A General Order Multichannel, Fast Least Squares Algorithm with Telecommunications Applications

James Faber

Center for Communications and Signal Processing
Electrical and Computer Engineering Department
North Carolina State University

CCSP-TR-88/29

October, 1988
Abstract

FABER, L. JAMES  A General Order Multichannel, Fast Least Squares Algorithm with Telecommunications Applications. (Under the direction of Sasan H. Ardalan and Sarah A. Rajala.)

A new, general order, multichannel transversal fast recursive least squares algorithm is derived, using the principles of geometric projection. It is a generalization of prior multichannel fast LS algorithms, in that the orders of the several input channel joint process filters can be independently and arbitrarily specified. Like other fast LS algorithms, it has the property that update computations are of the same order as the total filter size, N. The algorithm is applied to a special two-channel case in which the unit-delayed joint process signal is taken as one of the input channels; this configuration is the equation-error form for LS pole-zero joint process estimation. A new and significant (2N operations per recursion) computational savings is derived for this configuration. Identification of an IIR bandpass filter is demonstrated for several conditions of over- and underspecification of filter orders. Several extensions to the algorithm are developed, including exponential memory, complex data, and normalized variables. The adaptive algorithm is also applied to adaptive equalization and echo cancellation, using a general purpose digital system simulator. The system parameters are chosen to model full-duplex subscriber loop data communications (ISDN). A new approach to pole-zero echo cancellation is described and tested.
# TABLE OF CONTENTS

1. INTRODUCTION ........................................................................................................ 1
   1.1 Adaptive Filtering ............................................................................................. 1
   1.2 Linear Prediction and System Identification ................................................. 4
   1.3 Outline of Dissertation ................................................................................... 9

2. ADAPTIVE ALGORITHMS ....................................................................................... 14
   2.1 Least Mean Squares ....................................................................................... 14
   2.2 Least Squares .................................................................................................. 16

3. THE GENERAL ORDER, MULTICHANNEL FTF .................................................. 23
   3.1 Introduction ................................................................................................... 23
   3.2 Generalized Updates of Operators .................................................................. 33
   3.3 Derivation of the Fast Multichannel Algorithm ........................................... 36
   3.4 The Higher Order Multichannel Case ............................................................ 42
   3.4 Appendix: Operators on Augmented Subspaces ............................................ 45

4. THE FAST ARMA PREDICTOR .......................................................................... 49
   4.1 Introduction ................................................................................................... 49
   4.2 ARMA System Identification ........................................................................ 53
   4.3 Additional Observations ............................................................................... 60
   4.4 Filter Modeling Simulations .......................................................................... 63

5. ALGORITHM EXTENSIONS .................................................................................. 72
   5.1 Exponential Error Weighting ......................................................................... 72
   5.2 Multichannel Output ...................................................................................... 77
   5.3 Complex Data .................................................................................................. 79
   5.4 The Normalized General Order Algorithm ................................................... 80

6. APPLICATION SIMULATIONS .............................................................................. 88
   6.1 Introduction ................................................................................................... 88
   6.2 The Transmitter ............................................................................................. 91
   6.3 The Channel ................................................................................................... 97
   6.4 The Receiver .................................................................................................. 100
   6.5 Predictors ....................................................................................................... 108
   6.6 The Experiments ............................................................................................ 113

7. CONCLUSION ......................................................................................................... 136

8. REFERENCES .......................................................................................................... 139

9. APPENDIX: Simulation Modules and Topologies ............................................. 143
CHAPTER 1
INTRODUCTION

Adaptive filtering is a fairly recent offshoot of the mature field of linear filtering theory [1], but has already proven to be extremely useful in many signal processing applications. In this chapter, a brief overview of adaptive filtering, and some of the major application areas is given. Via the concepts of linear prediction and system identification, a unifying categorization of these application areas is described. Finally, a general preview of the topics considered in the remainder of this work is presented.

1.1. Adaptive Filtering

Filter design has always been an integral part of electrical engineering. Originally operating on continuous analog signals, and constructed of physical components, filters were used to separate, suppress or enhance definite regions of the electro-magnetic spectrum. Filters were, and in many applications still are, designed in terms of their attenuation or phase characteristics versus frequency, in such basic filter varieties as "low-pass", "band-stop", etc. However, following theories of optimal filtering developed by Wiener, Kalman and others [2]-[6], filter specification was expanded to include the concept of optimal signal prediction. In the time domain, signal prediction means that the filter output can accurately estimate the future value of its input, or some other (joint process) signal. Fixed pred-
Adaptive filters, however, can modify their parameters during operation to cope with unknown or changing conditions, and usually require little *a priori* signal information.

Adaptive filtering has been applied to a wide range of signal processing applications. Several representative examples are: speech analysis and encoding [7][8], spectral estimation [9], antenna beam forming [10], noise cancellation [11][12], echo cancellation [13]-[15], and voice- and data-channel equalization [16]-[19]. The application topics considered here will be primarily data-channel echo cancellation and equalization.

In this work, we will be concerned with sampled, rather than continuous data systems. This is the modern domain of computer implemented signal processing algorithms. Filter operation is determined by the operation of a computer program; filter components exist as abstract multipliers, which are no longer constrained to be fixed in value. Many linear filter configurations are available, each with certain advantages in various applications. However, we will deal primarily with the transversal, or finite impulse response (FIR) configuration. This form is illustrated in Figure 1.1. The filter consists of a cascade of unit-delay shift registers, with independently specified multiplying coefficients for each delayed input sample. The weighted samples are then summed to produce the filter output, which is often considered to be a prediction (estimate) of another, "joint process"
Figure 1.1 Transversal Filter

signal. An adaptive algorithm has the capability to modify the weighting coefficients, according to various rules, in order to produce a best prediction. These rules and a definition of "best" are discussed later.

The filter functions used here for sampled data signals are described mathematically by fixed-order linear difference equations with constant coefficients. That is, the relationship between an input sequence \( \{x(n)\} \) and an output sequence \( \{d(n)\} \) for a filter system can be written

\[
d(n) = \sum_{i=0}^{L-1} b_i x(n-i) + \sum_{i=1}^{M} a_i d(n-i),
\]

with finite \( L \) and \( M \). Here, \( N = L + M \) is considered the filter "order". In many cases, only the finite impulse response (FIR) form will be considered; this implies that all \( a_i \) equal zero. In the case of adaptive filters, the coefficients are variable,
and dynamically adjusted by various schemes, considered more fully in the next chapter.

1.2. Linear Prediction and System Identification

The diverse range of signal processing applications for adaptive filtering can be divided into two primary categories: linear prediction (LP) and system identification (SI) [23]. Linear prediction refers to prediction of the next sample of an unknown stationary or non-stationary signal, based on finite number of previous signal samples only. Linear prediction derives an estimate of the incoming signal without knowledge of the source excitation. LP applications most often concern signal separation, which is the extraction or elimination of certain components or redundancies of a signal. Consider the system illustrated in Figure 1.2, where the input signal $x(n)$ consists of the sum of two components, "desired signal" and "noise". Here noise can be any unwanted component, broadband or narrowband, correlated with the signal or not. The coefficients of the adaptive filter $W(z)$ are adjusted in order to predict a portion of the incoming signal. Obviously, this portion must be predictable (i.e., fit the assumed model), although it may be either the desired signal or the noise. The predicted component is then subtracted from the composite signal, leaving the unpredictable portion. This error signal, which may or may not be the ultimately desired signal, is used to modify the filter parameters, normally so that the power of the error signal is minimized. This implies that no further predictability exists, at least within the constraints of a particular signal
System identification is used to estimate the coefficients of an assumed model for an unknown fixed (stationary) system, or to track the parameters of a system that is time-varying in an unknown way. SI and LP are closely related since the prediction inherent in the latter requires (explicitly or not) the formulation of a linear signal-source model. However, SI normally requires access to the excitation signal† for the unknown process, as illustrated in Figure 1.3a. A useful equivalent system is shown in Figure 1.3b, which models both the system and SI filters with standard transversal forms, as in Figure 1.1. If the parameters of the identification

† Sometimes it is possible to create an estimate of the input signal, and achieve satisfactory results [23][28][29].
Figure 1.3 System Identification: a) General  
  b) Transversal Equivalent
filters match those of the unknown system, the error signal will be zero. It is assumed that model orders are known or bounded. Figure 1.4a shows a special all-zero case of SI which is often used. The estimate of the unknown system output is created by filtering only the input signal. Figure 1.4b extends this concept to multiple input channels, which is one application of the algorithms developed in this work.

Figure 1.5 shows another common configuration representing all-pole linear prediction. It is shown in [23] that LP can be considered a special case of system identification. This assertion is supported by comparing Figure 1.5 to 1.3b. Here only the system output signal is filtered; there is no access to the driving signal \( x(n) \) in this configuration.

In this work, we will consider two primary application areas for these various transversal filtering configurations: channel equalization and echo cancellation. These are depicted in Figure 1.6, and have been drawn to match the general system identification form, but with appropriate signal names. Equalization is usually considered an inverse filtering problem. However, with a simple linear network transformation (see figures 1.6a and 1.3a), equalization can be considered a system identification problem, where the unknown system to be identified, \( U(z) \), is the inverse of the data transmission channel. This will be considered further in the applications chapter.
Figure 1.4 All-Zero System Identification a) Single Channel  b) Multichannel
There have recently been discovered several fast, recursive least squares (LS) algorithms for the determination of the coefficients of transversal adaptive filters [22][31][32]. They are called "fast" since they all have a computational requirement per update that is a linear function of the filter size. Of these algorithms, the Fast Transversal Filter (FTF)[32] requires the fewest computations. The primary focus of this dissertation is the extension of the FTF algorithm to a new multichannel form. Although a multichannel version of this algorithm was previously developed in [32], it was constrained to have equal orders in all input channels. This work develops a general order multichannel extension of the algorithm, which is completely arbitrary in the orders of each of the input channel sub-filters. This
each of the input channel sub-filters. This required a conceptual modification of the derivation method. This development is shown in Chapter 3, and considers the configuration in Figure 1.4b.

In Chapter 4, the new algorithm is applied to a specific two-input configuration, known in system identification literature as the "equation error" formulation, as is illustrated in Figure 1.3b. This formulation allows a type of pole/zero (autoregressive, moving average, or ARMA) estimation of system parameters. The general order capability was not previously known using the multichannel FTF algorithm in this application. A significant \((2N+1)\) computational complexity reduction is then described and proven for this configuration, which previously required \(O(12N)\) computations per update. This reduction was shown to be available with the other known fast recursive algorithms (FK and FAEST) as well, which had not been noted in earlier work.

Chapter 5 addresses several additional topics:

1) algorithm extension to include "exponential data memory". This mechanism for discounting older data is important in many practical applications, where the filter must continuously adapt to slowly changing conditions.

2) consideration of the multichannel output case, and the complex data case. These are shown to be related, and straightforward algorithm extensions.

3) development of a normalized general order algorithm. The input linear prediction kernel is recursively normalized, which helps constrain the variance of
Two-Channel Equalization

Two-Channel Echo Cancellation

Figure 1.6 Transversal System Identification Applications
algorithm variables. Such normalized forms have been shown to provide superior numerical stability properties when compared to unnormalized ones; this is particularly important in practical implementations with limited-precision machines. However, the number of computations is increased, and it is necessary to take square roots.

Chapter 6 summarizes several application simulations of the algorithm for the Integrated Services Digital Network (ISDN) environment. This 192 kBps data-transmission network specification is still under development; it is intended to deliver a direct high-speed digital interface to subscribers on standard unconditioned 2-wire telephone lines [49][50]. This has created the need for adaptive line equalization and adaptive transmit (near-end) echo cancellation. The capacity of LS algorithms for rapid convergence make them attractive candidates for these requirements. The flexible pole/zero modeling capability can also produce tremendous reduction of the model order (and consequently, computational burden) required, compared to the current all-zero models. In a real-time situation, the general order capability allows allotment of limited computational time to particular channels; it also provides independent specification of pole and zero orders in the two-input equation error (ARMA) configuration described earlier.

The BLOSIM (Block Simulation) [51] system simulator program was used to build an end-to-end data transmission system, complete with adaptive equalizer and echo canceller. This program allowed verification of algorithm operation and
utility. An additional benefit of the simulation work was the generation of a large library of software modules available for future efforts in data transmission simulations.
CHAPTER 2

ADAPTIVE ALGORITHMS

Adaptive filters require some mechanism to adjust their parameters so as to generate the "best" estimate of the joint process signal. Appropriate coefficients for the adaptive transversal form are normally computed by algorithms that attempt to minimize estimation error power, since this can be considered the measure of a good estimate [11]. Minimization of error methods in general can be divided into two classes, stochastic and exact. Stochastic methods attempts to minimize the statistical average (or mean) squared error signal. These methods assume short-term stationarity of the input signal, and are formulated in terms of the gradient of the error power, with respect to the filter parameters. The least mean squares (LMS) algorithm [11][21] is a well known example of this class. Exact or least-squares (LS) algorithms exactly minimize the running sum of squared error values rather than the average error power.

2.1. Least Mean Squares

The stochastic autocorrelation matrix of a signal was shown by Wiener [4][21] to generate the minimum mean square solution to the optimal estimation (LP) problem. If it is assumed that the incoming signals are stationary and that the covariance statistics are known, then the optimal \((N \times 1)\) transversal filter vector for \(\{x(n)\}\) to predict \(\{d(n)\}\) is given as
where $\Phi_{zz}$ is the $(N \times N)$ autocorrelation matrix, and $\phi_{zd}$ is the $(N \times 1)$ cross-correlation vector. Since the autocorrelation matrix is Toeplitz (equal elements along major diagonals) and symmetric, direct inversion, $O(N^3)$, is not required to implement (2.1). Instead, the order-recursive Levinson/Durbin algorithm [9], which is $O(N^2)$, can be used. In practice, the true correlation statistics are usually unknown. They must be estimated from the incoming data, which requires accumulation of several data samples before computations can be performed. Thus, although theoretically optimal, this method performs poorly in practical parameter estimation, particularly if the data record is short [9].

The best known time-recursive (sample-by-sample update) algorithm based on statistical estimation is the LMS (least mean squares) algorithm. It is actually iterative, rather than truly recursive, in the sense that it never achieves the optimal Wiener solution mentioned above. It converges toward an assumed optimal solution (based on stationary signal statistics) by using the instantaneous error as an approximation of the gradient of the squared error function; small adjustments are made to each weight value scaled by the data sample associated with each weight. The LMS algorithm sequence is given as:

\begin{align}
(a) \quad e(n|n-1) &= d(n) - \hat{d}(n|n-1) = d(n) - x_N^T(n)w_N(n-1) \\
(b) \quad w_N(n) &= w_N(n-1) + \gamma x_N(n)e(n|n-1). \tag{2.2}
\end{align}

Here $x_N(n)$ defines the length $N$ vector of input data at time $n$,
\[ \mathbf{x}_N(n) = [x(n), \cdots, x(n-N+1)]^T \]

and \( \hat{d}(n|n-1) \) is an estimate of \( d(n) \), generated by the transversal filter of time \( n-1 \), defined as the vector

\[ \mathbf{w}_N(n-1) = [w_0(n-1), \cdots, w_{N-1}(n-1)]^T \]

The notation \( e(n|n-1) \) means the prediction error at time \( n \) due to the prediction from the filter of time \( n-1 \). The small positive constant \( \gamma \) is chosen based upon input signal power and the filter size [11]. It can be observed that this algorithm requires \( \sim 2N \) computations per update.

It is known that the LMS gradient-estimate algorithm has convergence properties that are related to the statistics of the input signal. In particular, convergence can be distinguished into convergence modes, each related to a particular eigenvalue of the input signal. This is often undesirable since extremely slow filter convergence can result if there is large difference between the power in signal modes [11]. In many applications, the LS algorithms described next have far superior convergence properties [20][25][30][39]. Convergence of LS algorithms does not depend on signal statistics, but is determined by the filter size.

