta_{p+1,k} / R^2_{p,k} \]  
(4.4.3-19)

\[ K^x_{p+1,k} = \Delta^x_{p+1,k} / R^2_{p,k} \]  
(4.4.3-20)

\[ \varepsilon_{p+1,k} = \varepsilon_{p,k} - K_{p+1,k} r_{p,k-1} \]  
(4.4.3-21)

\[ \varepsilon^x_{-p+1,k} = \varepsilon^x_{p,k} - K^x_{p+1,k} r_{p,k-1} \]  
(4.4.3-22)

\[ r_{p+1,k} = r_{p,k-1} - K_{p+1,k} \varepsilon_{p,k} \]  
(4.4.3-23)

After any user specified number of iterations defined as number of ALE observations (k in above equations), the recursive functions may be stopped, and the predictor coefficients and weights updated as

\[ A_{p+1,i} = A_{p,i} \cdot K_{p+1} B_{p,i-1} \]  
(4.4.3-24)

\[ B_{p+1,k} = B_{p,i-1} \cdot K_{p+1} A_{p,i} \]  
(4.4.3-25)

\[ W_{p+1,i} = W_{p,i} \cdot K^x_{p+1} B_{p,i-1} \]  
(4.4.3-26)
Before the fast Gradient subroutine is run, the following initial conditions are set:

\[ \Delta_{0,-1} = 0 \]  \hspace{1cm} (4.4.3-27)

\[ \epsilon_{-0,k} = r_{-0,k} = y_k \]  \hspace{1cm} (4.4.3-28)

\[ \epsilon_{0,k}^x = x_{-k-1} \]  \hspace{1cm} (4.4.3-29)

\[ R_{0,-1} = S \text{ (a priori estimate)} \]  \hspace{1cm} (4.4.3-30)

\[ A_{0,i} = B_{0,i} = 1 ; i = 0 \]  \hspace{1cm} (4.4.3-31)

\[ A_{0,i} = B_{0,i} = 0 ; i > 0 \]  \hspace{1cm} (4.4.3-32)

\[ W_{p,0} = 0 \]  \hspace{1cm} (4.4.3-33)

where \( i \) equals zero through \( N \) (number of filter weights). The relationship is coded in the program listing found in Appendix A. The subroutine FGRAD performs the required initialization and weight computation functions as described in this section.
5.0 Experimental Procedure

To conduct the signal processing simulations, a Fortran program was designed to generate the appropriate environment conditions and desired filter responses. The four filter algorithms are entered in the program as subroutines, as are the Fourier Transform routines for plotting the pre- and post-filtered data. The program is run on a Macintosh II computer with network access that provides tools for plotting the results. After each simulation, the filter computation time and obtained Signal to Noise Ratio are recorded. The frequency domain representation of the input and output data are stored and later plotted for comparison. The simulations for the four filter algorithms are applied to various combinations of number of data observations and filter weights.

5.1 Hardware Requirements

The hardware required for the DSP simulation created for this report must be capable of generating, filtering, and storing the resultant data in the application program. In some cases of comparison, the data may be generated in advance, and read in by the program. For each case, large data arrays are required for data storage in the filtering and plotting stages. At least 1MB of RAM is recommended for the host computer to store the data arrays, and to limit time consuming disk I/O. There are no requirements for processor types, except that a fast processor, and co-processor will be of great value if many iterations of the program are performed. Other hardware recommendations include a hard disk drive to store the data files, and a graphics printer for hard copies of data plots.
5.2 Software Requirements

The software requirements for the host computer include the need for a high level scientific language compiler, a graphing routine or package to plot the output data, and an operating system capable of supporting the program. The source code for the DSP and ALE function simulator is written in Fortran for the Macintosh II computer since (when programming began) it was the only language supporting the MC68020 32bit architecture. The program is written in a standard Fortran 77 format with Macintosh "Include" files accessed for algorithm timing.

The outputs from the simulator software are three files containing one column of real data that corresponds to magnitudes in decibels of the Fast Fourier Transforms of the input, output, and filter weight data. The input and output Spectrum data contain 256 points that represent the power spectral density from 0.0 through 0.5 normalized Hertz where 1.0 Hz norm. equals the system sampling frequency. The frequency response of the filter weights contains N points ( \( N = \) number of weights) that describe the attenuation characteristics of the Adaptive Filter. The N points are distributed evenly through 0-5 Hz norm., and the region around each of the N points is the FIR filter energy bin. The number of bins is directly proportional to the filter resolution. Data from the three output files is plotted on the Macintosh computers using a software package called "Cricket Graph."
5.3 Parameter Specifications

5.3.1 Specifications for Signal Environment

For comparison of the filter types, a specific test environment must be selected and applied to each filter. For this research on the filtering capabilities for multiple switching tones in correlated noise, the sample data shall contain 32,768 (32k) samples. In this data, a variable frequency tone will always be present at one of three equiprobable frequencies being reselected after every 50 samples. The noise source added to the signal data is wideband and correlated. It is generated by lowpass filtering data that is an amplitude modulated sine wave with the amplitude determined by a gaussian distributed random variable (see Chapter 4).

5.3.2 Filter Parameter Limits

A range of filter parameter test values is specified for the comparison of the different filter types. Values for number of filter taps and number of data observations must be determined and applied to each applicable filter under test. The number of filter taps implemented shall be set to even powers of two. This is done to maintain accuracy in the Spectral representation of the filter weight transfer functions as generated by taking an FFT of the weight values. For small numbers of filter taps, the FFT bins are wide and filter resolution will be poor. If the bins are too wide, the filter transfer function will not clearly discern the differences between the frequencies of the switched tones. For large numbers of filter taps, the bins are narrow and resolution high. The major drawback of generating large numbers of filter taps is increased computation times. Doubling the
number of filter taps approximately doubles the computation time for the filter taps, and also doubles the time required to pass the 32k data set through the FIR filter. The range of filter taps used in this experiment will be from 16 to 128 taps (16, 32, 64, 128).

The other critical parameter for computation of the ALE filter weights is the number of data samples observed. In order to converge to an estimate of the signal data, a portion of the 32k samples must be 'studied' by the ALE algorithm. Increasing the number of observations provides the algorithm with more information which should produce more accurate filter weights (*this may not be the case for all algorithms as will be seen). A minimum number of observations shall be computed as follows for multiple switching tones occurring at three equiprobable discrete frequencies (one bit equals 50 samples of a sinewave at one frequency):

The probability of at least one bit generated at all three frequencies, \( P(\geq 1\text{bit/frequency}) \), equals one minus the probability of zero bits generated at any one of the frequencies, \( 1 - P(0 \text{ bits at any frequency}) \). For a binomial distribution with 99% minimum probability of success, \( P(\text{success}) = 1 - \frac{n!}{(n-y)y!} \cdot p^y \cdot q^{n-y} \) where \( n = 12 \) bits. At 50 samples per bit, \( 12\text{bits} \times 50\text{samples/bit} = 600\text{samples} \) minimum observation requirement.