### 2.2. Least Squares

The other basic approach to parameter determination requires finding the exact least squares solution, \( i.e., \) the filter that would provide the smallest sum of squared errors if applied to all previous data. In vector form, the error based on the optimal filter can be expressed
\[
e(n \mid n) = d(n) - X(n)w_N(n) \tag{2.3}
\]

where

\[
e(n \mid n) = [e(1 \mid n), e(2 \mid n), \cdots, e(n \mid n)]^T,
\]
\[
d(n) = [d(1), d(2), \cdots, d(n)]^T
\]

are accumulations of scalar values defined earlier, and

\[
X(n) = \begin{bmatrix}
x(1) & 0 & \cdots & 0 \\
x(2) & x(1) & 0 & \cdots \\
\vdots & \ddots & \ddots & \ddots \\
x(n) & x(n-1) & \cdots & x(n-N+1)
\end{bmatrix}
\]

Thus, the sum of squared errors equals the error vector norm squared:

\[
\text{minimum} \sum_{i=1}^{n} e^2(i \mid n) = \min_{w.r.t. w_N} e^T(n \mid n)e(n \mid n). \tag{2.4}
\]

Given the independence of the columns of \(X(n)\), a well known result of linear algebra is that for an overdetermined set of equations (here meaning more data samples than adjustable weight parameters), a LS solution for the filter parameters exists and is unique [46]. This solution can be found by vector differentiation, or by using orthogonality principles for linear systems of equations. Without showing details, the optimal length \(N\) filter for all data up to time \(n\) can be expressed in matrix form as:

\[
w_N^*(n) = [X^T(n)X(n)]^{-1}X^T(n)d(n). \tag{2.5}
\]

Naturally, this direct approach is totally impractical for real-time implementation, since the \(X\) matrix continuously grows with time. If the filter vector were to be computed at each time increment, \(O(n^3)\) operations per update would be
Recursive forms are often sought in these situations; such forms use the solution found at the previous data sample to help generate the solution including the new sample. This can provide a substantial economy in computations.

2.2.1. Recursive Least Squares. A fundamental observation is that the $(N \times N)$ data matrix product $X^T X$ grows in a systematic way. This matrix product, called the sample autocorrelation matrix, $R_{XX}(n)$, is based on only a finite number of previous data samples, and is a statistical estimator of the true signal covariance matrix $\Phi_{zz}$. Hence, a relationship to the stochastic approaches can be seen. It is easy to show that $R_{XX}(n)$ can be decomposed as follows:

$$R_{XX}(n) = X^T(n)\Lambda(n)X(n) = \lambda R_{XX}(n-1) + x_N(n)x_N^T(n), \quad (2.6)$$

where $\Lambda(n) = \text{diag}[\lambda^{n-1}, \ldots, \lambda, 1]$, and $0 < \lambda \leq 1$ is now introduced as an exponential weighting factor. This factor is used for signals with non-stationary statistics in order to discount earlier data. Using this recursive form, the direct approach of (2.4) can be achieved in $O(N^3)$ computations, since matrix inversion is required.

The recursive least squares (RLS) algorithm was obtained as a special case of Kalman filtering theory [9]. It was derived using the recursive form of (2.6) and the matrix inversion lemma [52], which generates an explicit inverse formula for matrices of a certain type. The algorithm sequence is as follows:
\( (a) \quad e(n | n-1) = d(n) - w_N^T(n-1)x_N(n) \)
\( (b) \quad k_N(n) = [\lambda + x_N^T(n)Q(n-1)x_N(n)]^{-1}Q(n-1)x_N(n) \)
\( (c) \quad w_N(n) = w_N(n-1) + k_N(n)e(n | n-1) \)
\( (d) \quad Q(n) = \lambda^{-1}[Q(n-1) - k_N(n)x_N^T(n)Q(n-1)] \)

The matrix \( Q(n) = R_{XX}^{-1}(n) \) is propagated directly in the algorithm, and thus no matrix inversion is required. The \((N \times 1)\) vector \( k_N \) is often called the Kalman gain vector. Note the similarity of (2.7a,c) to the LMS algorithm (2.2). The Kalman gain vector is approximated there by \( \gamma x_N(n) \), a scaled input vector.

The RLS algorithm makes recursive computation of the LS filter solution possible at each time increment. It requires \( O(N^2) \) computations per update, where \( N \) is the length of the transversal prediction filter. However, this is still infeasible for present real-time signal processors.

2.2.2. Fast RLS. A substantial reduction in RLS computational complexity was achieved with the Fast Kalman (FK) algorithm of Morph, Ljung and Falconer [22][25]. This algorithm made real-time LS processing practical by requiring only \( \sim 10N \) computations per update. This is very competitive with the relatively primitive LMS. The algorithm was derived by exploiting the shifted column property of the data matrix \( X(n) \). The Kalman gain vector is computed via auxiliary forward and backward prediction filters for the input signal, which are also updated recursively.

\[\dagger\] These filters appear in all the fast algorithms, and are denoted \( f_N(n) \) and \( b_N(n) \) in this work.
This landmark was surpassed by two recently developed $\sim 7N$ recursive LS algorithms, the fast transversal filter (FTF) [32] and the fast a posteriori error sequential technique (FAEST) [31]. These new filter algorithms were designed to update the transversal filter form and are recursive, i.e., the algorithms move to a new LS optimal solution from the old solution, as each new data point becomes available. Recent work by Wang [12] has shown the mathematical relationship between the FTF and FAEST algorithm solutions to the LS problem, each more efficient than the Fast Kalman algorithm. The single channel (scalar) FTF algorithm is shown in Table 2.1.

Notably, the FTF algorithm was derived in [32] using a vector space or geometric projection approach. This approach has several advantages in understandability and generality, and was adopted for this work. In the vector space method, the span of matrix $X(n)$ is defined as a vector subspace. Then the LS estimate of the desired data vector obtainable by linear filtering of the input signal consists of the projection of the desired signal vector onto that subspace. The error vector (unpredictable component) is orthogonal to the estimate vector. This is illustrated conceptually in Figure 2.1, for a 2-dimensional (i.e. two columns) $X$ subspace. As data is added the subspace orientation, but not dimensionality, changes. However, a relatively simple update to the operator that projects onto the subspace proves to be possible; this generalized projection operator update then generates recursions for all necessary algorithm quantities. These concepts will be
Table 2.1
The Single Channel (Scalar) FTF Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1×1</td>
<td>N</td>
<td>( e_f(n</td>
</tr>
<tr>
<td>2</td>
<td>1×1</td>
<td>1</td>
<td>( e_f(n</td>
</tr>
<tr>
<td>3</td>
<td>1×1</td>
<td>1</td>
<td>( e_f(n) = e_f(n-1) + e_f(n</td>
</tr>
<tr>
<td>4</td>
<td>1×1</td>
<td>2</td>
<td>( \gamma_+(n) = \gamma(n-1)e_f(n-1)e_f^{-1}(n) )</td>
</tr>
<tr>
<td>5a</td>
<td>N×1</td>
<td>N+1</td>
<td>( c_{N+1}^f(n) = \begin{bmatrix} 0 \ e_N(n-1) \end{bmatrix} + \begin{bmatrix} 1 \ -f_N(n-1) \end{bmatrix} \frac{e_f(n</td>
</tr>
<tr>
<td>5b</td>
<td>1×1</td>
<td>0</td>
<td>( c_+(n) = \text{last element in } c_{N+1}(n) \text{ vector} )</td>
</tr>
<tr>
<td>6</td>
<td>N×1</td>
<td>N</td>
<td>( f_N(n) = f_N(n-1) + c_N(n-1)e_f(n</td>
</tr>
<tr>
<td>7</td>
<td>1×1</td>
<td>1</td>
<td>( e_b(n</td>
</tr>
<tr>
<td>8</td>
<td>1×1</td>
<td>3</td>
<td>( \gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b(n</td>
</tr>
<tr>
<td>9</td>
<td>1×1</td>
<td>1</td>
<td>( e_b(n</td>
</tr>
<tr>
<td>10</td>
<td>1×1</td>
<td>1</td>
<td>( e_b(n) = e_b(n-1) + e_b(n</td>
</tr>
<tr>
<td>11</td>
<td>N×1</td>
<td>N</td>
<td>( \begin{bmatrix} c_N(n) \ 0 \end{bmatrix} = c_{N+1}(n) + \begin{bmatrix} b_N(n-1) \ -1 \end{bmatrix} c_+(n) )</td>
</tr>
<tr>
<td>12</td>
<td>N×1</td>
<td>N</td>
<td>( b_N(n) = b_N(n-1) + c_N(n)e_b(n</td>
</tr>
<tr>
<td>13</td>
<td>1×1</td>
<td>N</td>
<td>( e(n</td>
</tr>
<tr>
<td>14</td>
<td>1×1</td>
<td>1</td>
<td>( e(n</td>
</tr>
<tr>
<td>15</td>
<td>N×1</td>
<td>N</td>
<td>( w_N(n) = w_N(n-1) + c_N(n)e(n</td>
</tr>
<tr>
<td>Total OPS</td>
<td>7N + 12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 2.1  Orthogonal Projection Onto a Subspace

developed more fully in the following chapters.
CHAPTER 3
THE GENERAL ORDER, MULTICHANNEL FTF‡

3.1. Introduction

Fast transversal recursive least squares (RLS) algorithms [31],[32] have filter update computational costs of order $N$, the filter length. Although these algorithms suffer from long-term instability problems, their convergence superiority in many applications, compared to the common LMS (gradient) algorithms, have made them an active area of research. The purpose of this chapter is to present the derivation of a multichannel fast transversal recursive least squares algorithm with general order inputs. The approach in this derivation is to use the method of geometric projections onto a vector space defined by the input data. The application of this technique from linear algebra to the transversal filter case was first given by Cioffi and Kailath [32]. The algorithm in this paper is distinguished from the multichannel algorithm shown there in which the order of the filter for each channel was constrained to be equal; here the order of each channel is completely arbitrary. This property, which allows greater flexibility in some applications, causes certain difficulties in the derivation. The key to resolving these difficulties is the use of permutation matrices, similar to those used by Falconer and Ljung [22] in their application of the fast Kalman algorithm to adaptive equalization. The

‡ This chapter appeared, with certain modifications, in [34].
derivation in this chapter is a generalization of the work by Ardalan [33], which described an application to a particular two channel system in which the (unit delayed) output signal is used as one input channel. This is called the ARMA or pole/zero estimation case, using the equation-error form for system identification [35]. The notation used herein is similar to that of [45].

3.1.1. Problem Description. The goal of the analysis is to recursively determine the length $N$ transversal filter $w_N(n)$, in the system shown in Figure 3.1, such that the optimal estimate of the (joint) process $\{d(n)\}$ is formed by a transversal filtering of the two input processes $\{x(n)\}$ and $\{y(n)\}$. These processes are real data sequences where data is assumed zero for $n \leq 0$; this is called the

![Figure 3.1 Two Channel System Block Diagram](image-url)
prewindowed case. (Note: only two input channels are considered for notational simplicity; the extension to the arbitrary $p$-channel case will be considered in section 3.4.) Here optimality is defined in a cumulative or exact least squares sense; this can be explained as follows. The estimation error incurred at time $i$, based upon the filter at time $n$ is defined

$$e(i|n) = d(i) - \hat{d}(i|n) = d(i) - w_N^T(n)z_N(i), \quad 1 \leq i \leq n,$$

where $\hat{d}(i|n)$ represents an estimate of $d(i)$ using the transversal filter of time $n$, and $z_N(i)$ is a length $N$ vector of past input samples from both channels:

$$z_N(i) = [x(i), x(i-1), \ldots, x(i-L+1)]^T,$$

where

$$w_N(n) = [x_L(i), y_M(i)]^T, \quad \text{with } N = L + M,$$

$$x_L(i) = [x(i), x(i-1), \ldots, x(i-L+1)]^T,$$

$$y_M(i) = [y(i), y(i-1), \ldots, y(i-M+1)]^T.$$

The length $N$ transversal filter $w_N(n)$ is partitioned into subfilters of lengths $L$ and $M$, corresponding to the respective input channels. Thus it is desired to find the

$$\min_{w.r.t. \ w_N(n)} \sum_{i=1}^{n} e^2(i|n).$$

This is the well known least squares (LS) problem, which can be succinctly expressed in vector notation.

3.1.2. Data Vectors and Matrices. The basic concept in the method of geometric projections is to locate vector estimates within a subspace which is spanned by a set of vectors formed from the input data. Vectors are formed of all samples of the input channels from time 1 to time $n$, where $n$ is the present or
latest time:

\[(n \times 1) \quad \mathbf{x}(n) = [x(1), x(2), \ldots, x(n)]^T, \quad \mathbf{y}(n) = [y(1), y(2), \ldots, y(n)]^T \quad (3.4)\]

A similar vector is defined with samples of the joint process \(\{d(n)\}\). It is also convenient to define a 'pinning vector' \([41][42][32]\

\[(n \times 1) \quad \pi(n) = [0, \cdots, 0, 1]^T. \quad (3.5)\]

This is the unit vector in the direction of the newest data sample, and will prove useful later.

A vector \(\mathbf{z}(n)\) is now defined with columns comprised of the input vectors. Additionally, \(\mathbf{z}_b(n)\) is defined with columns of input vectors that are each time delayed by the order of the filter assigned to that channel:

\[(n \times 2) \quad \mathbf{z}(n) = [\mathbf{x}(n), \mathbf{y}(n)], \quad \mathbf{z}_b(n) = [z^{-L}\mathbf{x}(n), z^{-M}\mathbf{y}(n)]. \quad (3.6)\]

Here \(z^{-L}\) for example, is the \(L\)-unit time delay operator. Note that the vector \(\mathbf{z}_b(n)\) is still of length \(n\), but with leading elements of zero since the data is prewindowed. It is also convenient to define the final rows of these vectors as

\[(2 \times 1) \quad \mathbf{z}(n) = [x(n), y(n)]^T = \mathbf{z}^T(n)\pi(n), \quad \mathbf{z}_b(n) = [x(n-L), y(n-M)]^T. \quad (3.7)\]

Note that these rows can be generated by inner products with \(\pi(n)\). If an \((n \times N)\) matrix is now defined by the following relationships
\[ Z_0(n) = [X_0(n), Y_0(n)] \]
\[ = [x(n), z^{-1}x(n), \ldots, z^{-L}x(n), y(n), \ldots, z^{-M}y(n)] \]
\[ = \begin{bmatrix} x(1) & 0 & \cdots & 0 & | & y(1) & 0 & \cdots & 0 \\ x(2) & x(1) & & | & y(2) & y(1) & \cdots \\ \vdots & \vdots & x(1) & | & \vdots & \vdots & y(1) \\ x(n) & x(n-1) & \cdots & x(n-L+1) & | & y(n) & y(n-1) & \cdots & y(n-M+1) \end{bmatrix} \quad (3.8) \]

then a vector representing the estimation error resulting from the transversal filter \( w_N(n) \) can be defined

\[ (n \times 1) \quad e(n|n) = \hat{d}(n) - d(n|n) = d(n) - Z_0(n) w_N(n), \quad (3.9) \]

where each element in \( e(n|n) \) corresponds to a scalar error from (3.1). A row of \( Z_0(n) \) would represent the data in the transversal filter \( w_N(n) \) at a particular time; row \( i \) of \( Z_0(n) \) is then \( z^T_N(i) \) defined in (3.2).

Other useful matrices can be defined. If each column of \( Z_0(n) \) is unit delayed:

\[ (n \times N) \quad Z_1(n) = [X_1(n), Y_1(n)] = [z^{-1}X_0(n), z^{-1}Y_0(n)] \]
\[ = [z^{-1}x(n), \ldots, z^{-L}x(n), z^{-1}y(n), \ldots, z^{-M}y(n)]. \quad (3.10) \]

Consider the \( Z_0(n) \) matrix augmented with one additional column for each of the input channels, representing the addition of a tap for each channel subfilter:

\[ (n \times N+2) \quad Z_+(n) = [X_0(n), z^{-L}x(n), Y_0(n), z^{-M}y(n)] \]
\[ = [x(n), X_1(n), y(n), Y_1(n)]. \quad (3.11) \]

Note that this augmented matrix can be described in two ways, using the previously defined matrices. The final row of \( Z_+(n) \) defines a vector, which can also be decomposed in two ways:
\[(N+2 \times 1) \quad z_{N+2}(n) = [x_L^T(n), x(n-L), y_M^T(n), y(n-M)]^T
= [x(n), x_L^T(n-1), y(n), y_M^T(n-1)]^T.\]  

(3.12)

3.1.3. Projection Operators. From (3.9), each element (tap) of \(w_N(n)\) defines a weight applied to a particular column of \(Z_0(n)\); i.e., the estimate vector \(\hat{d}(n \mid n)\) is a linear combination of the columns. Equivalently, the estimate vector must lie in the column space of \(Z_0(n)\). The columns span a subspace of \(R^n\), the space of all real-valued, length \(n\) vectors. These key observations lead to the concept of projection onto a subspace as a means of locating the least squares error estimate of a vector within that subspace [43]. It is a well known result of linear algebra that the minimum length (norm) error vector corresponds to that estimate of a vector found by taking its geometric (perpendicular) projection onto the subspace of input vectors. This is shown conceptually in Figure 3.2 for a two-dimensional subspace. Note that the error vector \(e(n \mid n)\) and the estimate vector \(\hat{d}(n \mid n)\) are perpendicular; this is a general orthogonality principle for LS solutions. An operator matrix \(P\) can be formed having the desired property of projection onto a particular subspace. The operator which projects onto \(Z_0(n)\) is found by solving the matrix equation:

\[
\min_{w.r.t. \ w_N(n)} e^T(n \mid n)e(n \mid n) = \min_{w.r.t. \ w_N(n)} e(n) = \min_{w.r.t. \ w_N(n)} e(n). \]  

(3.13)

Differentiating \(e(n)\) with respect to \(w_N(n)\) and equating to zero yields the LS prediction...
where it is assumed for simplicity that the inverse exists. Note that the least squares estimate of an arbitrary vector is obtained by premultiplying by a matrix that is a function of only the input data vectors! The projection operator onto the subspace of \( Z_0(n) \) is thus defined

\[
(n \times n) \quad P_0(n) = Z_0(n)[Z_0^T(n)Z_0(n)]^{-1}Z_0^T(n).
\] (3.15)

The operator \( P_1(n) \) is similarly defined for the \( Z_1(n) \) subspace.

An operator can also be found to generate the LS error vector which is orthogonal to the \( Z_0(n) \) subspace. Substituting (3.14) into (3.9) gives

\[
e(n | n) = d(n) - P_0(n)d(n) = [I - P_0(n)]d(n) = P_0^l(n)d(n).
\] (3.16)

This defines the orthogonal projection operator.
\[(n \times n) \quad P_0^\perp(n) = I - P_0(n); \quad \text{similarly,} \quad (n \times n) \quad P_1^\perp(n) = I - P_1(n). \] 

These operators generate the projection of a vector perpendicular to their defining subspaces.

3.1.4. Transversal Filter Operators. From (3.14), the optimal transversal filter can be written

\[
w_N(n) = [Z_0^T(n)Z_0(n)]^{-1}Z_0^T(n)d(n). \tag{3.18} \]

A transversal filter operator (matrix) can then be defined

\[(N \times n) \quad K_0(n) = [Z_0^T(n)Z_0(n)]^{-1}Z_0^T(n). \tag{3.19} \]

By premultiplication, this operator creates the transversal filter (for the input data) which would generate the unique LS estimate of a vector on the subspace of \(Z_0(n)\). Similar operators can be defined on the other subspaces shown earlier:

\[(N \times n) \quad K_1(n) = [Z_1^T(n)Z_1(n)]^{-1}Z_1^T(n), \tag{3.20a,b} \]

\[(N+2 \times n) \quad K_+(n) = [Z_+^T(n)Z_+(n)]^{-1}Z_+^T(n). \]

Several transversal filters can be generated using these operators on previously defined vectors, including a forward predictor for the input channels, a backward predictor, and a gain filter. These filters, and their associated prediction errors and residuals (squared error vector norms) are summarized with definitions in Table 3.1. The need for these quantities will be seen during the algorithm derivation. Most are not directly evaluated from their definitions; instead, relationships between the quantities are exploited to allow tremendous savings in computations. Of particular interest, the gain filter \(g_N(n)\) is the LS filter on \(Z_0(n)\) for predicting
Table 3.1
Definitions for the General Order, Two-Channel FTF Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Transversal Filters</strong></td>
</tr>
<tr>
<td>1</td>
<td>$N \times 1$</td>
<td>$w_N(n) = K_0(n)d(n)$ joint predictor</td>
</tr>
<tr>
<td>2</td>
<td>$N \times 2$</td>
<td>$f_N(n) = K_1(n)z(n)$ forward predictor</td>
</tr>
<tr>
<td>3</td>
<td>$N \times 2$</td>
<td>$b_N(n) = K_0(n)z(n)$ backward predictor</td>
</tr>
<tr>
<td>4</td>
<td>$N \times 1$</td>
<td>$g_N(n) = K_0(n)\pi(n)$ gain (unit predictor)</td>
</tr>
<tr>
<td>5</td>
<td>$N \times 1$</td>
<td>$c_N(n) = \gamma^{-1}(n)g_N(n)$ normalized gain</td>
</tr>
<tr>
<td>6</td>
<td>$N+2 \times 1$</td>
<td>$g_{N+2}(n) = K_+(n)\pi(n)$ augmented gain</td>
</tr>
<tr>
<td>7</td>
<td>$N+2 \times 1$</td>
<td>$c_{N+2}(n) = \gamma_+(n)g_{N+2}(n)$ norm. augm. gain</td>
</tr>
<tr>
<td>8</td>
<td>$N+2 \times 1$</td>
<td>$g_{N+2}^f(n) = K_+(n)\pi(n)$ forward shifted augm. gain</td>
</tr>
<tr>
<td>9</td>
<td>$N+2 \times 1$</td>
<td>$c_{N+2}^f(n) = \gamma_+(n)g_{N+2}^f(n)$ norm. f. shift. augm. gain</td>
</tr>
<tr>
<td>10</td>
<td>$N+2 \times 1$</td>
<td>$g_{N+2}^b(n) = K_+(n)\pi(n)$ backward shift. augm. gain</td>
</tr>
<tr>
<td>11</td>
<td>$N+2 \times 1$</td>
<td>$c_{N+2}^b(n) = \gamma_+(n)g_{N+2}^b(n)$ norm. b. shift. augm. gain</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Prediction Errors</strong></td>
</tr>
<tr>
<td>12</td>
<td>$1 \times 1$</td>
<td>$e(n</td>
</tr>
<tr>
<td>13</td>
<td>$1 \times 1$</td>
<td>$e(n</td>
</tr>
<tr>
<td>14</td>
<td>$2 \times 1$</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>15</td>
<td>$2 \times 1$</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>16</td>
<td>$2 \times 1$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>17</td>
<td>$2 \times 1$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>18</td>
<td>$1 \times 1$</td>
<td>$\gamma(n) = 1 - g_N^T(n)z_N(n) = &lt;\pi(n), P_0^T(n)\pi(n)&gt;$</td>
</tr>
<tr>
<td>19</td>
<td>$1 \times 1$</td>
<td>$\gamma_+(n) = 1 - g_{N+2}^T(n)z_{N+2}(n)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Residuals</strong></td>
</tr>
<tr>
<td>20</td>
<td>$1 \times 1$</td>
<td>$e(n) = &lt;e(n</td>
</tr>
<tr>
<td>21</td>
<td>$2 \times 2$</td>
<td>$e_f(n) = &lt;e_f(n</td>
</tr>
<tr>
<td>22</td>
<td>$2 \times 2$</td>
<td>$e_b(n) = &lt;e_b(n</td>
</tr>
</tbody>
</table>
the pinning vector $\pi(n)$. The associated prediction error $\gamma(n)$ (see Table 3.1.18) was shown in [41] to be the squared cosine of the angle between $z(n)$ and its estimate in the subspace of $Z_1(n)$. Thus, $\gamma(n)$ is a measure of the amount of new information (innovation) in the latest data sample, and is a key parameter in the derivation.

3.1.5. Permutation Operators. The permutation operators $S$ defined here are row-permuted identity matrices, such that pre-multiplication of a vector by $S$ will cause rearrangement of the vector's rows. Such matrices have the important property of orthogonality: $SS^T = I$ or $S^T = S^{-1}$. The $(N+2) \times (N+2)$ 'forward' and 'backward' permutation operator matrices, $S_f$ and $S_b$ respectively, are defined by their effect on the augmented vector $z_{N+2}(n)$:

$$S_f z_{N+2}(n) = [x(n), y(n), x_{L}^T(n-1), y_{M}^T(n-1)]^T = \begin{bmatrix} z(n) \\ z_{N}(n-1) \end{bmatrix} \quad (3.21a,b)$$

$$S_b z_{N+2}(n) = [x_{L}^T(n), y_{M}^T(n), x(n-L), y(n-M)]^T = \begin{bmatrix} z_{N}(n) \\ z_{b}(n) \end{bmatrix}$$

From these definitions it is easy to also describe matrix column rearrangement:

$$Z_+(n)S_f^T = [x(n), y(n), X_1(n), Y_1(n)] = [z(n), Z_1(n)]$$

$$Z_+(n)S_b^T = [X_0(n), Y_0(n), z^{-L}x(n), z^{-M}y(n)] = [Z_0(n), z_{b}(n)] \quad (3.22a,b)$$

The need for permutation operators arises in the multichannel derivation because of the form of the order updates used for the transversal filter operators. These updates, shown in the next section, require that augmenting vectors be appended to either the left or right of the data matrix; however, from (3.11) it can be seen
that augmenting columns naturally appear 'inside' the data matrix. Thus, permutation operators are used to first rearrange these columns so that the update forms may be properly applied.

Using the property of orthogonality of the permutation operators, the transversal filter operator for the permuted, augmented subspace can be deduced:

\[
\begin{align*}
(N+2 \times n) \quad K_+^f(n) &= [S_f Z_+^T(n)Z_+(n)S_f^T]^{-1}S_f Z_+^T(n) = S_f K_+(n), \\
(N+2 \times n) \quad K_+^b(n) &= S_b K_+(n). 
\end{align*}
\]

(3.23a,b)

Pre-multiplying a vector with a transversal filter operator \( K \) yields the transversal filter which would generate the LS error estimate of that vector, in the defining subspace of the operator. Each row of \( K \) transforms the vector into a particular filter tap. Thus, permutation of the rows of \( K \) is equivalent to permutation of the elements of the transversal filter. This concept is seen in the following equations for permuted gain filters (refer to Table 3.1):

\[
\begin{align*}
(N+2 \times 1) \quad g^f_{N+2}(n) &= S_f g_{N+2}(n) = S_f K_+(n) \pi(n) = K_+^f(n) \pi(n), \\
(N+2 \times 1) \quad g^b_{N+2}(n) &= S_b g_{N+2}(n) = K_+^b(n) \pi(n). 
\end{align*}
\]

(3.24a,b)

Using the orthogonality property of \( S \), these can be combined to show:

\[
g^b_{N+2}(n) = S_b S_f^T g^f_{N+2}(n)
\]

(3.25)

3.2. Generalized Updates of Operators
3.2.1. Projection Operator Order Update. In order to derive the recursive algorithm, it is necessary to consider the problem of updating the projection operator when new columns are added to the defining subspace. In general, the projection operator onto a subspace $X$ is defined

\[ P_X = X <X,X>^{-1}X^T = X(X^TX)^{-1}X^T \]  

(3.26)

where $< \cdot, \cdot >$ represents the inner product on the subspace $X$. The properties of symmetry ($P_X^T = P_X$) and idempotence ($P_X^2 = P_X$) are readily verified for this operator. The general perpendicular projection operator can be defined:

\[ P_X^\perp = I - P_X, \]  

(3.27)

also with the properties of symmetry and idempotence. Augmenting the subspace $X$ with columns $Y$ yields the following order updates for the projection operators:

\[ P_{X,Y} = P_X + P_{(P_X^\perp Y)} = P_X + P_X^\perp Y <Y, P_X^\perp Y>^{-1}Y^TP_X^\perp, \]  

(3.28)

\[ P_{X,Y}^\perp = I - P_{X,Y} = P_X^\perp - P_X^\perp <Y, P_X^\perp Y>^{-1}Y^TP_X^\perp. \]

The projection operator is updated by incorporating only the new information from the augmented columns, namely that portion of the subspace spanned by the new columns which is orthogonal to the old subspace. Pre- and post-multiplying (3.28) by arbitrary vectors $u^T$ and $v$ gives the useful identity

\[ <u, P_{X,Y}^\perp v> = <u, P_X^\perp v> - <u, P_X^\perp Y> <Y, P_X^\perp Y>^{-1} <Y, P_X^\perp v> \]  

(3.29)

3.2.2. Transversal Filter Operator Order Update. The general transversal filter operator defined on the subspace $X$ is

\[ K_X = <X,X>^{-1}X^T, \]  

(3.30)

and the following properties are easily demonstrated:
\[ K_X X = I, \quad X K_X = P_X, \quad K_X P_X = K_X. \] (3.31)

From (3.31) and assuming \( X_{n \times N}, Y_{n \times p} \)

\[ K_{X,Y}[X_{n \times N}, Y_{n \times p}] = I_{N+p} = \begin{bmatrix} I_N & 0_{N \times p} \\ 0_p & I_p \end{bmatrix}. \]

Thus, using (3.28), the update for post-appended columns is

\[ K_{X,Y} = K_{X,Y}P_{X,Y} = \begin{bmatrix} K_X \\ 0 \end{bmatrix} + \begin{bmatrix} -K_{XY} \\ I \end{bmatrix} <Y, P_Y^{-1}Y^T P_Y^T >. \] (3.32a)

Similarly, it can be shown for prepended columns

\[ K_{Y,X} = \begin{bmatrix} 0 \\ K_X \end{bmatrix} + \begin{bmatrix} I \\ -K_{XY} \end{bmatrix} <Y, P_Y^{-1}Y^T P_Y^T >. \] (3.32b)

3.2.3. Time Updates of Operators. Using (3.28) and (3.32a) with \( X = Z_0(n) \)
or \( X = Z_1(n) \) and \( Y = \pi(n) \), it can be shown that

\[ P_{0,\pi}(n) = \begin{bmatrix} P_0(n-1) & 0 \\ 0^T & 1 \end{bmatrix} \quad \text{and} \quad P_{0,\pi}^T(n) = \begin{bmatrix} P_0^T(n-1) & 0 \\ 0 & 1 \end{bmatrix} \] (3.33)

\[ P_{1,\pi}(n) = \begin{bmatrix} P_1(n-1) & 0 \\ 0^T & 1 \end{bmatrix} \quad \text{and} \quad P_{1,\pi}^T(n) = \begin{bmatrix} P_1^T(n-1) & 0 \\ 0 & 1 \end{bmatrix}. \]

Also, the transversal filter operator for the subspace augmented with \( \pi(n) \) can be expressed

\[ K_{0,\pi}(n) = \begin{bmatrix} K_0(n-1) & 0 \\ -z_N^T(n)K_0(n-1) & 1 \end{bmatrix}, \quad K_{1,\pi}(n) = K_1(n-1) \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \] (3.34)

The derivation of (3.33, 34) is shown in detail in appendix 3A; it will be important for extensions of the algorithm shown in later chapters. These two sets of equations, along with (3.28) and (3.32) can be systematically applied to complete the
vector space derivation of the fast transversal algorithm.

3.3. Derivation of the Fast Multichannel Algorithm

Now that the preliminary updates and definitions have been determined, it is possible to derive a recursive solution to the LS problem posed initially. The derivation basically requires substitutions into the update forms, and reductions using the definitions. The derivation begins with an update form for the joint prediction filter, and then systematically finds recursions for other variables as they appear. The use of the permutation operator is seen in sections 3.3.3, 10, 11, which deal with updates of the gain filter and its associated prediction error. The complete algorithm is summarized in Table 3.2 in proper order of execution. An operation count is also provided for each recursion.