The maximum number of observations chosen for the experiment is based on the cost of adding additional observation samples. Doubling the number of observations approximately doubles the ALE weight computation time, and (for most filters) provides an increase in the obtained SNR improvement. As the number of observations increases, a
limit to the convergence of weight values is approached. If more observations are made after the convergence rate slows, the weight computation time will increase at a higher rate than the measure of additional noise suppression. Figure 5.3-1 shows the convergence to the weight minimum Mean Squared Error as a function of more algorithm observations (based on measurements performed on these algorithm types [27] [15]). For small numbers of observations, the algorithms converge at a fast rate. The optimum value of selected observations is considered to be in the knee of the curve where the convergence rate begins to slow down. For the experiment in this report, the range of observations shall be from one thousand to eight thousand including: 1000,2000,3000,4000,6000,8000.

5.4 Digital Signal Processor Simulation

In the signal processing simulator software (ALE.FOR), the desired signal and noise source is generated filtered and plotted. When the program is run, the user supplies the necessary information in response to the screen prompts. The program performs the signal processing functions then places the output data in standard text files that are readable by the graphing software.

The program first generates the noiseless three tone, variable frequency, switching signal. A random digit corresponding to one of the tone frequencies is generated, and the next fifty samples are a sine wave of that frequency. Following generation of each fifty samples, another random number is generated to select the next tone frequency. Thirty-two thousand (32k) signal samples are created. After signal generation, 32k samples of correlated noise are added to the signal.
Figure 5.3-1 Comparison of Convergence Rates
Following the signal generation portion of the program, the chosen filter subroutine is called. Within each subroutine, the user is prompted to enter the necessary filter characteristics. The subroutine then generates a set of filter weights that is passed back to the main program. Each of the four filter subroutines requires input of the number of desired filter weights. In the LMS, Fast RLS, and Fast Gradient filters, the user must also enter the number of data observations (the DFT routine uses all 32k samples for these tests). After generating the filter weights, the program uses the weight values to FIR filter the entire 32k data set. The arrays containing the filtered and unfiltered data are passed to the data plot subroutine where the FFTs are computed and the data are output to text files (see Chapter 2.2).

For the purpose of unbiased algorithm comparisons, performance data is computed during each run of the program which provides computation times and achieved SNR improvements. Timing data is collected by the computer using a Macintosh 'Toolbox' system call. Computation times for calculating the filter weights, and for filtering the data are output to the screen for each program pass. The values for SNR data are determined in two ways; the first is to observe the data by sight, and the second is to compute it in the program. Visual observations of the input and output FFT plots are the first step in determining which filters are better than others. Since the same input data is used for all tests, the output data can be plotted on the same scales and overlaid on the computer screen, on paper, or on clear acetate viewgraphs. The graphed frequency spectrums will reveal portions of the data that are filtered extremely well, and will show where portions of the data may not be well filtered. The computer determination of SNR will be a measure of the peak signal power divided by the average noise power using the spectral data calculated in
the plotting subroutine. The peak signal power in decibels is twenty times the log of the sum of peak voltages at each of the three switched tone frequencies or

\[
\text{Signal power(dB)} = 20 \log (V_{1\text{max}} + V_{2\text{max}} + V_{3\text{max}}). \quad (5.4-1)
\]

The average noise power in decibels is calculated in this report as the average value of the spectrum power beyond the three filter passbands. Out of the 256 point FFT plot, the passbands shall be considered the points at V_{\text{max}} for each frequency plus and minus 10 points. The total number of points allocated for the three passbands is 60, and the other 196 data points are averaged as

\[
\text{Noise power(dB)} = \left( \sum_{i=1}^{196} \text{power(dB)}_i \right) / 196. \quad (5.4-2)
\]

The output SNR minus the input SNR will be one of the benchmark measures of filter capability.

The procedure is summarized as follows:

1: Select the test algorithm to filter three random switching tones in correlated noise from - DFT, LMS, Fast RLS, Fast Gradient.

2: Enter number of filter observations (except DFT) from - 1k, 2k, 3k, 4k, 6k, 8k.

3: Enter number of filter weights from - 16, 32, 64, 128.
4: Manually record in notebook the value of ALE weight computation time (seconds).

5: Record input SNR from Power Spectral Density (PSD) of input signal using FFT plot calculation.

6: Record output and output minus input SNRs from PSD of output signal using FFT plot calculation.

7: Graph input versus output PSD (Cricket Graph program on Macintosh) using data from FFT plot subroutines.

8: Perform filter and parameter tradeoff analysis.
Chapter 6: Results

6.0 Results

The four filter algorithms are tested under the same conditions for input environment, and signal characteristics. Results include comparisons of filter Signal to Noise Ratio improvements, and filter complexity measured in weight computation time. Observations are based on these measurements versus the selection of multiple filter parameters (number of filter taps and number of observed data samples). Output from the filter performance data files are graphed for visual and computational comparison.

Algorithm testing begins with generation of the signal environment. Selected parameters for the environment are three possible tone frequencies with random equiprobable likelihood for appearance in time. One of the three tone frequencies is randomly selected every 50 data samples. The additive noise source is a wideband correlated data stream as described in Chapter 4. One of the filter types is selected, and the associated parameters entered. After filtering the data with the generated weights, the output data are generated, and the respective SNRs calculated.

6.1 The Digital Fourier Transform Filter

The first filter tests are performed on the Digital Fourier Transform classical version of an adaptive filter. Since this filter averages the overall spectral density of the incoming signal, areas of high correlation (tones) will form bandpass regions in the filter weight response. The DFT algorithm performs simple computations, and is stable for all forms of
Data from 64tap DFT

Figure 6.1-1: Classical DFT Filter
dynamic signal environments. Unfortunately, the DFT filter is not capable of distinguishing wideband correlated noise from narrow band correlations like the switching tones, and passes the regions of wideband correlated noise with little attenuation. If there are regions of correlated noise that are higher in magnitude than some of the signal tones, then the output SNR may not be acceptable. Figure 6.1-1 shows the output from one of the DFT filter simulations. From this figure, it is apparent that the DFT filter is not capable of removing wideband correlated noise. Data for each of the DFT weight values are plotted in Appendix B.