3.3.1. Joint Prediction Filter. From (3.32a) with $X = Z_0(n)$, $Y = \pi(n)$,

$$K_{0,\pi}(n) = \begin{bmatrix} K_0(n) \\ 0 \end{bmatrix} + \begin{bmatrix} -K_0(n)\pi(n) \\ 1 \end{bmatrix} <\pi(n), \quad P_0(n)\pi(n) >^{-1}\pi^T(n)P_0^I(n)$$  \hspace{1cm} (3.35)

Post-multiply (3.35) by $d(n)$, and use expansion (3.34) with definitions (3.1.4, 18):

$$\begin{bmatrix} K_0(n-1) \\ d(n) \end{bmatrix} = \begin{bmatrix} K_0(n) \\ d(n) \end{bmatrix} - \begin{bmatrix} g_N(n) \\ -1 \end{bmatrix} \gamma^{-1}(n) <\pi(n), \quad P_0^I(n)d(n) >$$

Take just the upper partitions and use (3.1.1, 5, 13)

$$w_N(n) = w_N(n-1) + c_N(n)\epsilon(n|n).$$  \hspace{1cm} (3.36)
Table 3.2
The General Order, Two-Channel Transversal RLS Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2 \times 1$</td>
<td>$2N$</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>2</td>
<td>$2 \times 1$</td>
<td>2</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>3a</td>
<td>$2 \times 2$</td>
<td>3*</td>
<td>$e_f(n) = e_f(n-1) + e_f(n</td>
</tr>
<tr>
<td>3b</td>
<td>$2 \times 2$</td>
<td>5*</td>
<td>$e_f^{-1}(n): 2 \times 2$ matrix inverse</td>
</tr>
<tr>
<td>4</td>
<td>$1 \times 1$</td>
<td>6</td>
<td>$\gamma_+(n) = \gamma(n-1) - e^T_f(n</td>
</tr>
<tr>
<td>5a</td>
<td>$N+2 \times 1$</td>
<td>$2N+2$</td>
<td>$c^b_{N+2}(n) = \begin{bmatrix} 0 \ [c_N(n-1)] \end{bmatrix} + \begin{bmatrix} I_2 \ -f_N(n-1) \end{bmatrix} e_f^{-1}(n)e_f(n</td>
</tr>
<tr>
<td>5b</td>
<td>$N+2 \times 1$</td>
<td>0**</td>
<td>$c^b_{N+2}(n) = S_bS^T_f e^f_{N+2}(n)$</td>
</tr>
<tr>
<td>5c</td>
<td>$2 \times 1$</td>
<td>0**</td>
<td>$c^b_+(n) = \text{last 2 elements in } c^b_{N+2}(n) \text{ vector}$</td>
</tr>
<tr>
<td>6</td>
<td>$N \times 2$</td>
<td>$2N$</td>
<td>$f_N(n) = f_N(n-1) + c^b_N(n</td>
</tr>
<tr>
<td>7</td>
<td>$2 \times 1$</td>
<td>4</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>8</td>
<td>$1 \times 1$</td>
<td>4</td>
<td>$\gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b^T(n</td>
</tr>
<tr>
<td>9</td>
<td>$2 \times 1$</td>
<td>2</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>10</td>
<td>$2 \times 2$</td>
<td>3*</td>
<td>$e_b(n) = e_b(n-1) + e_b(n</td>
</tr>
<tr>
<td>11</td>
<td>$N \times 1$</td>
<td>$2N$</td>
<td>$\begin{bmatrix} c_N(n) \ 0 \end{bmatrix} = c^b_{N+2}(n) - \begin{bmatrix} -b_N(n-1) \ +I_2 \end{bmatrix} c^b_+(n)$</td>
</tr>
<tr>
<td>12</td>
<td>$N \times 2$</td>
<td>$2N$</td>
<td>$b_N(n) = b_N(n-1) + c_N(n)e^T_b(n</td>
</tr>
<tr>
<td>13</td>
<td>$1 \times 1$</td>
<td>$N$</td>
<td>$e(n</td>
</tr>
<tr>
<td>14</td>
<td>$1 \times 1$</td>
<td>1</td>
<td>$e(n</td>
</tr>
<tr>
<td>15</td>
<td>$N \times 1$</td>
<td>$N$</td>
<td>$w_N(n) = w_N(n-1) + c_N(n)e(n</td>
</tr>
<tr>
<td>Total OPS</td>
<td>$12N + 32$</td>
<td>* matrix symmetry allows reduction in OPS. ** since vector rearrangement is predetermined.</td>
<td></td>
</tr>
</tbody>
</table>
3.3.2. Joint Process Error. Transpose (3.36), post-multiply by $z_N(n)$ and subtract from $d(n)$:

$$d(n) - w_N^T(n)z_N(n) = d(n) - w_N^T(n-1)z_N(n) - e(n|n)z_N^T(n)z_N(n)$$

(3.37)

With definitions (3.1.12, 13)

$$e(n|n) = \gamma(n)e(n|n-1).$$

(3.38)

3.3.3. Gain Filter. From (3.22), with $Y = z(n)$ and $X = Z_1(n)$, the subspace $[Y, X]$ is $Z_+(n)S_f^T$; the associated TF operator is $K_1^+(n)$. Use (3.32b) and post-multiply by $\pi(n)$,

$$K_1^+(n)\pi(n) = \begin{bmatrix} 0 \\ K_1(n) \end{bmatrix} \pi(n) + \begin{bmatrix} I \\ -K_1(n)z(n) \end{bmatrix} <z(n), P_1^+(n)z(n)>^{-1} <z(n), P_1^+(n)\pi(n)>
$$

It is readily shown that $K_1(n)\pi(n) = g_N(n-1)$; with definitions (3.1.2, 8, 15, 21)

$$g_{N+2}^+(n) = \begin{bmatrix} 0 \\ g_N(n-1) \end{bmatrix} + \begin{bmatrix} I_2 \\ -f_N(n) \end{bmatrix} e_f^{-1}(n)e_f(n|n).
$$

(3.39)

Similarly, from (3.32a) with $X = Z_0(n)$, $Y = z_b(n)$, post-multiply by $\pi(n)$

$$K_+^b(n)\pi(n) = \begin{bmatrix} K_0(n) \\ 0 \end{bmatrix} \pi(n) + \begin{bmatrix} -K_0(n)z_b(n) \\ I \end{bmatrix} \cdot <z_b(n), P_0^+(n)z_b(n)>^{-1} <z_b(n), P_0^+(n)\pi(n)>
$$

From definitions (3.1.3, 4, 10, 17, 22)

$$g_{N+2}^b(n) = \begin{bmatrix} g_N(n) \\ 0 \end{bmatrix} + \begin{bmatrix} -b_N(n) \\ I_2 \end{bmatrix} e_b^{-1}(n)e_b(n|n)
$$

(3.40)

or with (3.25)
\[
\begin{bmatrix}
g_b(N(n)) \\
0
\end{bmatrix} = S_b S_f^T g_{N+2}^b(n) + \begin{bmatrix} b_N(n) \\
-I_2
\end{bmatrix} \epsilon_b^{-1}(n) e_b(n \mid n).
\]

From (3.40) we define the last \((2 \times 1)\) element of \(g_{N+2}^b(n)\) as

\[ g^b_+(n) = \epsilon_b^{-1}(n) e_b(n \mid n). \]

Also, with (3.1.11), the last element of \(c_{N+2}^b(n)\) is defined

\[ c^b_+(n) = \gamma_+^{-1}(n) \epsilon_b^{-1}(n) e_b(n \mid n). \]

### 3.3.4. Forward Prediction Filter.

Use (3.32a) with \(X = Z_1(n), Y = \pi(n)\) and expansion (3.34); post-multiply by \(z(n)\) and take only the top partition:

\[ f_N(n) = f_N(n-1) + c_N(n-1) e_f^T(n \mid n). \]

### 3.3.5. Forward Prediction Error.

Transpose (3.43), post-multiply by \(z_N(n-1)\), and subtract from \(z(n)\):

\[ z(n) - f_N^T(n) z_N(n-1) = z(n) - f_N^T(n-1) z_N(n-1) - e_f(n \mid n) c_N^T(n-1) z_N(n-1) \]

With definitions

\[ e_f(n \mid n) = \gamma(n-1) e_f(n \mid n-1). \]

### 3.3.6. Forward Prediction Residual.

Use (3.29) with \(X = Z_1(n), Y = \pi(n)\), and \(u = v = z(n)\),

\[ <z(n), P_{1,\pi}^1(n) z(n)> = <z(n), P_{1}^1(n) z(n)> - <z(n), P_1^1(n) \pi(n)> \cdot <\pi(n), P_1^1(n) \pi(n)>^{-1} <\pi(n), P_1^1(n) z(n)> \]

Use (3.33) and partition \(z(n) = \begin{bmatrix} z(n-1) \\
T(n)
\end{bmatrix},\)
\[ \epsilon_f(n) = \epsilon_f(n-1) + \epsilon_f(n \mid n-1) \epsilon_f^T(n \mid n). \] (3.47)

3.3.7. **Backward Prediction Filter.** Use (3.32b) with \( Y = \pi(n), \ X = Z_0(n), \) postmultiply by \( z_b(n). \)

\[ b_N(n) = b_N(n-1) + c_N(n) e_b^T(n \mid n). \] (3.48)

3.3.8. **Backward Prediction Error.** Transpose (3.48), post-multiply by \( z_N(n) \) and subtract from \( z_b(n): \)

\[ z_b(n) - b_N^T(n) z_N(n) = z_b(n) - b_N^T(n-1) z_N(n) - e_b(n \mid n) c_N^T(n) z_N(n) \] (3.49)

With definitions

\[ e_b(n \mid n) = \gamma(n) e_b(n \mid n-1). \] (3.50)

3.3.9. **Backward Prediction Residual.** Use (3.29), with \( X = Z_0(n), \ Y = \pi(n), \) and \( u = v = z_b(n); \) use (3.33) and partition \( z_b(n) = \begin{bmatrix} z_b(n-1) \\ z_b^T(n) \end{bmatrix}, \)

\[ \epsilon_b(n) = \epsilon_b(n-1) + \epsilon_b(n \mid n-1) e_b^T(n \mid n). \] (3.51)

3.3.10. **Gain Error.** Use (3.1.19) and substitute (3.24a)

\[ \gamma_+(n) = 1 - g_{N+2}^T(n) S_f z_{N+2}(n). \]

Use (3.39) and (3.21a). Definitions (3.1.15, 18) provide

\[ \gamma_+(n) = \gamma(n-1) - \epsilon_f^T(n \mid n) \epsilon_f^{-1}(n) \epsilon_f(n \mid n). \] (3.52)

Similarly, use (3.1.19) and substitute (3.24b),

\[ \gamma_+(n) = 1 - g_{N+2}^b(n) S_b z_{N+2}(n) \]

Now, use (3.40) and (3.21b).
\[
\gamma_+(n) = \gamma(n) - e_b^T(n|n)\epsilon_b^{-1}(n)e_b(n|n).
\] (3.53)

Thus, with (3.42b) and (3.50), (3.53) becomes

\[
\gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b^T(n|n-1)c_+^b(n)]^{-1}.
\] (3.54)

### 3.3.11. Normalized Gain Filter

Substitute (3.43) into (3.39); use (3.53) and divide through by \(\gamma_+(n)\)

\[
\epsilon_{N+2}(n) = \begin{bmatrix}
0 \\
\epsilon_N(n-1)
\end{bmatrix} + \begin{bmatrix}
I_2 \\
-f_N(n-1)
\end{bmatrix}\epsilon_{-1}^{-1}(n)e_f(n|n)\gamma_+^{-1}(n).
\] (3.55)

Similarly, use (3.40), substitute (3.48), and divide by \(\gamma_+(n)\)

\[
\begin{bmatrix}
\epsilon_N(n) \\
0
\end{bmatrix} = \epsilon_{N+2}(n) + \begin{bmatrix}
\epsilon_{N+2}(n) \\
-I_2
\end{bmatrix}c_+^b(n).
\] (3.56)

### 3.3.12. Simpler Backward Prediction Error Update

Instead of using the definition (3.16) to evaluate \(e_b(n|n-1)\), requiring \(2N\) multiplies, a simpler form can be found. Post-multiply (3.51) by (3.42b) to yield

\[
e_b(n)c_+^b(n) = \gamma_+^{-1}(n)e_b(n|n) = e_b(n-1)c_+^b(n) + e_b(n|n)e_b^T(n|n-1)c_+^b(n).
\] (3.57)

Using simple algebra,

\[
e_b(n-1)c_+^b(n) = \gamma_+^{-1}(n)e_b(n|n)[1 - \gamma_+(n)e_b^T(n|n-1)c_+^b(n)],
\] (3.58)

but from (3.54), the term in brackets is equal to \(\gamma_+(n)\gamma^{-1}(n)\). With (3.50),

\[
e_b(n|n-1) = e_b(n-1)c_+^b(n).
\] (3.59)

This form requires only \(p^2\) multiplies, where \(p\) is the number of channels.
3.4. The Higher Order Multichannel Case

This section considers some extensions of the previous sections to the general \( p \)-channel case, rather than only two input channels. In general, this is a straightforward procedure and will not be shown here in detail, since only a change in the dimensionality of vectors and matrices used in Table 3.2. is required. However, from a computational standpoint, certain savings are possible, and derivations of these will be shown next.

If the number of channels \( p \) becomes large, table equation (3.2.3b) becomes computationally costly, since a \( p \times p \) matrix inverse is required. With standard Gaussian inversion, this will require \( O(p^3) \) multiplications. It is possible to reduce this to an \( O(p^2) \) process, as will now be described.

From (3.47) we can write

\[
e_f(n|n-1)e_f^T(n|n) = \epsilon_f(n) - \epsilon_f(n-1) = A
\]  

(3.60)

where matrix \( A \) is a notational convenience. Now taking (3.52), and premultiplying by \( e_f(n|n-1) \) and post-multiplying by \( e_f^T(n|n) \) gives

\[
\gamma_+(n)A = \gamma(n-1)A - \gamma(n-1)A\epsilon_f^{-1}(n)A
\]  

(3.61)

Finally, post-multiplying by \( A^{-1}\epsilon_f(n) \), and re-arranging gives

\[
\epsilon_f(n) = \gamma_+^{-1}(n)\gamma(n-1)\epsilon_f(n-1)
\]  

(3.62)

This important relationship is now used to modify (3.52) and (3.55). For computational convenience, we define the intermediate variables
The algorithm can be manipulated so that these variables will recur and thus reduce required computations. Note that $a(n)$ requires $p^2$ multiplications (OPS) to compute and $a(n)$ requires $p + 1$ OPS. Substituting (3.62) for $e_f(n)$ in (3.52) and re-arranging gives

$$\gamma_+(n) = \gamma(n-1)a(n)$$

(3.64)

Using the matrix inverse lemma with (3.47), we can then write

$$e_f^{-1}(n) = e_f^{-1}(n-1) - \gamma_+(n)a(n)a^T(n)$$

(3.65)

Because of the symmetry of the $e_f$ matrix, this equation requires $p + \frac{1}{2}(p^2 + p)$ OPS to compute. (It is interesting to note that (3.62) can not be used directly to update $e_f^{-1}(n)$. The scalar multiplication would erroneously preserve the initial value of the matrix.) Finally, with the new relationships, (3.55) becomes

$$c_{N+p}(n) = \begin{bmatrix} 0_p \\ c_N(n-1) \end{bmatrix} + \begin{bmatrix} I_p \\ -f_N(n-1) \end{bmatrix} a(n)$$

(3.66)

This equation requires $pN$ OPS to implement.

The complete $p$-channel, general order algorithm is summarized in Table 3.3. The total number of OPS required is $(5p + 2)N + 3p^2 + 6p + 4$. For $p = 2$, this total is $12N + 28$, which is less than the "direct inversion" method for two channels of the previous section. For $p > 2$, this approach is clearly superior.
Table 3.3  
The General Order, $p$-Channel Transversal RLS Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$p \times 1$</td>
<td>$pN$</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>2</td>
<td>$p \times 1$</td>
<td>$p$</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>3a</td>
<td>$p \times 1$</td>
<td>$p^2$</td>
<td>$a(n) = e_f^{-1}(n-1)e_f(n</td>
</tr>
<tr>
<td>3b</td>
<td>$1 \times 1$</td>
<td>$p+1$</td>
<td>$\gamma_+(n) = \gamma(n-1)[1 + e_f^T(n</td>
</tr>
<tr>
<td>4</td>
<td>$p \times p$</td>
<td>$\frac{1}{2}(p^2+3p)$</td>
<td>$e_f^{-1}(n) = e_f^{-1}(n-1) - \gamma_+(n)a(n)a^T(n)$</td>
</tr>
<tr>
<td>5a</td>
<td>$N+p \times 1$</td>
<td>$pN$</td>
<td>$c_{N+p}^f(n) = \begin{bmatrix} 0_p \ c_{N(n-1)} \ -f_N(n-1) \end{bmatrix}a(n)$</td>
</tr>
<tr>
<td>5b</td>
<td>$N+p \times 1$</td>
<td>$0$</td>
<td>$c_{N+p}^b(n) = S_bS_f^Tc_{N+p}^f(n)$</td>
</tr>
<tr>
<td>5c</td>
<td>$p \times 1$</td>
<td>$0$</td>
<td>$c_+^b(n) = \text{last } p \text{ elements in } c_{N+p}^b(n) \text{ vector}$</td>
</tr>
<tr>
<td>6</td>
<td>$N \times p$</td>
<td>$pN$</td>
<td>$f_N(n) = f_N(n-1) + c_{N(n-1)}e_f^T(n</td>
</tr>
<tr>
<td>7</td>
<td>$p \times 1$</td>
<td>$p^2$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>8</td>
<td>$1 \times 1$</td>
<td>$p+2$</td>
<td>$\gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b^T(n</td>
</tr>
<tr>
<td>9</td>
<td>$p \times 1$</td>
<td>$p$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>10</td>
<td>$p \times p$</td>
<td>$\frac{1}{2}(p^2+p)$</td>
<td>$e_b(n) = e_b(n-1) + e_b(n</td>
</tr>
<tr>
<td>11</td>
<td>$N \times 1$</td>
<td>$pN$</td>
<td>$\begin{bmatrix} c_N(n) \ 0_p \end{bmatrix} = c_{N+p}^b(n) - \begin{bmatrix} -b_N(n-1) \ I_p \end{bmatrix}c_+^b(n)$</td>
</tr>
<tr>
<td>12</td>
<td>$N \times p$</td>
<td>$pN$</td>
<td>$b_N(n) = b_N(n-1) + c_N(n)e_b^T(n</td>
</tr>
<tr>
<td>13</td>
<td>$1 \times 1$</td>
<td>$N$</td>
<td>$e(n</td>
</tr>
<tr>
<td>14</td>
<td>$1 \times 1$</td>
<td>$1$</td>
<td>$e(n</td>
</tr>
<tr>
<td>15</td>
<td>$N \times 1$</td>
<td>$N$</td>
<td>$w_N(n) = w_N(n-1) + c_N(n)e(n</td>
</tr>
</tbody>
</table>

Total OPS = $(5p + 2)N + 3p^2 + 6p + 4$
APPENDIX 3A

Operator Expansions for $\pi$-Augmented Subspaces

This appendix shows detailed proofs of equations (3.33, 34), decompositions of the projection and transversal filter operators for a subspace augmented with the "pinning" vector, $\pi$. These are important relationships, used in the derivation of all the transversal filter update equations. They are also critical in the proper development of the exponentially windowed version of the FTF algorithm, as described in Chapter 5.