6.2 Least Mean Squares Filter

The Least Mean Squares (LMS) adaptive filter algorithm was first run for the data using 1000 observations on a 16 tap filter. The result was a diverging set of undefined numerical values for the weights. To check the filter capabilities, a smaller number of observations were applied, and the algorithm run again. Using 50 observations, the LMS algorithm produced a defined set of weights. However, in this small amount of data, the weights were only able to converge to one of the three tone frequencies. In Figure 6.2-1, it is apparent that the LMS routine attenuates some of the correlated noise source, but also attenuates one of the tone frequencies. The LMS filter weights begin to diverge out of the range of the plotting range after about 500 observations in the switching three tone signal environment (Figure 6.2-2). For numbers of observations greater than 500, the filter weights diverge by becoming very large, and the FIR filters do not pass readable data.
Data from 50 obs. Unbiased LMS

Figure 6.2-1: 50 Observation LMS
Figure 6.2-2: 500 Observation Unbiased LMS
Data From 3k obs. 32tap LMS

Figure 6.2-3: 3000 Obs. 32 Tap Biased LMS
In order to stabilize the environment, the incoming signal data was compressed to values between zero and plus one volt from the original data ranging from minus one to plus one volt. Response of the LMS filter became much more well behaved, and no more of the output data and filter weight values diverged. Figure 6.2-3 is a typical output response plot of the modified LMS adaptive algorithm. From this plot it is seen that the weights converge to all three tone frequencies, and the noise levels are significantly reduced. The remaining problem with the modified routine is that compressing the signal in this matter added a d.c. bias to the data, and the LMS filter weights interpreted the bias as a highly correlated portion of the incoming signal. The output of the biased LMS filter includes a very high level of noise in the lowest frequency range around zero Hertz. If important signal information is located in the lower frequencies of the received data, the high level of noise may interfere with and hide desired signal information. A complete set of LMS data results for three switching tones of biased samples are plotted in Appendix B.

6.3 Fast Recursive Least Squares

The Fast Recursive Least Squares (Fast RLS) adaptive filter algorithm was run for several different size sets of observations, and the result was consistent divergence of the filter weights. Above 50 observations the weight computations became numerically unstable; but at 50 observations, the filter produced real valued weights (see Figure 6.3-1). Since the signal tone switches frequency every 50 samples, it is likely that the sudden environment change causes instability in the Fast RLS algorithm. In these initial tests, the "forgetting factor" omega is set to 1.0 (other values between 0.0 and 1.0 all caused weight divergence). To further test the algorithm, the environment is stabilized as in the LMS routine by compressing the signal data to within zero and plus one volts.
Data from 50 obs. Unbiased RLS

Figure 6.3-1: 50 Observation Unbiased Fast RLS
Data from 50 obs. Biased RLS

Figure 6.3-2: 50 Observation Biased Fast RLS
In the compressed signal environment, the Fast RLS algorithm exhibits distinct changes in behavior. Stability is gained in computations through several thousand observations using 16 and 32 tap filters. However, 64 and 128 tap filters become much less stable, and cause early divergence in filter characteristics (see Fast RLS data in Appendix 2). In Figure 6.3-2, it is evident that the filter has passbands in the frequency regions of the signal tones, and it is also seen that the d.c. bias introduced in the signal compression passes unwanted low frequency data.

6.4 Fast Gradient Filter

In the implementation of the Fast Gradient Lattice Filter, several new properties are exhibited by observing the output filtered data. First, the unbiased data is easily adapted to by the simulated algorithm; and second, the computed filter weights keep converging and never become unstable with additional observations. From the full range of collected data (see Appendix B), a performance tradeoff is conducted to select the best filter parameters.

The output response from the Fast Gradient filter in Figure 6.4-1 indicates that the filter weights have adjusted to cancel the wideband correlated noise source and have formed passbands around the frequencies of the switching tones. The output spectrum shows a fairly uniform peak level of noise and a high signal level in the passbands. Figure 6.4-2 is the frequency response of the filter weights in the selected filter. From this response, it can be seen that the filter attenuation is high in the areas of strong noise, and low in the low noise regions providing some measure of noise whitening. Attenuation is low in the signal passband regions for each tone.
Figure 6.4-1: 3000 Observation 32tap Fast Gradient
Figure 6.4-2: Optimized Filter Weight Frequency Response
To select the optimum Fast Gradient filter parameters, twenty-four runs were conducted using the full range of observations and numbers of taps (1k-8k observations, 16-128 weights). Table 6.4-1 shows the weight computation time and achieved SNR improvement for each combination of parameters. The results of the measurements are plotted in Figures 6.4-3 and 6.4-4. The plot of time data in Figure 6.4-3 shows a linear relationship between number of observations and number of filter weights versus computation time for the tap values. The smaller numbers of observations and numbers of taps each reduce the times by one half for each 50% size reduction. Therefore, the smallest parameter sizes must be chosen that provide good filter performance. From Figure 6.4-4, the performance curves show large increases in attained SNR improvement for increases of small numbers of observations. However, as the number of observations gets large, the net increase in SNR becomes smaller as more observations are made. The performance curve for the 32 tap filter is higher than the other sets of taps, and thus an optimum filter will be chosen from this data. The knee in the performance curve where SNR increases slow down appears at three thousand observations, therefore the computational time increase in using more than 3000 observations will not be considered to be worth while.

Using the optimized Fast Gradient Lattice Adaptive filter as an Adaptive Line Enhancer in the signal processing system shall be performed as follows:

1. The received data is digitized and stored in a 32k buffer of samples.

2. The Fast Gradient form of the ALE is run for 3000 iterations using the first 3000 data samples.
Table 6.4-1: Computation Times and SNR Improvements For 3000 Obs.
32 Tap Fast Gradient ALE

<table>
<thead>
<tr>
<th>Number of Observations</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k</td>
<td>6.40 sec</td>
<td>12.63 sec</td>
<td>25.07 sec</td>
<td>49.93 sec</td>
</tr>
<tr>
<td></td>
<td>6.8 dB</td>
<td>7.1 dB</td>
<td>6.0 dB</td>
<td>4.1 dB</td>
</tr>
<tr>
<td>2k</td>
<td>11.92 sec</td>
<td>23.58 sec</td>
<td>46.90 sec</td>
<td>93.52 sec</td>
</tr>
<tr>
<td></td>
<td>7.0 dB</td>
<td>7.8 dB</td>
<td>7.3 dB</td>
<td>6.5 dB</td>
</tr>
<tr>
<td>3k</td>
<td>17.43 sec</td>
<td>34.53 sec</td>
<td>68.72 sec</td>
<td>137.10 sec</td>
</tr>
<tr>
<td></td>
<td>7.9 dB</td>
<td>8.7 dB</td>
<td>8.4 dB</td>
<td>7.0 dB</td>
</tr>
<tr>
<td>4k</td>
<td>22.93 sec</td>
<td>45.48 sec</td>
<td>90.57 sec</td>
<td>180.70 sec</td>
</tr>
<tr>
<td></td>
<td>8.1 dB</td>
<td>8.9 dB</td>
<td>8.5 dB</td>
<td>7.9 dB</td>
</tr>
<tr>
<td>6k</td>
<td>33.98 sec</td>
<td>67.37 sec</td>
<td>134.22 sec</td>
<td>267.87 sec</td>
</tr>
<tr>
<td></td>
<td>8.3 dB</td>
<td>9.6 dB</td>
<td>9.2 dB</td>
<td>8.9 dB</td>
</tr>
<tr>
<td>8k</td>
<td>44.98 sec</td>
<td>89.28 sec</td>
<td>177.85 sec</td>
<td>355.02 sec</td>
</tr>
<tr>
<td></td>
<td>8.3 dB</td>
<td>9.9 dB</td>
<td>9.7 dB</td>
<td>9.6 dB</td>
</tr>
</tbody>
</table>
Fast Gradient Computation Times

Figure 6.4-3: Fast Gradient Computation Times
Fast Gradient Filter Performance

![Graph showing Fast Gradient SNR Improvements]

Figure 6.4-4: Fast Gradient SNR Improvements
3. The 32 calculated weight values are stored as a vector array in memory.