Theorem 1: Given an $(n \times N)$ matrix $U$ with its final row partitioned:

$$ U = \begin{bmatrix} A \\ a^T \end{bmatrix} \quad (3A.1) $$

Define an augmented matrix $V$ with the $(n \times 1)$ unit vector $\pi$ right appended:

$$ V = [U \ \pi] = \begin{bmatrix} A & 0 \\ a^T & 1 \end{bmatrix} \quad (3A.2) $$

If the transversal filter operator for an arbitrary subspace $X$ is defined

$$ K_X = (X^T X)^{-1} X^T \quad (3A.3) $$

then the transversal filter operator for the subspace $V$ can be expanded as

$$ K_V = \begin{bmatrix} K_A & 0 \\ -a^T K_A & 1 \end{bmatrix} \quad (3A.4) $$

Proof: Substituting (3A.2) into the definition of $K_V$ gives

$$ K_V = \begin{bmatrix} A^T A + a a^T & a \\ a^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} A^T \\ a^T \\ 0^T \\ 1 \end{bmatrix} \quad (3A.5) $$
From a matrix inversion lemma for partitioned matrices [46], if $P^{-1}$ is defined, $q$, $r$ are column vectors, and $s$ is scalar, then

$$
\begin{bmatrix}
P & q \\
r^T & s
\end{bmatrix}^{-1} = \begin{bmatrix} P^{-1} + tP^{-1}q r^T P^{-1} & -tP^{-1}q \\
-t r^T P^{-1} & t
\end{bmatrix}
$$

with $t = [s \cdot r^T P^{-1} q]^{-1}$. This expansion can be verified by direct multiplication.

For convenience, now define

$$B = A^T A + aa^T \quad (3A.7)$$

This then gives

$$K_V = \begin{bmatrix}
(B^{-1} + \beta B^{-1} aa^T B^{-1}) A^T & (B^{-1} + \beta B^{-1} aa^T B^{-1}) a - \beta B^{-1} a \\
-\beta a^T B^{-1} A^T & -\beta a^T B^{-1} a + \beta
\end{bmatrix} \quad (3A.8a,b)$$

$$\beta = [1 - a^T B^{-1} a]^{-1}$$

From a lemma for the inverse of summed matrices [46], if $P^{-1}$ is defined, $q$, $r$ are column vectors, and $s$ is scalar, then

$$
(P + q s r^T)^{-1} = P^{-1} - tP^{-1} q r^T P^{-1}
$$

with $t = [s \cdot r^T P^{-1} q]^{-1}$. Thus, (3A.7) provides

$$B^{-1} = (A^T A)^{-1} - \alpha (A^T A)^{-1} aa^T (A^T A)^{-1} \quad (3A.10a,b)$$

$$\alpha = [1 + a^T (A^T A)^{-1} a]^{-1}$$

It can readily be shown that $\beta = \alpha^{-1}$. These definitions can now be substituted into each term of (3A.8a) to complete the proof. For an alternative proof, evaluate equation (3.32b) with $U = \pi$. □

**Discussion:** The projection operator for a subspace determines (by premultiplication) the LS estimate vector of an arbitrary vector, within the subspace.
This implies that the estimate vector is a linear combination of the columns of the subspace. The transversal filter operator, in effect, determines the particular weight to apply to each column to generate the estimate vector. The first row of the transversal filter operator matrix (times the arbitrary vector) generates the weight for the first column, etc. Consider now the right augmentation of the subspace by the \( \pi \) vector. This new column is also weighted to contribute to the LS estimate vector; however, since it is only non-zero in the last row, its column weight cannot affect the estimate error except in the final row. Thus, since squared error is minimized for the LS estimate, the weight for the \( \pi \) column is selected to exactly zero any estimation error due to the last row. The LS weights for the other columns are determined as if the final row did not exist. This is exactly what (3A.4) expresses. The name "time annihilation" vector sometimes given to \( \pi \) is thus explained, since the final row of the subspace matrix contains the most recent data values.

**Theorem 2**: If \( U \) and \( V \) are defined as in theorem 1, and the projection operator for the subspace \( X \) is defined

\[
P_X = X(X^TX)^{-1}X^T
\]  

then the projection operator for the subspace \( V \) can be expanded as

\[
P_V = \begin{bmatrix} P_A & 0 \\ 0^T & 1 \end{bmatrix}
\]  

**Proof**: From the operator definitions (3A.3, 11), \( P_V = V K_V \). Substitute (3A.2) and (3A.4) and reduce. \( \square \)
Finally, there is an additional simple theorem that will prove useful in later derivation.

**Theorem 3:** Given $A = \beta B$, with $A$, $B$ matrices and $\beta$ scalar; then

\[
P_A = P_B \\
K_A = \beta^{-1} K_B
\]  

(3A.13a,b)

**Proof:** Direct substitution into the operator definitions (3A.3, 11). □
4.1. Introduction

The application of transversal least squares (LS) algorithms to certain problems of real-time system identification has become feasible with the discovery of "fast" recursive LS algorithms. These adaptive algorithms are known as the Fast Kalman [22], FAEST [31], and the FTF [32] algorithms. Because the algorithms can perform their update computations in a time directly proportional to the filter length, $N$, they are suitable for real-time implementation. Since these algorithms have a substantial advantage in convergence speed in many situations, they can compete with the ubiquitous order $N$ LMS (gradient) algorithm.

One important problem is ARMA (autoregressive, moving average) identification of systems. ARMA estimation can be important in such fields as adaptive echo cancellation and equalization. These practical applications often require an exorbitant number of taps when implemented with the standard transversal (FIR, or all-zero) type configuration, because the impulse response of a real system is often very long. The transversal model forms the impulse response directly, with each tap coefficient corresponding to a sample of the impulse response. By contrast, an IIR filter forms a parametric model for a system which

‡ This chapter appeared, with certain modifications, in [35] and [36].
can model lengthy impulse responses with many fewer coefficients.

One problem with adaptive IIR filters is their potential for instability, since adaptation algorithms can force response poles outside the z-domain unit circle. However, if the filter is configured in the "equation error" form [23], as illustrated in Figure 4.1, the resultant filter is always stable. This is because the equation error form is not a true IIR form, with feedback paths. All signals are actually filtered with unconditionally stable FIR filters. (The filter update algorithm's stability is not guaranteed, however!) Additionally, since this form is linear in the filter parameters (tap weights), a unique LS solution is always available [23]. The output of the unknown process is filtered, as well as the input signal, to produce an

Figure 4.1 Equation Error (after Astrom[23])
estimate of the output of the system. The error of this estimate can be used to adjust the parameters of the model (filter) in order to minimize the LS error of the estimate.

The Fast Transversal Filter (FTF) algorithm [32] is a fast (order \( N \)), recursive, least squares algorithm. As such, it computes the LS filter solution in an extremely efficient manner. A multi-channel version of this algorithm allows multiple input channels and multiple output channels. In Chapter 3, the multi-channel FTF algorithm was extended to the case where various input channels might have different filter orders. The two input channel case of this algorithm was shown to require \(12N + 27\) multiply or divide operations per recursive time update, where \( N \) is the total number of filter taps. In this chapter, we show how the general order, two input channel FTF algorithm can be applied to the problem of ARMA system identification. This configuration is also the one used for decision-directed equalization or echo cancellation, and hence has important applications. For these applications, we will show that a computational savings of \(2N + 1\) operations per update can be achieved.

Consider a two input channel joint process predictor, where now one of the input channels is connected to the unit-delayed† joint process signal, \(d(n)\). This is depicted in Figure 4.2. The indicated computational savings result since the

† It is assumed that all transversal filters have a tap weight associated with the immediate input sample. The joint process signal must be unit delayed, or the optimal prediction filter would trivially be unity for the first parameter, and zero for the rest.
Figure 4.2 Modified Two-Channel Joint Predictor for ARMA Estimation
internal structure of the FTF algorithm requires the recursive computation of forward and backward predictors for the input channels. Because the ultimate goal of the joint process algorithm is the forward prediction of the desired signal, it should seem reasonable that the forward predictor for the unit-delayed desired signal (one of the input channels) is related to the joint process filter. This will be proven. Similarly, errors associated with these filters are related. These relationships are exploited to obtain the indicated computational reduction. The reduction available with this special configuration will also apply to the Fast Kalman and the FAEST fast LS algorithms, which also use forward predictors of the input channels.\textsuperscript{‡} This had not previously been recognized for applications of these algorithms.

4.2. ARMA System Identification

If a finite order, rational spectral model is postulated for an unknown linear system, with input \( \{x(n)\} \) and output \( \{d(n)\} \), the system transfer function can be expressed in the z-domain as

\[
H(z) = \frac{D(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_{L-1} z^{-L+1}}{1 - a_1 z^{-1} - \cdots - a_M z^{-M}}
\]

The roots of the numerator and denominator polynomials of \( H(z) \) correspond to the zeroes and poles of the system transfer function. This relationship is expressed in the time domain as an autoregressive-moving average (ARMA) difference equa-

\textsuperscript{‡} It was shown in [12] that all three algorithms are equivalent mathematically, i.e. with infinite precision arithmetic, since they all solve the LS problem.
Thus, it is assumed that there is also an additive uncorrelated noise term, and that the zero and pole polynomial orders \( L \) and \( M \) are known. This equation suggests a form for the estimate of the joint process \( \{d(n)\} \) at time \( n \) that is linear in filter parameters available at time \( n \):

\[
d(n) = \sum_{i=0}^{L-1} b_i x(n-i) + \sum_{i=1}^{M} a_i d(n-i),
\]

(4.1)

Thus, it is assumed that there is also an additive uncorrelated noise term, and that the zero and pole polynomial orders \( L \) and \( M \) are known. This equation suggests a form for the estimate of the joint process \( \{d(n)\} \) at time \( n \) that is linear in filter parameters available at time \( n \):

\[
d(n) = \sum_{i=0}^{L-1} b_i x(n-i) + \sum_{i=1}^{M} a_i d(n-i),
\]

where \( \alpha_i(n) \) and \( \beta_i(n) \) are adaptive weighting coefficients evaluated at time \( n \). A prediction error is then

\[
e(n|n) = d(n) - \hat{d}(n|n).
\]

Note that the prediction in (4.2) is formed from a linear combination of previous \( \{d(n)\} \) rather than a regression on \( \{\hat{d}(n|n)\} \). This is sometimes called the "equation error" formulation [23], and permits a tractable solution to the LS problem.

The general-order multichannel FTF algorithm formulation was derived in detail in the previous chapter (and [33]) and was summarized in Table 3.2. This algorithm can be effectively applied to this problem of ARMA system identification [34]. The concept is to use the output of the process to be identified as input to one of the data channels. This scheme is as described in Figure 4.2, where the (unit-delayed) output of the system to be identified, \( d(n) \), is included as one input to the two-channel, joint process prediction filter \( w_N(n) \). Important definitions for this special case are summarized in Table 4.1.
Table 4.1
Definitions for the Two-Channel Fast Transversal Configuration

<table>
<thead>
<tr>
<th>Vectors and Matrices</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $z_N(i) = [x_L(i), y_M(i)]^T$, with $L + M = N$</td>
<td></td>
</tr>
<tr>
<td>2 $x_L(i) = [x(i), x(i-1), \ldots, x(i-L+1)]^T$</td>
<td></td>
</tr>
<tr>
<td>3 $y_M(i) = [y(i), y(i-1), \ldots, y(i-M+1)]^T$</td>
<td></td>
</tr>
<tr>
<td>4 $x(n) = [x(1), x(2), \ldots, x(n)]^T$</td>
<td></td>
</tr>
<tr>
<td>5 $y(n) = [y(1), y(2), \ldots, y(n)]^T$</td>
<td></td>
</tr>
<tr>
<td>6 $z(n) = [x(n), y(n)]$; $z_b(n) = [z^{-L}x(n), z^{-M}y(n)]$</td>
<td></td>
</tr>
<tr>
<td>7 $z(n) = [x(n), y(n)]^T$; $z_b(n) = [x(n-L), y(n-M)]^T$</td>
<td></td>
</tr>
<tr>
<td>8 $Z_0(n) = [z^0x(n), \ldots, z^{-L+1}x(n), y(n), \ldots, z^{-M+1}y(n)]$</td>
<td></td>
</tr>
<tr>
<td>9 $Z_1(n) = [z^{-1}x(n), \ldots, z^{-L}x(n), z^{-1}y(n), \ldots, z^{-M}y(n)]$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10 $P(n) = Z(n)(Z^T(n)Z(n))^{-1}Z^T(n)$ projection</td>
<td></td>
</tr>
<tr>
<td>11 $K(n) = [Z^T(n)Z(n)]^{-1}Z^T(n)$ transversal filter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transversal Filters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12 $w_N(n) = K_0(n)d(n)$ joint process predictor</td>
<td></td>
</tr>
<tr>
<td>13 $f_N(n) = K_1(n)z(n)$ forward predictor</td>
<td></td>
</tr>
<tr>
<td>14 $b_N(n) = K_0(n)z_b(n)$ backward predictor</td>
<td></td>
</tr>
<tr>
<td>15 $g_N(n) = K_0(n)\pi(n)$ gain (unit predictor)</td>
<td></td>
</tr>
<tr>
<td>16 $c_N(n) = \gamma^{-1}(n)g_N(n)$ normalized gain</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prediction Errors</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17 $e(n</td>
<td>n-1) = d(n) - w_T^T(n-1)z_N(n)$</td>
</tr>
<tr>
<td>18 $e(n</td>
<td>n) = d(n) - w_T^T(n)z_N(n)$</td>
</tr>
<tr>
<td>19 $e_f(n</td>
<td>n-1) = z(n) - f_T^T(n-1)z_N(n-1)$</td>
</tr>
<tr>
<td>20 $e_f(n</td>
<td>n) = z(n) - f_T^T(n)z_N(n-1)$</td>
</tr>
<tr>
<td>21 $e_b(n</td>
<td>n-1) = z_b(n) - b_T^T(n-1)z_N(n)$</td>
</tr>
<tr>
<td>22 $e_b(n</td>
<td>n) = z_b(n) - b_T^T(n)z_N(n)$</td>
</tr>
<tr>
<td>23 $\gamma(n) = 1 - g_T^T(n)z_N(n)$</td>
<td></td>
</tr>
<tr>
<td>24 $\gamma_+(n) = 1 - g_T^T(n)z_{N+2}(n)$</td>
<td></td>
</tr>
</tbody>
</table>
The scalar prediction error now can be described in vector notation as

\[ e(n|n) = d(n) - \hat{d}(n|n) = d(n) - w_N^T(n)z_N(n) \] (4.3)

where both data channels are collected into the \((N \times 1)\) vector

\[ z_N(n) = \begin{bmatrix} x_L(n) \\ d_M(n-1) \end{bmatrix}, \quad L + M = N, \] (4.4)

and the \((N \times 1)\) joint process prediction filter \(w_N(n)\) is partitioned into two portions of lengths \(L\) and \(M\) that operate on the input and output processes, respectively.