4. The entire 32k data samples are filtered using the 32 taps as an FIR filter.

5. The filtered data are sent to the next stages of signal and information processing.

Results from the 3000 point Fast Gradient ALE are plotted in the time domain. Figure 6.4-5 shows the filtered ALE time output versus the noisy signal input.
Figure 6.4-5: Optimized Filtered Output Versus Noisy Input
Chapter 7: Summary

7.0 Conclusions

From the obtained experimental results, the optimal choice for an Adaptive Line Enhancer for randomly switching tones in correlated noise is the Fast Gradient Lattice Filter. In the experiment simulated in this thesis, the best Fast Gradient filter parameters are 32 taps and 3000 data observations. A comparison with the Digital Fourier Transform (DFT), Least Mean Squares (LMS), and Fast Recursive Least Squares (Fast RLS) filters provides such evidence. The DFT algorithm is a simple and fast filtering method, however it is not capable of removing received sources of correlated noise. The DFT filter generates weight values that are representations of the overall average spectral content. If the noise in part of the received frequency band is correlated and high in power, then the DFT filter will form taps to pass those frequencies with low attenuation.

The LMS filter is generally regarded as a robust algorithm capable of adapting to an environment with low computational cost. It is not a fast converging algorithm, but with a very small convergence factor, it will converge to stable received environments and will attenuate most sources of noise. In a dynamic environment such as the one simulated in this report, the LMS algorithm can lose its stability and will generate diverging tap values. By compressing the data between 0.0 and 1.0 volts, the LMS algorithm again converges but introduces a d.c. bias in the spectrum of tap values. Low frequency noise sources are passed with insufficient attenuation.
In the Fast RLS algorithm, stable signal environments are adapted to very quickly. If too many observations are made, the filter parameters begin to diverge, and filter performance is degraded. In the dynamic switching environment, the Fast RLS filter begins to suffer as soon as the received information begins to change frequencies. Compressing the incoming data as done with the LMS routine prolongs the useful range of the Fast RLS algorithm, however the passband becomes spoiled with the d.c. and low frequency components introduced with data biasing.

When a simple and stable environment is encountered, one may implement any one of these filters to improve the received Signal to Noise Ratio. In Figure 5.3-1, a comparison of the convergence properties shows that filter weights may be quickly generated by a Fast RLS algorithm, or more slowly convergent weights may be efficiently generated by the LMS algorithm. If the received signal environment becomes more dynamic and undergoes many characteristic changes in time, then these filters may not improve the system SNR. In fact, the LMS or Fast RLS filters may worsen the received signal quality if the filter becomes unstable and the weight values diverge from the minimum error values. Quantization errors and precision limitations can be blamed for some of the failures. In the case of the Fast RLS algorithm, the convergence is over committed to a present received signal, and a sudden change drives the algorithm to an unstable state of operation. Figure 7.0-1 shows an approximation to the filter behavior characteristics in the multi-tone switching environment simulated and observed in this report.
Figure 7.0-1: Approximate Convergence Characteristics in Multiple Switching Tone Environment
7.1 The Fast Gradient Adaptive Line Enhancer

Implementing the Fast Gradient algorithm in a signal processing system introduces a highly reliable adaptive filter that can be tailored to one's needs or limitations. The Fast Gradient algorithm is stable and not as sensitive to limits in computing precision as the LMS and Fast RLS algorithms. In a dynamic signal environment, the Fast Gradient Lattice ALE can be used to filter incoming digitized samples to a desired measure of quality. If abundant processing time is available, then a large number of samples may be observed forming a large set of filter weights allowing the filter to converge to the best signal estimate. For time restricted systems, fewer samples may be observed and fewer weights computed to provide an acceptable measure of SNR improvement. Reducing the number of observations conserves ALE weight computation time, and reducing the number of filter taps reduces both computation and data FIR filtering times. Therefore, the best use of filter parameters is to minimize the necessary number of filter weights, and to observe as many samples as possible that still provide substantial increases in output SNR. In this report, 32 filter weights provided the best filtering characteristics for the received multi tone signal. For other signal environments where the tones are more closely spaced, the filter bins may need to be made smaller to distinguish the different tones, and additional weights may be necessary.

7.2 Future Study

Additional work that may provide more efficient means for Adaptive Line Enhancement of dynamic digital signal environments includes the study of other lattice filter implementations, and the study of numerical precision limitations and effects. Other forms
of lattice filters have been developed such as the Recursive Least Squares Lattice filter, Bulk and Joint Lattice filters, and several versions of Normalized Lattice filters. It should prove to be a worthwhile experiment to compare the different lattice forms in this type of signal environment. In some tests conducted [29,15], the Normalized Lattice forms have not performed very well. On the study of numerical precision limitations, much research is being conducted, and this author hopes to test the Fast Gradient Lattice filter capabilities in more restricted precision environments. If the Fast Gradient filter algorithm can be implemented with integer numeric functions, then tighter computer time and memory constraints can hopefully be met.
References

8.0 References


Appendix A

Computer Listing of ALE Simulation
PROORAM ALE

Written by Robert D. Ritter for Masters Thesis
Adaptive Line Enhancer for Multiple Switching Tones in a Correlated Noise Environment

April 1988

PURPOSE:

THIS PROGRAM IMPLEMENTS ALL OF THE ADAPTIVE LINE ENHANCER ALGORITHMS AND
PLOTS THE FREQUENCY SPECTRUM FOR THE RAW SIGNAL, PROCESSED SIGNAL AND TAP VALUES.

Include Macintosh Timer Routine

INCLUDE :INCLUDE FILES:EVENT.INC

IMPLICIT NONE

INTEGER NUMBTONES,NZSRC,FILTYP,BIT,I,J,RNDSTART,RNDX,RND,
RNDM,STAT,LENGTH,DELTAT,A

INTEGER TIMBEF,TIMAFf,DELTIM,TOOLBX

REAL X(32768),OMEGA,SHIFT,TWOP1,OMSHFT1,OMSHFT2,PNT1,FREQ,
XN1,XN2,TEMPl,TEMP2,INTER,WEIGHT(0:256),XHAT(32768),SPB,
TIMDIF,SQ,AV,VAR,SNR

OPEN(UNIT=2,FILE='ALEX.OUT,STATUS='NEW')
OPEN(UNIT =3,FIl...E='ALEXHAT.OUT,ST ATUS='NEW)
OPEN(UNIT =4,FIl...E='ALEWEIGHT.OUT,ST ATUS='NEW)

DATA XHAT/32768*0.0/
DATA X/32768*0.0/
DATA XN1,XN2/2*0.0/
DATA TWOP1/6.283185/
DATA RNDSTART/1/!RANDOM NUMBER INIT
DATA SPB/50.0/!SAMPLES PER BIT - bps FSK
DATA WEIGHT/257*0.0/
DATA INTER/0.1/

WRITE (*,10) 'Enter integer number of tones (1 OR 2 OR 3):
READ(*,30) NUMBTONES
WRITE (*,10)'
NUMBTONES = 3

WRITE (*,10) 'Enter normalized frequency of first tone:'
READ(*,20) OMEGA
WRITE (*,10) "
OMEGA = 0.2

IF (NUMBTONES .GE. 2) THEN
    WRITE (*,10) 'Enter normalized shift frequency:'
    READ(*,20) SHIFTS
    WRITE (*,10) "
    SHIFTS = 0.0
C READ (*,20) SHIFT
C WRITE (*,10) "*
C END IF
SHIFT = 0.1
C
WRITE (*,10) 'Enter noise source:'
WRITE (*,10) ' (0) = No noise'
WRITE (*,10) ' (1) = Gaussian noise'
WRITE (*,10) ' (2) = Correlated noise'
READ (*,30) NZSRC
WRITE (*,10) 'Enter Signal to Noise Ratio (SNR dB):'
READ (*,20) SNR
WRITE (*,10) "
SNR = 10.0**(SNR/20.0)
C WRITE (*,20) SNR
C
WRITE (*,10) 'Enter Adaptive filter type'
WRITE (*,10) ' (0) = Classical DFF'
WRITE (*,10) ' (1) = LMS'
WRITE (*,10) ' (2) = Fast RLS'
WRITE (*,10) ' (3) = Fast Gradient'
READ(*,30) FILTYP
WRITE (*,10) "
C 10 FORMAT (1H, A)
20 FORMAT (1H, F15.6)
30 FORMAT (1H, 110)
C
WRITE (*,10) 'ADAPTIVE FILTERING PROCESS UNDERWAY...'
C *** GENERATE SWITCHING TONE SIGNAL ***
C
OMEGA = OMEGA * TWOP
OMSHFT1 = OMEGA + SHIFT * TWOP
OMSHFT2 = OMSHFT1 + SHIFT * TWOP
C
BIT = 0
C
IF (NUMBTONES .EQ. 1) THEN
DO (I=1,32768)
X(I) = SNR * COS(OMEGA*I)
TEMP1 = X(I)
END DO
GOTO 100
END IF
C
IF (NUMBTONES .EQ. 2) THEN
40 CALL RANDOM(PNT1,RNDSTART,RNDX,RNDA,RNDM)
IF (PNT1 .LE. 0.0) GOTO 40
C
IF (PNT1 .LE. 0.5) THEN
FREQ = OMEGA
ELSE
FREQ = OMSHFT1
END IF
C
DO (I = 1,32768)
C
IF (AINT(I/SPB) .GT. BIT) THEN
BIT = BIT +1
50 CALL RANDOM(PNT1,RNDSTART,RNDX,RNDA,RNDM)
IF (PNT1 .LE. 0.0) GOTO 50
C
IF (PNT1 .LE. 0.5) THEN
FREQ = OMEGA
ELSE
FREQ = OMSHFT1
END IF
END IF
C
X(I) = SNR * COS(FREQ*I)
END DO
GOTO 100
END IF
C
C
IF (NUMBTONES .EQ. 3) THEN
60 CALL RANDOM(PNT1,RNDSTART,RNDX,RNDA,RNDM)
IF (PNT1 .LE. 0.0) GOTO 60
C
IF (PNT1 .LE. 0.3333) FREQ = OMEGA
IF (PNT1 .GE. 0.3333) FREQ = OMSHFT1
IF (PNT1 .GE. 0.6667) FREQ = OMSHFT2
END IF
C
DO (I = 1,32768)
C
IF (AINT(I/SPB) .GT. BIT) THEN
BIT = BIT +1
70 CALL RANDOM(PNT1,RNDSTART,RNDX,RNDA,RNDM)
IF (PNT1 .LE. 0.0) GOTO 70
C
IF (PNT1 .LE. 0.3333) FREQ = OMEGA
IF (PNT1 .GE. 0.3333) FREQ = OMSHFT1
IF (PNT1 .GE. 0.6667) FREQ = OMSHFT2
END IF
C
X(I) = SNR * COS(FREQ*I)
END DO
GOTO 100
END IF
C
CALL QUIT ! Undefined number of switching tones
C
100 IF (NZSRC .EQ. 0) GOTO 200 ! *** NO NOISE ***
C
IF (NZSRC .EQ. 1 ) THEN ! *** GAUSSIAN NOISE ***
DO (I = 1,32768)
CALL GAUSS(XN1,XN2,RNDSTART,RNDX,RNDA,RNDM)
X(I) = X(I) + XN1
END DO
GOTO 200
END IF
C
Correlated noise is Cyclo-Stationary Process
C
IF (NZSRC .EQ. 2) THEN ! *** CORRELATED NOISE ***
TEMP1 = 0.0
TEM2 = 0.0
DO (I=1,132768)
CALL GAUSS(XN1,XN2,RNDSTART,RNDX,RNDA,RNDM)
TEM2 = XN1*(SIN(INTER*I))
X(I) = X(I) + TEM2 + 0.9*TEM1 ! * Low Pass Filter
TEM1 = TEM2
END DO

GOTO 200
END IF

CALL QUTS ! *** UNDEFINED NOISE SOURCE ENTERED ***

200 STAT = 0

*** CALCULATE INPUT SNR

DO (I=1,100)
WRITE (*,11) 'X = ', X(I)
END DO
SQ = 0.0
AV = 0.0
VAR = 0.0
SNR = 0.0
DO (I = 1,132768)
    SQ = SQ + X(I)**2
    AV = AV + ABS(X(I))
END DO
VAR = SQ/(I-1) - (AV/(I-1))**2
SNR = (SQ/(I-1)) - VAR)/(VAR)
SNR = 10.0· LOG10(SNR)
WRITE (*, 11) 'SQ = ',SQ/(I-1)
WRITE (*, 11) 'AV = ',AV/(I-1)
WRITE (*, 11) 'VAR = ',VAR
WRITE (*, 11) 'SNR = ',SNR
11 FORMAT ( IH, A, F18.6 , A )

*** CALL APPROPRIATE FILTER SUBROUTINE ***

TIMBEF = TOOLBX(TICKCOUNT)

IF (FILTYP.EQ.0) CALL DFT(X,STAT,WEIGHT,LENGTH)
IF (FILTYP.EQ.1) CALL LMS(X,STAT,WEIGHT,LENGTH)
IF (FILTYP.EQ.2) CALL RLS(X,STAT,WEIGHT,LENGTH)
IF (FILTYP.EQ.3) CALL FGRAD(X,STAT,WEIGHT,LENGTH)

IF (STAT.EQ.0) CALL QUTS ! *** NO FILTER CALLED ***

TIMAFT = TOOLBX(TICKCOUNT)
DELTIM = TIMAFT - TIMBEF
TIMDIF = DELTIM / 60.0

WRITE (*, 19) 'ALE time (sec) = ',TIMDIF,CHAR(9),DELTIM
19 FORMAT ( IH , A , F14.6 , A , 110)
WRITE (*,10) ' Filtering...