The prediction errors can be expressed as a vector which accumulates the scalar error at each time increment:

\[ e(n|n) = d(n) - \hat{d}(n|n) = d(n) - Z_0(n)w_N(n) \] (4.5)

where now an \((n \times N)\) matrix of input data is created

\[ Z_0(n) = [X_0(n), D_1(n)] \]

\[ = [x(n), z^{-1}x(n), \ldots, z^{-L+1}x(n), z^{-1}d(n), \ldots, z^{-M}d(n)] \]

\[ = \begin{bmatrix} x(1) & 0 & \ldots & 0 & | & 0 & 0 & \ldots & 0 \\ x(2) & x(1) & \ldots & | & d(1) & 0 & \ldots & \cdot \\ x(3) & x(2) & \ldots & | & d(2) & d(1) & \ldots & \cdot \\ \cdot & \cdot & \ldots & x(1) & | & \cdot & \cdot & d(1) \\ \cdot & \cdot & \ldots & \cdot & | & \cdot & \cdot & \cdot \\ x(n) & x(n-1) & \ldots & x(n-L+1) & | & d(n-1) & d(n-2) & \ldots & d(n-M) \end{bmatrix} \]

With these definitions, the standard two-channel equations apply, with computational cost \(12N + 27\) operations [33][34]. However, an important simplification is possible with this special case, which can be seen by examination of the forward
prediction filter, \( f_N(n) \). This filter vector contains two length \( N \) columns, each corresponding to separate forward prediction filters for one of the input channels. For general input channel \( x(n) \) and \( y(n) \), the two columns of the filter are designated:

\[
(N \times 2) \quad f_N(n) = [f^x_N(n), f^y_N(n)]
\]  \hspace{1cm} (4.6)

or in this ARMA modeling case, where \( y(n) = z^{-1}d(n) \),

\[
(N \times 2) \quad f_N(n) = [f^x_N(n), f^d_N(n)].
\]  \hspace{1cm} (4.7)

Similarly, the forward prediction error vector containing the scalar error associated with each subfilter is expressed:

\[
(2 \times 1) \quad e_f(n|n) = [e^x_f(n|n), e^d_f(n|n)]^T
\]  \hspace{1cm} (4.8)

From Table 4.1.13, the forward prediction filter is given as

\[ f_N(n) = K_1(n)z(n) = K_1(n)[x(n), z^{-1}d(n)], \]

and thus

\[ f^d_N(n) = K_1(n)z^{-1}d(n). \]  \hspace{1cm} (4.9)

However, from Table 4.1.9

\[ Z_1(n) = \begin{bmatrix} 0^T \\ Z_0(n-1) \end{bmatrix}. \]  \hspace{1cm} (4.10)

By direct substitution of (4.10) into the definition for the transversal filter operator, (4.1.11), it is easily shown that

\[ K_1(n) = [0, K_0(n-1)] \]  \hspace{1cm} (4.11)

which relates operators for time delayed signals. Substituting this identity into (4.9) gives
\[ f_d^N(n) = [0, K_0(n-1)] \begin{bmatrix} 0 \\ d(n-1) \end{bmatrix} = K_0(n-1)d(n-1), \] (4.12)

and, using definition (4.1.12)

\[ f_d^N(n) = w_N(n-1). \] (4.13)

Thus, it is shown that the optimal LS forward predictor for the delayed output signal is the same as the optimal joint process predictor from the previous iteration!

This is proven since the subspace of projection, \( Z_0(n-1) \), is the same in both cases. This is intuitively clear if it is realized that the joint process filter is a forward predictor of the desired signal \( d(n) \). The algorithm internally also requires a forward prediction of each input channel.

Applying the identity of (4.13), a similar connection is readily shown between the forward prediction errors associated with the respective filters:

\[
e_f^d(n | n) = e(n-1 | n-1),
\]
\[
e_f^d(n | n-1) = e(n-1 | n-2). \] (4.14a,b)

The relationships of (4.13) and (4.14) create a significant reduction in the number of computations required to implement the fast transversal ARMA predictor, compared to the standard 2-channel joint predictor case. The fact that several necessary quantities are available from the preceding iteration allows a \( 2N + 1 \) reduction in operations. The savings result since one half of the \((N \times 2)\) forward prediction filter no longer needs to be computed; also one element of the \((2 \times 1)\) forward prediction error, normally computed by inner product, is also directly available. The resulting algorithm is summarized in Table 4.2, and the new relation-
### Table 4.2: General Order, ARMA Transversal RLS Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>1x1</td>
<td>N</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>1b</td>
<td>2x1</td>
<td>0(1)</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>2a</td>
<td>1x1</td>
<td>1</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>2b</td>
<td>2x1</td>
<td>0(1)</td>
<td>$e_f(n</td>
</tr>
<tr>
<td>3a</td>
<td>2x1</td>
<td>4</td>
<td>$a(n) = e_f^{-1}(n-1)e_f(n</td>
</tr>
<tr>
<td>3b</td>
<td>1x1</td>
<td>3</td>
<td>$\gamma_+(n) = \gamma(n-1)[1 + e_f^T(n</td>
</tr>
<tr>
<td>4</td>
<td>2x2</td>
<td>5(2)</td>
<td>$e_f^{-1}(n) = e_f^{-1}(n-1) - \gamma_+(n)a(n)a^T(n)$</td>
</tr>
<tr>
<td>5a</td>
<td>N+2x1</td>
<td>2N</td>
<td>$c_{N+2}(n) = \begin{bmatrix} 0 \ c_N(n-1) \end{bmatrix} + \begin{bmatrix} I_2 \ -f_N(n-1) \end{bmatrix} a(n)$</td>
</tr>
<tr>
<td>5b</td>
<td>N+2x1</td>
<td>0(3)</td>
<td>$c_{N+2}^b(n) = S_S^Tc_{N+2}^f(n)$</td>
</tr>
<tr>
<td>5c</td>
<td>2x1</td>
<td>0(3)</td>
<td>$c_+^b(n)$: last 2 elements in $c_{N+2}^b(n)$ vector</td>
</tr>
<tr>
<td>6a</td>
<td>Nx1</td>
<td>N</td>
<td>$f_N^<em>(n) = f_N^</em>(n-1) + c_N(n-1)e_f^*(n</td>
</tr>
<tr>
<td>6b</td>
<td>Nx2</td>
<td>0(1)</td>
<td>$f_N(n) = [f_N^*(n), w_N(n-1)]$</td>
</tr>
<tr>
<td>7</td>
<td>2x1</td>
<td>4</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>8</td>
<td>1x1</td>
<td>4</td>
<td>$\gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b^T(n</td>
</tr>
<tr>
<td>9</td>
<td>2x1</td>
<td>2</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>10</td>
<td>2x2</td>
<td>3(2)</td>
<td>$e_b(n) = e_b(n-1) + e_b(n</td>
</tr>
<tr>
<td>11</td>
<td>Nx1</td>
<td>2N</td>
<td>$\begin{bmatrix} c_N(n) \ 0 \end{bmatrix} = c_{N+2}^b(n) - \begin{bmatrix} -b_N(n-1) \ +I_2 \end{bmatrix} c_+^b(n)$</td>
</tr>
<tr>
<td>12</td>
<td>Nx2</td>
<td>2N</td>
<td>$b_N(n) = b_N(n-1) + c_N(n)e_b^T(n</td>
</tr>
<tr>
<td>13</td>
<td>1x1</td>
<td>N</td>
<td>$e(n</td>
</tr>
<tr>
<td>14</td>
<td>1x1</td>
<td>1</td>
<td>$e(n</td>
</tr>
<tr>
<td>15</td>
<td>Nx1</td>
<td>N</td>
<td>$w_N(n) = w_N(n-1) + c_N(n)e(n</td>
</tr>
</tbody>
</table>
| Total OPS | | 10N +27 | (1) partially available from previous iteration  
(2) matrix symmetry allows reduction in OPS.  
(3) vector permutation is predetermined. |
ships are featured in equations (4.2.1a, 2a, 6a). An operations count is provided for each step.

4.3. Additional Observations

Although this discussion was couched in terms of ARMA system identification, the results can be applied as well to other specialized two-channel applications, including training in adaptive equalization, and data-directed echo cancellation and equalization. These applications all utilize an additional transversal predictor for the "desired" or reference signal. It is important to note that the new relationships are not approximations. Instead, they are exact equations, based upon a subspace of projection.

The computational savings is available with any of the order $N$ fast algorithms in this application, including the FTF, FAEST, and Fast Kalman algorithms. All these algorithms use forward predictors of the input signals, which have been partially computed in the joint process update.

Note that the $2N+1$ savings demonstrated in this analysis is the same number of operations required to update the joint process equations, (4.2.13, 14, 15). This is not coincidental, since the original two-channel joint process predictor has now been modified into a form more nearly equivalent to a two-channel linear predictor. If it is desired to specifically eliminate the joint process equations, the analysis has shown that the linear prediction recursions alone are valid, but the availability of the most current joint process predictor $w_N(n)$ is delayed by one time
increment, i.e., at time $n$, the LS filter $w_N(n-1)$ is computed. This filter is contained in the second column of the updated forward prediction filter, $f_N(n)$.

In many applications, this loss of information presents no real problem. Since the LS algorithm converges very quickly, the time-delayed filter is soon very close to the possible current filter. Also, often the forward prediction error is the only output required from the algorithm, rather than actual filter parameters. Such applications would include equalization and echo cancellation, illustrated in Figure 4.3.

Alternatively, suppose the analysis problem is modified such that the input vector $z(n)$ is now defined as $[z^{-1}x(n), z^{-1}d(n)]$. This case is identical to that described in [22] for adaptive equalization with training. With this signal definition, the present value of the input signal, $x(n)$ is not considered available to the joint process estimation filter. The problem has thus been converted into a two-channel linear predictor. The derivation of the FTF equations proceeds in exactly the same way with the new input vector definition, except that now the LS joint process filter must be defined $w_N(n) = K_1(n)d(n)$. Compare to (4.1.12). Since the forward prediction filter is defined $f_N(n) = K_1(n)z(n)$, with the previous definition of the input vector $z(n)$, the LS joint prediction filter is seen to be contained in the second column of the forward prediction filter. Thus, the joint process filter would (in theory) operate upon the unit-delayed $x(n)$ signal as well as the unit-delayed $d(n)$ signal. Of course, the joint process equations are totally
Two-Channel Equalization

Two-Channel Echo Cancellation

Figure 4.3 Two-Channel Applications a) Equalization b) Echo Cancellation
superfluous in this case: the linear prediction equations alone are sufficient. This fact was not observed in [22], and unnecessarily inflated the number of update computations specified.

4.4. Filter Modeling Simulations

In this section simulation results for the Fast Pole-Zero Recursive Least Squares algorithm are presented. The algorithm was programmed in the C language. To test the algorithm, the poles and zeroes of a 10th-order bandpass digital filter at a sampling rate of 32 kHz were estimated. The filter consisted of a 6th-order lowpass Elliptic IIR filter with a cutoff frequency of 3.4 kHz, in cascade with a 4th-order highpass Chebyshev IIR filter with a 200 Hz cutoff frequency. The zero and pole locations for the cascaded filter are shown in Table 4.3. The pole-zero distribution of the filter and its frequency response and impulse response are plotted in Figure 4.4. The input to the filter was a white Gaussian noise sequence with variance 1.0. Figure 4.5a shows the pole and zero estimates obtained by the algorithm after 200 iterations, with the zero and pole orders each specified at 10 ($L = 11, M = 10$). Note the accurate estimation of the the lowpass Elliptic filter poles and zeroes. The algorithm manages to estimate the 4th-order highpass section with a 2nd-order system. The algorithm then effectively cancels the remaining two pole-zero pairs by overlapping them.
Table 4.3
Zero and Pole Locations for Test Bandpass Filter

<table>
<thead>
<tr>
<th>Filter</th>
<th>Zeroes</th>
<th>Poles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real</td>
<td>Imaginary</td>
</tr>
<tr>
<td>Lowpass Section</td>
<td>-0.2096841</td>
<td>± 0.9777766</td>
</tr>
<tr>
<td></td>
<td>0.5805129</td>
<td>± 0.8142516</td>
</tr>
<tr>
<td></td>
<td>0.6636783</td>
<td>± 0.7480187</td>
</tr>
<tr>
<td>Highpass Section</td>
<td>1.0000000</td>
<td>0.000000*</td>
</tr>
<tr>
<td></td>
<td>*4th-order real zero</td>
<td>0.9601757</td>
</tr>
</tbody>
</table>
Figure 4.4. Test Filter: a) pole-zero plot  b) frequency response  c) impulse response
Figure 4.5. Estimated Filter (L=11, M=10): a) pole-zero plot  b) error energy

Figure 4.5b shows the instantaneous energy of the joint process prediction error expressed in dB, i.e., $10 \log_{10} e^2(n|n)$, versus the number of iterations. For comparison the desired response signal is similarly plotted in dB. Note the rapid convergence of the algorithm in approximately 40 iterations, which is twice the number of poles and zeroes. It can be observed that the prediction error is more than 100 dB below the desired signal, indicating the high degree of accuracy achievable by pole-zero modeling compared to impulse response modeling.

The effect of zero over-specification is presented in Figure 4.6. In this case the number of zeroes was over-specified by 10 ($L = 21, M = 10$). The estimated zeroes are distributed around the origin and two of them cancel two of the poles. The rapid convergence and high degree of prediction accuracy is again observed in Fig-
Figure 4.6. Estimated Filter (L=21, M=10): a) pole-zero plot  b) error energy

Figure 4.6. The interesting case of both zero and pole over-specification (L = 21, M = 20) is presented in Figure 4.7. Notice the mutual cancellation of excess poles and zeroes. In the case of both pole and/or zero overspecification, the prediction error was comparable to the case where the order is specified correctly.

The case where zeroes are over-specified and poles under-specified and the case where zeroes are under-specified and poles over-specified are presented in Figures 4.8 and 4.9 respectively. Notice that the former achieves a much lower prediction error than the latter. In either case, the prediction error is higher than when the orders are correctly specified.

An important property of the algorithm is its ability to predict the desired signal in the presence of noise. In Figure 4.10, the prediction error is plotted for
Figure 4.7. Estimated Filter ($L=21, M=20$): a) pole-zero plot  b) error energy

Figure 4.8. Estimated Filter ($L=21, M=5$): a) pole-zero plot  b) error energy
Figure 4.9. Estimated Filter (L=6, M=20): a) pole-zero plot   b) error energy

the case where white Gaussian noise with rms value 0.1 was added to the desired response signal. Notice that the prediction error is apparently uncorrelated, in contrast to the filtered and correlated desired response signal. In other words, the desired signal has been eliminated from the prediction error. In an echo cancellation application this corresponds to the elimination of echo in the presence of additive noise. The pole-zero distribution for this case is also shown in Figure 4.10. Notice that the additive noise affects the location of the poles and zeroes. An explanation of this observation follows. In an all-zero or impulse response modeling, the RLS algorithm averages out the effect of additive noise in the desired response signal. In this case, the additive noise does not produce a bias in the weight vector as long as it is white and uncorrelated. However, when pole-zero modeling, the noise-contaminated desired response signal is used to estimate the current desired
Figure 4.10. Estimated Filter, additive noise: a) pole-zero plot  b) error energy response sample. Thus the sample space used to estimate the desired response includes the additive noise, and the weight vector will be biased, even though the additive noise may be white and uncorrelated [36]. This bias was shown to have a bound determined by desired signal to additive noise ratio; in an echo cancellation application, the noise level is very low if far end transmission is disabled during adaptation.

The simulation results presented in this section confirm the theoretical results of [36][37] on the uniqueness of the Maximum Likelihood estimates of the parameters of an ARMA model. When the orders of the poles and zeroes are correctly specified, the fast pole-zero RLS algorithm gives unique estimates that correspond closely to the actual poles and zeroes of the process (see Figures 4.4, 5). When pole
and/or zero orders are over-specified, a spectral matching is still achieved, as demonstrated by the very small prediction error in Figures 4.6, 7.
CHAPTER 5
ALGORITHM EXTENSIONS

In this chapter we consider several extensions to the general order, multichannel algorithm. Algorithm versions with exponential memory, multichannel output, complex data, and normalized variables are developed. These extensions have been considered previously for the multichannel algorithm, but not for the general order case.

The exponential memory multichannel case is considered first. This had not previously been thoroughly described, when using transversal filter operators in the vector space derivation; an important operator decomposition is derived for this situation which had not previously appeared. Following a discussion of the independence of multiple outputs, it is shown that the complex data LS algorithm is a simple extension of the multichannel real algorithm. Finally, the normalized multichannel algorithm is derived in terms of the prediction filters used throughout this work, rather than the "residual" filters of previous literature. This provides both continuity in notation, and helps further describe the connection between the normalized and the standard algorithm.

5.1. Exponential Error Weighting

In an environment where the data signals are not stationary, but have statistics that are changing relatively slowly, it is possible to estimate a reasonable system model. One approach is to steadily "forget" or discount older data, in order to
produce a system estimate based on the more recent data.

The LS minimization problem considered in Chapter 3 can be extended to a more general weighted minimization

\[
\min_{w(r.t., w(n))} \sum_{i=1}^{n} \lambda(i) e^2(i|n)
\]

(5.1)

where \( \lambda(i) \) is some scalar weighting function. This weighting function can take an arbitrary form, but there are two common ones. The first defines \( \lambda(i) = 1 \) for \( n - M < i \leq n \) which represents a constant amplitude, moving window of length \( M \). This weighting leads to the so-called Sliding Window Covariance algorithm [45]. The other common weighting function is considered here, where \( \lambda(i) = \lambda^{n-i} \), and \( 0 < \lambda \leq 1 \). If \( \lambda = 1 \), this is the same prewindowed case considered earlier. When \( \lambda < 1 \), this window exponentially discounts, or "forgets" errors that occur further in the past. It will be shown that this can alternatively be considered exponential forgetting of older input data. This seemingly complicated weighting function is used because it leads to simple scalar updates, when a recursive solution to the LS problem is found.