TIMBEF = TOOLBX(TICKCOUNT)
DO (I = 1, 32768)
DO (J = 0, LENGTH - 1)
   IF(I - I - DELTA .GT. 0) THEN
      XHAT(I) = XHAT(I) + WEIGHT(I) * X(I - I - DELTA)
   END IF
END DO
END DO

TIMAF = TOOLBX(TICKCOUNT)
DELTIM = TIMAF - TIMBEF
TIMDIF = DELTIM / 60.0

WRITE (*, 19) 'Filter time (sec) = ', TIMDIF, CHAR(9), DELTIM

WRITE (*, 10) 'Begin data plot ...

CALL PLOTX(X, 512, 9, 2)
CALL PLOTX(XHAT, 512, 9, 3)
CALL PLOTW(WEIGHT, LENGTH)
CALL QUIT
STOP
END

********************************************************************************

SUBROUTINE DFT(X, STAT, WEIGHT, LENGTH)

Subroutine to perform Classical DFT filter routine

IMPLICIT NONE

REAL X(513), X(I), WEIGHT(I)
INTEGER NUM, LENGTH, THESSVAL, I, STAT, IJ

WRITE(*, 401) 'Enter number of DFT iterations '
READ(*, 402) THESSVAL
WRITE(*, 401) ' '
THESSVAL = 32768

WRITE(*, 401) 'Enter number of Filter Weights '
READ(*, 402) LENGTH
WRITE(*, 401) ' '

401 FORMAT(1H, A)
402 FORMAT(1H, I10)

NUM = INT(THESSVAL / LENGTH) - 1

DO (I = 0, NUM)
   DO (J = 0, LENGTH)
      WEIGHT(I) = WEIGHT(I) - X(I * LENGTH + J) / NUM
   END DO
END DO

DO 410 IJ = 0, LENGTH
WRITE (*, 411) 'WEIGHT',WEIGHT(IJ)
CONTINUE
FORMAT ( 1H , A, F10.6 )

STAT = 1
RETURN
END

SUBROUTINE LMS(X,STAT,WEIGHT,LENGTH)
Subroutine to perform Least Mean Squares adaptive algorithm
IMPLICIT NONE

REAL X(513),X1(I),U,E,A(513),WEIGHT(I)
INTEGER THESVAL,LENGTH,I,STAT,I

DATA X1/I513/0.0I
DATA A/513*0.0I
WRITE(*,401) 'Enter number of Adaptive iterations'
READ(* ,402) THESVAL
WRITE(*,401) , ,
WRITE(*,401) 'Enter number of Adaptive Filter Weights '
READ(· ,402) LENGTH
WRITE(*,401)

C WRITE(*,401) 'Enter Convergence Rate, µ ';
C READ(* ,403) U
C WRITE(*,401) •

401 FORMAT( 1H, A )
402 FORMAT(1H,I10)
403 FORMAT(1H,F10.6)

DO (I = 1,THESVAL)
A(I) = X(I+100)
END DO

DO 400 I=1,THESVAL

400 PERFORM SHIFT FUNCTION
DO (K = LENGTH+1.2.-1)
X1(K) = X1(K-1)
END DO
X1(1) = X(1) !***** /10.0 +0.5 ! *** ADD D.C. BIAS

PERFORM LMS ADAPTIVE ALGORITHM

C *** SEE APPENDIX 9.B "PROGRAM OF LMS ADAPTIVE ALGORITHM"
C Digital Spectral Analysis With Applications - S.L. Marple
... Insert program here and modify to run with REAL data

```
EF = ABS(EF)  ! ***

DO (K = 1,LENGTH)
   A(K) = A(K) - 2.0 *U*EF*X(K+1)
END DO
```

400 CONTINUE

```
CALCULATE THE PREDICTOR COEFFICIENTS

DO 410 IJ = 0,LENGTH
   WEIGHT(IJ) = A(IJ)
   WRITE (*, 411) 'WEIGHT', WEIGHT(IJ)
410 CONTINUE
411 FORMAT (1H1, A, F10.6)
```

```
STAT = 1
RETURN
END
```

```
SUBROUTINE RLS(X1,STAT,WEIGHT,LENGTH)
Subroutine to perform Fast Recursive Least Squares adaptive algorithm

IMPLICIT NONE

REAL SAMPLE,X(513),X(1),AF(0:512),AB(512),C(513),EF,EPF,EB
REAL FF,PB,GAMMA,OMEGA,SAVE,EPB,TEMP,HOLD
REAL WEIGHT(1)

INTEGER IP,INIT,K,MK,STAT,THESVAL,I,IJ,LENGTH

DATA XI513*0.01
DATA AF/513*0.01
DATA AB/512*0.01
DATA CI513*0.01
DATA EF,EPF,EB,PF,PB,GAMMA,OMEGA,SAVE,EPB,TEMP,HOLD/0.01
DATA IP,INIT,K,MK,THESVAL/I,7*0/

WRITE(*,401) 'Enter number of Adaptive iterations'
READ(*,402) THESVAL
WRITE(*,401) ''
WRITE(*,401) 'Enter number of Adaptive Filter Weights'
READ(*,402) IP
WRITE(*,401) ''
LENGTH = IP
```
WRITE(*,401) 'Enter Convergence Rate, OMEGA'
READ(*,403) OMEGA
WRITE(*,401) '  
OMEGA = 1.0

401 FORMAT(IH,A)
402 FORMAT(IH,I10)
403 FORMAT(IH,F10.6)

INIT = 0
DO 1000 I=1,THESVAL
  PERFORM SHIFT FUNCTION
  DO (K=LENGTH+1,2,-1)
    X(K) = X(K-1)
  END DO
  X(1) = X(1) / 10 + 0.5  ! *** ADD D.C. BIAS

*** SEE APPENDIX 9.C
"FAST RLS ALGORITHM AND PROGRAM TO SOLVE EXPONENTIALLY
WINDOWED EQUATIONS OF LINEAR PREDICTION"
Digital Spectral Analysis With Applications - S.L. Marple

... Insert program here and modify to run with REAL data

CALCULATE THE PREDICTOR COEFFICIENTS

AF(0) = 0.0
DO 200 JJ = 0,LENGTH
  WEIGHT(JJ) = -AF(JJ)
  WRITE (*, 11) 'WEIGHT',WEIGHT(JJ)
200 CONTINUE
  !
  FORMAT ( IH, A, F10.6 )