The exponential weighting choice gives the following LS minimization:

\[
\min_{w(n)} \epsilon(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i|n)
\]

(5.2)

In vector form, the accumulated squared error can be expressed as

\[
\epsilon(n) = e^T(n|n) \Lambda(n) e(n|n)
\]

(5.3)

where the matrix \( \Lambda(n) \) is defined
\((n \times n)\) \(\Lambda(n) = diag [\lambda^{n-1}, \ldots, \lambda^1, 1]\)

Since \(\Lambda(n)\) is square and diagonal, its square root is simply defined as

\[\Lambda^{\frac{1}{2}}(n) = diag [\lambda^{\frac{1}{2}(n-1)}, \ldots, \lambda^{\frac{1}{2}}, 1]\] (5.4)

A useful decomposition is

\[\Lambda^{\frac{1}{2}}(n) = \begin{bmatrix}
\lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}(n-1) & 0 \\
0^T & 1
\end{bmatrix}\] (5.5)

The underbar notation is used to describe vectors and matrices that have been premultiplied by a \(\Lambda^{\frac{1}{2}}\) matrix; e.g. using (5.5) and the data vector \(z(n)\)

\[z(n) = \Lambda^{\frac{1}{2}}(n)z(n) = \Lambda^{\frac{1}{2}}(n) \begin{bmatrix} z(n-1) \\ z^T(n) \end{bmatrix} = \begin{bmatrix} \lambda^{\frac{1}{2}}z(n-1) \\ z^T(n) \end{bmatrix}\] (5.6)

For convenience, the modified inner product in (5.3) can be then written in terms of vectors each modified by the square root of the exponential weighting matrix.

\[e(n) = \Lambda^{\frac{1}{2}}(n)e(n \mid n) + \Lambda^{\frac{1}{2}}(n)e(n \mid n) = \Lambda^{\frac{1}{2}}(n)e(n \mid n)\] (5.7)

Thus in vector form, the modified error can be expressed

\[e(n \mid n) = d(n) - Z_0(n)w_N(n)\] (5.8)

\[Z_0(n) = \Lambda^{\frac{1}{2}}(n)Z_0(n) = \Lambda^{\frac{1}{2}}(n) \begin{bmatrix} Z_0(n-1) \\ z_N^T(n) \end{bmatrix} = \begin{bmatrix} \lambda^{\frac{1}{2}}Z_0(n-1) \\ z_N^T(n) \end{bmatrix}\] (5.9)

Here, \(Z_0(n)\) is the multichannel data matrix defined before.

The projection and transversal operators for the arbitrary modified subspace \(X\) are defined

\[P_X = X(X^T X)^{-1}X^T = \Lambda^{\frac{1}{2}}X(X^T \Lambda X)^{-1}X^T \Lambda^{\frac{1}{2}}\] (5.10)

\[K_X = (X^T X)^{-1}X^T = (X^T \Lambda X)^{-1}X^T \Lambda^{\frac{1}{2}}\] (5.11)

The forms for the general augmented-column (order) update for the projection and
transversal filter operators will not change with modified subspaces, i.e.

$$P_{X,Y} = P_X - P_X Y P_Y Y^{-1} Y^T P_X$$

(5.12)

$$K_{X,Y} = \begin{bmatrix} K_X \\ 0 \end{bmatrix} + \begin{bmatrix} -K_X Y \\ I \end{bmatrix} <Y, P_X Y Y^{-1} Y^T P_X$$

(5.13a)

$$K_{Y,X} = \begin{bmatrix} 0 \\ K_X \end{bmatrix} + \begin{bmatrix} I \\ -K_X Y \end{bmatrix} <Y, P_X Y Y^{-1} Y^T P_X.$$ (5.13b)

It is also necessary to find the operator matrix decompositions for modified subspaces augmented with the $\pi$ vector, as in (3.33, 34). Note that $\pi(n) = \pi(n)$.

For concreteness, consider the subspace $Z_0(n)$. Then with (5.9), and using the theorems of Appendix 3A, it can be shown that

$$P_{\theta,\pi}(n) = \begin{bmatrix} P_{\theta}(n-1) & 0 \\ 0^T & 1 \end{bmatrix}, \quad P_{\theta,\pi}(n) = \begin{bmatrix} P_{\theta}(n-1) & 0 \\ 0^T & 0 \end{bmatrix}$$

(5.14)

$$K_{\theta,\pi}(n) = \begin{bmatrix} \lambda^{-\frac{1}{2}} K_{\theta}(n-1) \\ -\lambda^{-\frac{1}{2}} N^T(n) K_{\theta}(n-1) \end{bmatrix}$$

(5.15)

Analogous expressions can be developed for the unit delayed subspace, $Z_1(n)$. The expression in (5.15) appears to be original. Note the appearance of $\lambda^{-\frac{1}{2}}$ in the upper left term of the TF operator decomposition; it has an important effect when prediction filters are generated. For example, as in (3.35) but with modified vectors,

$$\begin{bmatrix} \lambda^{-\frac{1}{2}} K_{\theta}(n-1) \\ 1 \end{bmatrix} \begin{bmatrix} \lambda^{\frac{1}{2}} d(n-1) \\ d(n) \end{bmatrix} = \begin{bmatrix} K_{\theta}(n) \\ 0 \end{bmatrix} d(n) - \begin{bmatrix} g_{\theta}(n) \\ -1 \end{bmatrix} \gamma^{-1}(n) \pi^T(n) P_{\theta}(n) d(n)$$

Taking just the upper partitions gives
\[ w_N(n-1) = w_N(n) - g_N(n) \gamma^{-1}(n) e(n | n) \]  

This is the same filter recursion as for the non-windowed case, since the \( \lambda \) terms cancel!

The derivation of the exponential memory, pre-windowed algorithm can proceed exactly as in section 3.3, except for the exponential modification to \( n \)-vectors and operators, as just described. As in (5.16), most of the recursions are not affected by exponential weighting. Exceptions occur in the updates of the prediction residual matrices. For example, use (5.12) with \( X = Z_i(n) \), \( Y = \pi(n) \), and pre- and postmultiply by \( z(n) \):

\[
\begin{align*}
 z^T(n) P_{1,\pi}(n) z(n) \\
= z^T(n) P_{1}(n) z(n) - z^T(n) P_{1}(n) \pi(n) [ \pi^T(n) P_{1}(n) \pi(n) ]^{-1} \pi^T(n) P_{1}(n) z(n)
\end{align*}
\]

(5.17)

But from (5.6), (5.14)

\[
 z^T(n) P_{1,\pi}(n) z(n) = [ \lambda^{\pi}_{1,\pi} (n-1), z(n) ] \begin{bmatrix} P_1(n-1) & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda^{\pi}_{1,\pi}(n-1) \\ z^T(n) \end{bmatrix}
\]

(5.18)

Therefore

\[ e_f(n) = \lambda e_f(n-1) + e_f(n | n-1) e_f^T(n | n) \]  

(5.19)

and the \( \lambda \) factor appears in the update formula. There is a similar result for the backward prediction residual matrix, \( e_b \).

Computational reductions can also be found using the methods of Chapter 3.4, except now define
\[ a(n) = \lambda^{-1} e_f^{-1}(n-1)e_f(n \mid n-1) \]  
This gives

\[ e_f^{-1}(n) = \lambda^{-1} e_f^{-1}(n-1) - \gamma_+(n)a(n)a^T(n) \]  

(5.21)

The complete algorithm is summarized in Table 5.1, for the general order, \( p \)-channel case. Compared with Table 3.3, an additional \( p^2 + 3p \) OPS are required for exponential memory.

5.2. Multichannel Output

A multiple output LS algorithm implies that there are several joint process (desired) signals that are to be estimated based on the (possibly multichannel) input signal. Each output channel can be seen to be independent of the others, from the projection definition of the least-squares solution. Since the projection operator is a function only of input data and not the joint process signal(s), each desired signal (output channel) generates a squared estimation error which is minimized independently. Now the output error vector \( e(n \mid n) \) has multiple columns, representing the error contribution from each desired signal estimate:

\[ (n \times q) \quad e(n \mid n) = [e_1(n \mid n), \ldots, e_q(n \mid n)] \]

This implies a multi-column joint process estimation filter, \( w_N(n) \). The \( q \)-output channel LS problem is expressed mathematically using the matrix \( \text{trace} \) function:

\[
\min_{w_N(n)} \text{trace} \ e^T(n \mid n)e(n \mid n) = \min_{w_N(n)} \sum_{j=1}^{q} \sum_{k=1}^{n} [e_{(j)}(k \mid n)]^2
\]

(5.22)

where \( e_{(j)}(k \mid n) \) is the error associated with output \( j \) at time \( k \) due to filter \( w_N(n) \).

The joint process update equations for a \( q \) output channel system are
Table 5.1
The Exponential Memory General Order, $p$-Channel RLS Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$p \times 1$</td>
<td>$pN$</td>
<td>$f_f(n</td>
</tr>
<tr>
<td>2</td>
<td>$p \times 1$</td>
<td>$p$</td>
<td>$f_f(n</td>
</tr>
<tr>
<td>3a</td>
<td>$p \times 1$</td>
<td>$p^2 + p$</td>
<td>$a(n) = \lambda^{-1}f_f^{-1}(n-1)f_f(n</td>
</tr>
<tr>
<td>3b</td>
<td>$1 \times 1$</td>
<td>$p + 1$</td>
<td>$\gamma_+(n) = \gamma(n-1)[1 + f_f^T(n</td>
</tr>
<tr>
<td>4</td>
<td>$p \times p$</td>
<td>$p^2 + 2p$</td>
<td>$f_f^{-1}(n) = \lambda^{-1}f_f^{-1}(n-1) - \gamma_+(n)a(n)a^T(n)$</td>
</tr>
<tr>
<td>5a</td>
<td>$N+p \times 1$</td>
<td>$pN$</td>
<td>$c_{N+p}^f(n) = \begin{bmatrix} 0_p \ c_N(n-1) \end{bmatrix} + \begin{bmatrix} I_p \ -f_N(n-1) \end{bmatrix}a(n)$</td>
</tr>
<tr>
<td>5b</td>
<td>$N+p \times 1$</td>
<td>0</td>
<td>$c_{N+p}^b(n) = S_bS_f^Tc_{N+p}^f(n)$</td>
</tr>
<tr>
<td>5c</td>
<td>$p \times 1$</td>
<td>0</td>
<td>$c_+^b(n) = \text{last } p \text{ elements in } c_{N+p}^b(n) \text{ vector}$</td>
</tr>
<tr>
<td>6</td>
<td>$N \times p$</td>
<td>$pN$</td>
<td>$f_N(n) = f_N(n-1) + c_N(n-1)f_f^T(n</td>
</tr>
<tr>
<td>7</td>
<td>$p \times 1$</td>
<td>$p^2 + p$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>8</td>
<td>$1 \times 1$</td>
<td>$p + 2$</td>
<td>$\gamma(n) = \gamma_+(n)[1 - \gamma_+(n)e_b^T(n</td>
</tr>
<tr>
<td>9</td>
<td>$p \times 1$</td>
<td>$p$</td>
<td>$e_b(n</td>
</tr>
<tr>
<td>10</td>
<td>$p \times p$</td>
<td>$p^2 + p$</td>
<td>$e_b(n) = \lambda e_b(n-1) + e_b(n</td>
</tr>
<tr>
<td>11</td>
<td>$N \times 1$</td>
<td>$pN$</td>
<td>$\begin{bmatrix} c_N(n) \ 0_p \end{bmatrix} = c_{N+p}^b(n) - \begin{bmatrix} -b_N(n-1) \ I_p \end{bmatrix}c_+^b(n)$</td>
</tr>
<tr>
<td>12</td>
<td>$N \times p$</td>
<td>$pN$</td>
<td>$b_N(n) = b_N(n-1) + c_N(n)e_b^T(n</td>
</tr>
<tr>
<td>13</td>
<td>$1 \times 1$</td>
<td>$N$</td>
<td>$e(n</td>
</tr>
<tr>
<td>14</td>
<td>$1 \times 1$</td>
<td>1</td>
<td>$e(n</td>
</tr>
<tr>
<td>15</td>
<td>$N \times 1$</td>
<td>$N$</td>
<td>$w_N(n) = w_N(n-1) + c_N(n)e(n</td>
</tr>
</tbody>
</table>

Total OPS. = $(5p^2 + 2)N + 4p^2 + 9p + 4 = 12N + 38$ (for $p = 2$)
\[(q \times 1) \quad e(n \mid n - 1) = d(n) - w_N^T(n - 1)z_N(n) \] (5.23a)

\[(q \times 1) \quad e(n \mid n) = \gamma(n) e(n \mid n - 1) \] (5.23b)

\[(N \times q) \quad w_N(n) = w_N(n - 1) + c_N(n) e(n \mid n) \] (5.23c)

where \(N\) is the total number of taps in all input channel filters.

5.3. Complex Data

The LS problem for complex signals can be expressed mathematically in terms of the complex estimation error:

\[
\min_{w_N(n)} \text{trace } \hat{e}^H(n \mid n)\hat{e}(n \mid n)
\] (5.24)

where \(H\) represents Hermitian (conjugate) transpose. However, from a practical (implementation) point of view, it is simpler to consider that each complex signal consists of two independent real signals, since it can assumed that physically, the complex signal is separated into its real and imaginary parts, via demodulation, etc. That is, given a complex input signal \(\{\hat{x}(n)\}, \hat{x}(n) = x_r(n) + jx_i(n)\), two real data signals, \(\{x_r(n)\}\) and \(\{x_i(n)\}\), can be extracted. The complex data LS algorithm for the case of a single input channel is thus easily recognized to correspond to a two-channel input, two-channel output algorithm for real data signals. This simplifying viewpoint is often neglected. The independence of the output channels still applies, as discussed in the previous section.

It does not make much sense to consider the "general-order" option for the two components of a single complex input signal, since both components travel through the same channel. Of course, for multiple complex input channels, the
general-order option is still applicable to the various pairs of real signals generated.

5.4. The Normalized General Order Algorithm‡

One goal of normalization is to achieve bounded coefficients and variables in algorithm computation. This is important for implementation by fixed point, dedicated processors. The normalizing process preserves the dynamic range of algorithm internal variables, and thus available bits may be used efficiently. Normalization has also been shown to provide numerical properties superior to the standard algorithm. This is important in practical applications to delay instability in the fast RLS algorithm.

Figure 5.1 shows the structure of all known fast LS transversal algorithms. The linear prediction kernel operates purely on the input signal to recursively compute forward, backward, and gain prediction filters. These filters are then used to generate the filter $w_N(n)$, which predicts the joint process signal(s). The linear prediction kernel operates independently of the joint process computations. It can be normalized, and still provide the information necessary to generate the joint process filter.

Normalization of RLS algorithms was originally demonstrated for the fast lattice algorithm [40,42], and later for the fast transversal algorithm [32,43]. That work included the multichannel development but did not consider the general-

‡ This section, with certain modifications, appeared in [44].
Figure 5.1. Structure of Fast Transversal LS Algorithms
order multichannel transversal RLS algorithm, nor the ARMA modeling application. These topics are considered here.

5.4.1. Normalization in the vector space. A normalized vector is one which has a norm or "length" of one. An arbitrary vector can be normalized to unit length simply by finding its norm and scaling; the direction of the normalized vector is not changed. Let $\mathbf{A}$ represent a generalized column vector (vector aggregate or matrix); then the normalized vector is defined

$$\tilde{\mathbf{A}} \doteq \mathbf{A} \mathbf{A}^\top \mathbf{A}^{-T/2} = \mathbf{A} (\mathbf{A}^\top \mathbf{A})^{-T/2}$$

which also introduces the matrix square root \cite{40}, $\mathbf{A}^{1/2}$, of a positive definite matrix $\mathbf{A}$, as the lower triangular matrix satisfying

$$\mathbf{A} = \mathbf{A}^{1/2} \mathbf{A}^{T/2} = \mathbf{A}^{1/2} (\mathbf{A}^{1/2})^\top$$

with

$$\mathbf{A}^{-1/2} = (\mathbf{A}^{1/2})^{-1}, \quad \mathbf{A}^{-T/2} = (\mathbf{A}^{-1/2})^\top$$

5.4.2. Normalized Operator Updates The fast RLS algorithms consist of a linear prediction kernel and an external joint process prediction. The linear prediction portion computes a gain vector (the Kalman gain), based upon forward and backward predictions of the input signal \{z(n)\}. It is possible to generate a normalized linear prediction kernel, where the original internal transversal filters are modified based on the variance of their outputs. The method of projection onto a subspace determined by the input data is used to find the LS estimators. Thus, it is essential to define normalized operators, which are modifications of the standard
projection and transversal filter operators used in the previous chapters. These were also described in [42,32]. Define an operator that expresses the inner product of two residual vectors, each normalized modulo the orthogonal subspace.

\[
\rho_X(U,V) \triangleq (U^T P_X U)^{-1/2} U^T P_X V (V^T P_X V)^{-T/2} = U^T P_X V
\]  

(5.25)

Using the expansion for projection operators, it can also be shown that

\[
(U^T P_{X,Y} U)^{-1/2} = [I - \rho_X(U,Y)\rho_X(Y,U)]^{-1/2}(U^T P_X U)^{-1/2}
\]  

(5.26)

These relations then give [42]

\[
\rho_{X,Y}(U,V) = [I - \rho_X(U,Y)\rho_X(Y,U)]^{-1/2} [\rho_X(U,V) - \rho_X(U,Y)\rho_X(Y,V)]
\]

\[
\cdot [I - \rho_X(V,Y)\rho_X(Y,V)]^{-T/2}
\]  

(5.27)

Define a normalized filter generation operator

\[
\kappa_X(U) \triangleq K_X(U)(U^T P_X U)^{-T/2} = K_X(U)
\]  

(5.28)

The following decompositions can then be shown:

\[
\kappa_{X,Y}(U) = \begin{bmatrix} \kappa_X(U) & -\kappa_X(Y) \\ 0 & (Y^T P_X Y)^{-T/2} \end{bmatrix} \rho_X(Y,U) \begin{bmatrix} [I - \rho_X(U,Y)\rho_X(Y,U)]^{-T/2} \\ 0 \end{bmatrix}
\]  

(5.29a)

\[
\kappa_{Y,X}(U) = \begin{bmatrix} 0 & (Y^T P_X Y)^{-T/2} \\ \kappa_X(U) & -\kappa_X(Y) \end{bmatrix} \rho_X(Y,U) \begin{bmatrix} [I - \rho_X(U,Y)\rho_X(Y,U)]^{-T/2} \\ 0 \end{bmatrix}
\]  

(5.29b)

5.4.3. Normalized Filter and Prediction Error Definitions Using the expansions of the previous section, various definitions can be generated.