STAT = 1
RETURN
END

SUBROUTINE FGRAD(X,STAT,WEIGHT,LENGTH)
This is the Fast Gradient subroutine,
Many thanks for development go to
G. Gacyna (Mitre Corp.) and P. Moser (AVTEC Systems)
IMPLICIT NONE
INTEGER THESVAL,LENGTH,STAT,ISTOP,I,IP,IP1,IPM1,IPDELTA
REAL X(1),WEIGHT(1),U,YMl,E,R(0:256,2),ES,DELTA(256),
DELTA(S(256),XLABDA,RR(0:256,2),XMU,RRI,XKR(256),
XM(256),ELAST,A(0:256,0:256),B(0:256,0:256),
W(0:256,0:256),XN1,XN2

DATA E,ES,RRI,ELAST, YM1,XN1,XN2/8*0.0/
DATA XMU/1.0E-18/
DATA IDELTAS/10/SAMPLE OFFSET
DATA XLABDA/1.0/
DATA DELTA256*0.0/
DATA RR/S14*0.0/
DATA DELTAS256*0.0/
DATA R514*0.0/
DATA W/66049*0.0/
DATA A/66049*0.0/
DATA B/66049*0.0/
DATA XKS/256*0.0/
DATA XKR/256*0.0/

WRITE(*,401) 'Enter number of Adaptive iterations '
READ(*,402) THESVAL
WRITE(*,401) '! 

WRITE(*,401) 'Enter number of Adaptive Filter Weights '
READ(*,402) LENGTH
WRITE(*,401) ' 

401 FORMAT(1H,A)
402 FORMAT(1H,110)

DO 1000 I=1,THESVAL
  IF(I.LE.IDELTA) THEN
    YT = 0.0
  ELSE
    YT = X(I-IDELTA)
  END IF
  IF(I.GE.2) THEN
    XM1 = X(I-1)
  ELSE
    XM1 = 0.0
  END IF

*BEGIN ALE PROCEDURE*

ESTIMATE DATA USING ADAPTIVE FILTERING

E = YT
R(0,2) = YT
ES = XM1
ISTOP = 1
IF(ISTOP .GT. LENGTH) ISTOP = LENGTH
DO 150 IP = 0, ISTOP - 1
  IPP1 = IP + 1
  IPPM1 = IP - 1
  DELTA(IPP1) = XLAMBDA * DELTA(IPP1) + 2.0 * E * R(IP, 1) +
  DELTAS(IPP1) = XLAMBDA * DELTAS(IPP1) + 2.0 * ES * R(IP, 1)
C
  RR(IP, 2) = XLAMBDA * RR(IP, 1) + E * E + R(IP, 1) * R(IP, 1)
  IF(ABS(RR(IP, 2)) .LE. XMU) THEN
    RRI = 0.0
    ELSE
    RRI = 1.0 / RR(IP, 2)
  END IF
  XKR(IPP1) = DELTA(IPP1) * RRI
  XKS(IPP1) = DELTAS(IPP1) * RRI
  ELAST = E
  E = E - XKR(IPP1) * R(IP, 1)
  ES = ES - XKS(IPP1) * R(IP, 1)
  RR(IP, 2) = R(IP, 1) - XKR(IPP1) * ELAST
  150 CONTINUE

DO (IP = 0, ISTOP)
  R(IP, 1) = R(IP, 2)
  RR(IP, 1) = RR(IP, 2)
END DO
C
WRITE (*, 15) 'INCREMENT = ', I
15 FORMAT ( 1H4, A, 16 )
1000 CONTINUE
C
C CALCULATE THE PREDICTOR COEFFICIENTS
C
DO 200 IJ = 0, LENGTH
  IF (IJ .EQ. 0) THEN
    A(0, IJ) = 1.0
    B(0, IJ) = 1.0
    ELSE
    A(0, IJ) = 0.0
    B(0, IJ) = 0.0
  END IF
  DO (IP = 0, LENGTH - 1)
    IF (IJ .EQ. 0) W(IP, IJ) = 0.0
    A(IP + 1, IJ) = A(IP, IJ) - XKR(IP + 1) * B(IP, IJ - 1)
    B(IP + 1, IJ) = B(IP, IJ - 1) - XKR(IP + 1) * A(IP, IJ)
    W(IP + 1, IJ) = W(IP, IJ) - XKS(IP + 1) * B(IP, IJ - 1)
  END DO
  WEIGHT(IJ) = -W(LENGTH, IJ)
  WRITE (*, 11) 'WEIGHT', WEIGHT(IJ)
200 CONTINUE
11 FORMAT ( 1H4, A, F10.6 )
C
STAT = 1
RETURN
C
END
SUBROUTINE PLOTX(X,N,M,IFILE)

Calculate Power Spectral Density for input & output signal data

INCLUDE 'INCLUDE FILES:EVENT.INC

DIMENSION X(1),OUT(256),WINDOW(256),OUTF(256)
COMPLEX S(512)
INTEGER TIMBEF,TIMAFT,DELTIM,TOOLBX
REAL NAV, SIGAV, DBOUT(256)
DATA WINDOW/256*0.0/

******Initialize Window and Spectrum Accumulator******

IF(WINDOW(128).EQ.0.0) CALL CREATE_WINDOW(256,WINDOW)
DO (I=1,256)
OUT(I) = 0.0
END DO

CALCULATE SPECTRUM
TIMBEF = TOOLBX(TICKCOUNT)
DO 1000 I=1,32768,128
ISTR = I+255
IF(ISTR.GT.32768) GO TO 1000

CREATE COMPLEX SAMPLES
K=0
DO (J=1,ISTR)
K = K+1
S(K) = WINDOW(K)*CMPLX(X(J),0.0)
END DO

ZERO PAD
DO (J=257,512)
S(J) = CMPLX(0.0,0.0)
END DO

CALL FFT(S,512,9,1)
DO (I=1,256)
OUT(I) = OUT(I) + CABS(S(I))
END DO
1000 CONTINUE

TIMAFf = TOOLBX(TICKCOUNT)
DELTIM = TIMAFf - TIMBEF
TIMDIFF = DELTIM / 60.0

SMOOTH SPECTRUM
OUTF(1) = 0.5*OUT(1) + 0.5*OUT(2)
DO (I = 2,255)
OUTF(J) = 0.25*OUT(J-1) + 0.5*OUT(J) + 0.25*OUT(J+1)
END DO
OUTF(256) = 0.5*OUT(255) + 0.5*OUT(256)

DO (J=1,256)
IF(OUTF(J) .LE. 0.0) THEN
  DBOUT(J) = -100.
ELSE
  DBOUT(J) = 20.0*ALOG10(OUTF(J))
END IF
WRITE(IFILE,6010) DBOUT(J)
END DO

C CALCULATE SNR's
NAV = 0.0
SIGAV = 0.0
DO (J = 1,93)
  NAV = NAV + DBOUT(J)
END DO
DO (J = 113,145)
  NAV = NAV + DBOUT(J)
END DO
DO (J = 165,196)
  NAV = NAV + DBOUT(J)
END DO
DO (J = 216,256)
  NAV = NAV + DBOUT(J)
END DO
SIGAV = OUTF(103) + OUTF(155) + OUTF(206)
NAV = NAV/196.0
SIGAV = 20.0 * ALOG10(SIGAV)
WRITE(*,20) 'NOISE AVERAGE POWER (dB) = ',NAV
WRITE(*,20) 'SIGNAL AVERAGE POWER (dB) = ',SIGAV
WRITE(*,20) 'Signal to Noise Ratio (dB) = ',SIGAV-NAV
FORMAT (1H , A, F14.6)
WRITE(*,10) 'PRESS ENTER TO CONTINUE'
READ(*,30) N
WRITE(*,10) '
FORMAT (1H , A)
FORMAT (1H , F2.2)
RETURN
END

SUBROUTINE PI..DTW(W,N)
C PURPOSE:
C THIS SUBROUTINE CALCULATES THE TRANSFER FUNCTION OF THE WEIGHTS
C
C IMPLICIT NONE
REAL W(I), OUT(257)

INTEGER N, N2, M2, I
COMPLEX S(257)

PERFORM FFT TO GET TRANSFER FUNCTION

N2 = 2*N
M2 = N2
CALL PWR2(M2)
DO (I=1,N)
   S(I) = CMPLX(W(I-1), 0.0)
   END DO
DO (I=N+1,N2)
   S(I) = CMPLX(0.0, 0.0)
   END DO
CALL FFT(S, N2, M2, 1)
DO (I=1,N)
   OUT(I) = CABS(S(I))
   IF(OUT(I) .LT. 0.0) THEN
     OUT(I) = -100.
   ELSE
     OUT(I) = 20.0*ALOG10(OUT(I))
   END IF
   WRITE(4,6010) I-1, OUT(I)
   WRITE(4,6010) OUT(I)
   END DO

SUBROUTINE CREATE_WINDOW(N, WINDOW)

POUNSE:
THIS SUBROUTINE CREATES HANNING WINDOW WEIGHTS.

DIMENSION WINDOW(256)
DATA TWOPI/6.283185/

DO = N-1
DO (I=1,N)
   WINDOW(I) = 0.5 - 0.5*COS(TWOPI*I/PNENTS)
   CONTINUE
RETURN
END
COOLEY'S SIMPLE FFT PROGRAM-USES DECIMATION IN TIME ALGORITHM.
TAKEN FROM IEEE PRESS "PROGRAMS FOR DIGITAL SIGNAL PROCESSING",
PAGE 2.1-10.

PARAMETERS:
X IS AN N==2**M POINT COMPLEX ARRAY THAT INITIALLY CONTAINS THE
INPUT DATA AND CONTAINS THE TRANSFORM ON COMPLETION
N IS THE SIZE OF THE ARRAY = 2**M
M POWER OF TWO SIZE OF THE FFT
INV =1 FOR FFT, 2 FOR IFFT

---------------------------------------------------------------------

COMPLEX X(1),U,W,T,CN
DATA PI*3.141593/

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

SIGN = (-1)**INV
NV2 = N/2
NM1 = N-1
J = 1
DO 40 J=1,NM1
   IF(I .GE. J) GO TO 10
   T = X(I)
   X(I) = X(J)
   X(J) = T
   K=NV2
   IF(K .GE. J) GO TO 30
   J = J-K
   K = K/2
   GO TO 20
  10  J = J+K
  20  CONTINUE
DO 70 L=1,M
   LE = 2**L
   LE1 = LE/2
   U = CMPLX(1.0,0.0)
   W = CMPLX(COS(PI/LE1),SIGN*SIN(PI/LE1))
   DO 60 J=1,LE1
      DO 50 I=J,N,LE
         IP = I+LE1
         T = X(IP)*U
         X(IP) = X(I) - T
         X(I) = X(I) + T
      CONTINUE
      U = U*W
  50  CONTINUE
  60  CONTINUE
IF(INV .EQ. 1) RETURN
FN = N
DO (I=1,N)
   X(I) = X(I)/FN
END DO
RETURN
END

---------------------------------------------------------------------

C
C
C
C
SUBROUTINE PWR2(N)
C
C PURPOSE:
C FIND THE POWER OF TWO CLOSEST TO N.
C
INTEGER TESTVAR, IPWT

*********************************************************
 XI = N
M = ALOG10(XI)/ALOG10(2.0)
TESTVAR = INT(M)
IF (2**M .NE. 2**TESTVAR) TESTVAR = TESTVAR + 1
N = TESTVAR
TESTVAR = 2**M
IF(TESTVAR .LT. N) THEN
M = M+1
GO TO 1
ELSE IF(TESTVAR .GT. N) THEN
M = M-1
GO TO 1
ELSE
IPWT = M
END IF
N = IPWT
RETURN
END

SUBROUTINE RANDOM(Z, RND, X, A, M)
C
SUBROUTINE FOR GENERATING RANDOM NUMBERS HAVING A UNIFORM DISTRIBUTION, BY THE MIXED MULTIPLICATIVE CONGRUENTIAL METHOD.
C
INTEGER A, X
IF (RND .EQ. 0) GO TO 1
RND = 0
M = 2**20
X = 566387
A = 2**10 + 3
C 1
X = MOD(A*X, M)
FX = X
FM = M
Z = FX/FM
RETURN
END

SUBROUTINE GAUSS(XNORM1, XNORM2, RNDSTART, RNDX, RNDN, RNDM)
C
C PURPOSE:
C THIS FUNCTION GENERATES A GAUSSIAN DISTRIBUTION USING A BOX-JENKINS APPROACH.
C

C DATA TWOPI/6.283185/

CALL RAND(Q(PNT1,RNDSTART,RNDX,RNDA,RNDM)
   IF(PNT1 .LE. 0.0) GO TO 10
20 CALL RAND(Q(PNT2,RNDSTART,RNDX,RNDA,RNDM)
   IF(PNT2 .LE. 0.0) GO TO 20
XNORM1 = (-2. * ALOG(PNT1))
A = SQRT(XNORM1)
XNORM1 = A * COS(TWOPI * PNT2)
   XNORM2 = A * SIN(TWOPI * PNT2)
C RETURN
END

SUBROUTINE QUIT(S
   JUNK = 1
   WRITE(*,111) ' *** MALFUNCTION ***'
111 FORMAT (1H,A)
   END
C END
Appendix B

ALE Simulation Output Data
Data From 3k obs. 16tap LMS

Data From 3k obs. 32tap LMS

Data From 3k obs. 64tap LMS

Data From 3k obs. 128tap LMS
Data From 8k obs. 16tap LMS

Data From 8k obs. 32tap LMS

Data From 8k obs. 64tap LMS

Data From 8k obs. 128tap LMS
The vita has been removed from the scanned document