Normalized filters:

\[
\bar{f}_N(n) \triangleq \kappa_1(z) = f_N(n)\epsilon_f^{-\tau^2}(n)
\]  

(5.30a)
\[ B_N(n) \triangleq \kappa_0(z_b) = b_N(n)e_b^{\gamma^{-2}(n)} \]  
\[ \bar{g}_N(n) \triangleq \kappa_0(\pi) = g_N(n)\gamma^{-\pi(n)} \]  
\[ \bar{g}_N(n-1) = \kappa_1(\pi) = g_N(n-1)\gamma^{-\pi(n-1)} \]  
\[ \bar{g}_{N+2}(n) \triangleq \kappa_+(\pi) = g_{N+2}(n)\gamma^{+\pi(n)} \]  
\[ \bar{g}_{N+2}(n) = \kappa_{z,1}(\pi) = g_{N+2}(n)\gamma^{+\pi(n)} \]  
\[ \bar{g}_{N+2}(n) = \kappa_{0,z_b}(\pi) = g_{N+2}(n)\gamma^{-\pi(n)} \]  
\[ \bar{g}_{N+2}(n) = S_bS_f^Tg_{N+2}(n) \] 

Normalized prediction errors:

\[ \bar{e}_f(n|n-1) \triangleq [e_f^{-\pi}(n-1), -f_N^T(n-1)]S_fz_{N+2}(n) \]  
\[ \bar{e}_f(n|n) \triangleq \rho_1(z,\pi) = e_f^{-\pi(n)}e_f(n|n)\gamma^{-\pi(n-1)} \]
\[ = e_f^{-\pi(n)}[I_2, -f_N^T(n)]S_fz_{N+2}(n)\gamma^{-\pi(n-1)} \]
\[ = [e_f^{-\pi(n)}, -f_N^T(n)]S_fz_{N+2}(n)\gamma^{-\pi(n-1)} \]
\[ \bar{e}_b(n|n) \triangleq \rho_0(z_b,\pi) = e_b^{-\pi(n)}e_b(n|n)\gamma^{-\pi(n)} \]
\[ = [-B_N^T(n), e_b^{-\pi(n)}]S_bz_{N+2}(n)\gamma^{-\pi(n)} \]  

5.4.4. Square Root Forms The square root expressions appearing in the update equations of (5.4) are of a special form which generate either a scalar or a symmetric matrix. The scalar form for a \( p \times 1 \) vector \( x \) is defined as

\[ \mu_z \triangleq [1 - x^T x]^{\frac{1}{2}} \]

The \((p \times p)\) matrix form is defined

\[ M_z \triangleq [I_p - xx^T]^{\frac{1}{2}} = I_p - \frac{1}{1 + \mu_z} xx^T \]
\[ M_z^{-1} = [I_p - xx^T]^{-\frac{1}{2}} = I_p + \frac{1}{\mu_z(1 + \mu_z)} xx^T \]  

These original identities allow simple computation of the matrix square roots, and
can be easily verified by direct multiplication. Although based on the approach of [32], these are more efficient. The following relationships also allow simplification during the derivation.

\[ M_±^1 x = μ_±^1 x, \ x^T M_±^1 = μ_±^1 x^T \]  \hspace{0.5cm} (5.34a,b)

In this work, the vector x will represent either \( e_f(n|n) \) or \( e_b(n|n) \), which generate \( μ_f \) and \( M_f \), or \( μ_b \) and \( M_b \), respectively.

### 5.4.5. Normalized Algorithm Derivation

With these definitions, it is possible to complete the derivation of the normalized algorithm. Substituting \( X = Z_1(n) \), \( Y = π(n) \), and \( U = z(n) \) into (5.26) and (5.29a) gives respectively:

\[ e_f^{-1}(n) = M_f e_f^{-1}(n-1) \]  \hspace{0.5cm} (5.35)

\[ \bar{f}_N(n-1) = \{ \bar{f}_N(n) - \bar{g}_N(n-1) e_f^T(n|n) \} M_f^{-1} \]  \hspace{0.5cm} (5.36)

Using \( X = Z_0(n) \), \( Y = π(n) \), and \( U = z_b(n) \) in (5.26) and (5.29a) gives respectively:

\[ e_b^{-1}(n) = M_b e_b^{-1}(n-1) \]  \hspace{0.5cm} (5.37)

\[ \bar{b}_N(n-1) = \{ \bar{b}_N(n) - \bar{g}_N(n) e_b^T(n|n) \} M_b^{-1} \]  \hspace{0.5cm} (5.38)

Using (5.29a,b) with \( U = π(n) \), and \( X = Z_1(n) \), \( Y = z(n) \) or \( X = Z_0(n) \), \( Y = z_b(n) \) gives respectively:

\[ \bar{g}_N(n+2) = \begin{bmatrix} 0 \\ \bar{g}_N(n-1) \end{bmatrix} μ_f^{-1} + \begin{bmatrix} \epsilon_f^{-1}(n) \\ -\bar{f}_N(n) \end{bmatrix} \epsilon_f(n|n) μ_f^{-1} \]  \hspace{0.5cm} (5.39)

\[ \bar{g}_b(n+2) = \begin{bmatrix} \bar{g}_N(n) \\ 0 \end{bmatrix} μ_b^{-1} + \begin{bmatrix} -\bar{b}_N(n) \\ \epsilon_b^{-1}(n) \end{bmatrix} \epsilon_b(n|n) μ_b^{-1} \]  \hspace{0.5cm} (5.40)
Transpose (5.26), post-multiply by $z_N(n-1)$, and subtract from $\epsilon_f^{-u}(n-1)z(n)$:

$$\epsilon_f^{-u}(n-1)z(n) - T_N^T(n-1)z_N(n-1)$$

$$= \epsilon_f^{-u}(n-1)z(n) - M_f^{-1}T_N^T(n)z_N(n-1)$$

$$+ M_f^{-1}\tilde{\epsilon}_f(n|n)g_N^T(n-1)z_N(n-1)$$

(5.41)

$$\bar{\epsilon}_f(n|n-1) = M_f^{-1}[M_f \epsilon_f^{-u}(n-1), T_N^T(n)]S_f z_{N+2}(n)$$

$$+ M_f^{-1}\bar{\epsilon}_f(n|n)\gamma^{-u}(n-1)g_N^T(n-1)z_N(n-1)$$

$$= \gamma^u(n-1)\mu_f^{-1}\bar{\epsilon}_f(n|n)$$

$$+ \mu_f^{-1}\bar{\epsilon}_f(n|n)\{\gamma^{-u}(n-1) - \gamma^u(n-1)\}$$

$$\gamma^u(n-1)\bar{\epsilon}_f(n|n-1) = \mu_f^{-1}\bar{\epsilon}_f(n|n) \triangleq a(n)$$

(5.42)

It is easily shown that

$$\bar{\epsilon}_f^T(n|n)\bar{\epsilon}_f(n|n) = 1 - \mu_f^2$$

Thus, simple algebra gives

$$\mu_f = [1 + a^T(n)a(n)]^{-u}$$

(5.43)

$$\bar{\epsilon}_f(n|n) = \mu_f\gamma^u(n-1)\bar{\epsilon}_f(n|n-1)$$

(5.44)

From (5.40) and (5.34)

$$\tilde{g}_b^b(n) = \mu_b^{-1}e_b^{-v}(n)\bar{e}_b(n|n) = \mu_b^{-1}e_b^{-v}(n-1)M_b\bar{e}_b(n|n)$$

$$\bar{e}_b(n|n) = e_b^{-v}(n-1)\tilde{g}_b^b(n)$$

(5.45)

Transpose (5.39) and post-multiply by $S_f z_{N+2}(n)$; with algebra, one gets

$$\gamma_{+}^{-u}(n) - \gamma_{+}^u(n) = \mu_f^{-1}\gamma^{-u}(n-1) - \mu_f\gamma^u(n-1)$$

Since both unknowns are greater than zero,

$$\gamma_{+}^u(n) = \mu_f\gamma^u(n-1)$$

(5.46)

Similarly, from (5.40) and post-multiplying by $S_b z_{N+2}(n)$

$$\gamma_b^u(n) = \mu_b^{-1}\gamma_{+}^u(n)$$

(5.47)
There are two alternatives for finding recursions for $\mathcal{B}_N(n)$ and $\tilde{g}_N(n)$, depending on the desired order of execution. Both require simultaneous solution of (5.39) and (5.40). However, the preferable approach generates $M_b^{-1}$, which appears elsewhere in the algorithm; the other generates $M_b$, which does not. First solving (5.40) for $\tilde{g}_N(n)$ gives

$$\begin{bmatrix}
\tilde{g}_N(n) \\
0
\end{bmatrix} = \mu_b \tilde{g}_{N+2}(n) - \begin{bmatrix}
-\mathcal{B}_N(n) \\
\epsilon_b^{\rightarrow 2}(n)
\end{bmatrix} \tilde{e}_b(n \mid n) \tag{5.48}$$

Substituting this into (5.39) and rearranging gives

$$\begin{bmatrix}
\mathcal{B}_N(n) \\
0
\end{bmatrix} = \begin{bmatrix}
\mathcal{B}_N(n-1) \\
x x
\end{bmatrix} M_b^{-1} + \mu_b^{-1} \tilde{g}_{N+2}(n) \tilde{e}_b^T(n \mid n) \tag{5.49}$$

where "xx" is an unneeded quantity. All required recursions are now available. The complete algorithm is summarized in proper order in Table 5.2, with an operations count for each step.
Table 5.2 Normalized General Order, Two-Channel Transversal RLS Algorithm

<table>
<thead>
<tr>
<th>EQ</th>
<th>DIM</th>
<th>OPS</th>
<th>Computation</th>
</tr>
</thead>
</table>
| 1 | $2 \times 1$ | $2N + 4$ | $\bar{e}_f(n | n-1) = \epsilon_f^{-
\gamma_h(n-1)} \epsilon_f(n | n-1) - T_N(n-1)z_N(n-1)$ |
| 2a | $2 \times 1$ | 2 | $a(n) = \gamma_h(n-1) \bar{e}_f(n | n-1)$ |
| 2b | $1 \times 1$ | 3 | $\mu_f = [1 + a^T(n)a(n)]^{-\n\gamma_h}$ |
| 2c | $2 \times 1$ | 2 | $\bar{e}_f(n | n) = \mu_f a(n)$ |
| 2d | $2 \times 2$ | 5 | $M_f = I_2 - [1 + \mu_f]^{-1} \bar{e}_f(n | n) \epsilon_f^T(n | n)$ |
| 3 | $1 \times 1$ | 1 | $\gamma_+^h(n) = \mu_f \gamma_h(n-1)$ |
| 4 | $2 \times 2$ | 8 | $\epsilon_f^{-T/2}(n) = \epsilon_f^{-T/2}(n-1)M_f$ |
| 5 | $N \times 2$ | $6N + 2$ | $T_N(n) = T_N(n-1)M_f + \bar{g}_N(n-1)\bar{e}_f^T(n | n)$ |
| 6a | $N+2 \times 1$ | $3N + 4$ | $\bar{g}_N+2(n) = \mu_f^{-1}\begin{bmatrix} 0 \\ \bar{g}_N(n-1) \end{bmatrix} + \begin{bmatrix} \epsilon_f^{-T/2}(n) \\ \epsilon_f^{-T/2}(n) \end{bmatrix} a(n)$ |
| 6b | $N+2 \times 1$ | 0 | $\bar{g}_b^+(n) = S_b S_f^T \bar{f}$ |
| 6c | $2 \times 1$ | 0 | $\bar{g}_b^+(n) = \text{last 2 elements in } \bar{g}_N+2(n)$ vector |
| 7 | $2 \times 1$ | 4 | $\bar{e}_b(n | n) = \epsilon_b^{-T/2}(n-1) \bar{g}_b^+(n)$ |
| 8a | $1 \times 1$ | 3 | $\mu_b = [1 - \bar{e}_b^T(n | n) \epsilon_b(n | n)]^{-\n\gamma_b}$ |
| 8b | $2 \times 2$ | 5 | $M_b^{-1} = I_2 + [\mu_b(1+\mu_b)]^{-1} \bar{e}_b(n | n) \epsilon_b^T(n | n)$ |
| 9 | $1 \times 1$ | 1 | $\gamma_b(n) = \mu_b^{-1}\gamma_b^h(n)$ |
| 10 | $2 \times 2$ | 8 | $\epsilon_b^{-T/2}(n) = M_b^{-1}\epsilon_b^{-T/2}(n-1)$ |
| 11 | $N \times 2$ | $6N + 2$ | $\begin{bmatrix} B_N(n) \\ 0 \end{bmatrix} = \begin{bmatrix} B_N(n-1) \\ \epsilon_f^{-T/2}(n-1) \end{bmatrix} M_b^{-1} + \mu_b^{-1}\bar{g}_N+2(n)\epsilon_b^T(n | n)$ |
| 12 | $N \times 1$ | $3N$ | $\begin{bmatrix} \bar{g}_N(n) \\ 0 \end{bmatrix} = \mu_b\bar{g}_N+2(n) - \begin{bmatrix} -B_N(n) \\ \epsilon_b^{-T/2}(n) \end{bmatrix} \bar{e}_b(n | n)$ |
| 13 | $1 \times 1$ | $N$ | $e(n | n-1) = d(n) - w_N^T(n-1)z_N(n)$ |
| 14 | $N \times 1$ | $N + 1$ | $w_N(n) = w_N(n-1) + \gamma_h\bar{g}_N(n) e(n | n-1)$ |

Total OPS $22N + 55$
CHAPTER 6
APPLICATION SIMULATIONS

6.1. Introduction

In this chapter, the performance of the general order multichannel algorithm is examined via simulation. The simulations are performed in the context of a full-duplex baseband data transmission system. The general order, multichannel LS algorithms developed in the previous chapters are used to implement an adaptive equalizer and an echo canceller, both important components of such data transmission systems.

The simulations performed here are all supported by the BLOSIM (Block Simulator) program [51]. This program was originally developed at the University of California. It has undergone extensive improvement and debugging since its import to NCSU, and now has a graphical interface called CAPSIM (Capture and Simulate) [55]. The most important features of this program are a hierarchical module connection structure, and the capability to run from compiled code after system definition. This second feature allows fairly rapid and efficient simulations. The hierarchy of modules allows systematic definition and reuse of user created nodes (like subroutines) called 'stars'. These stars are connected into higher level modules called 'galaxies', via a topology definition that describes their internal flow of data. Data flow between modules is moderated by data buffers that are maintained by the BLOSIM program, and are invisible to the user. They allow
synchronous, asynchronous or multirate data flow between modules.

Star modules are user defined in C-language code, and have carefully specified boundary requirements, enforced by the control program; this guarantees transportability between users. In the course of preparing these simulations, many stars have been written and stored in an available library. Some of these are discussed functionally in the following. The developed code for some stars is listed in Appendix 9. Galaxy topologies are diagrammed for various experiment descriptions. Some of these are detailed in Appendix 9.

The physical characteristics used in the simulations are intended to model the Integrated Services Digital Network (ISDN) [49,50]. This protocol, as yet not fully standardized, comprises high bit rate \((F_b = 192\text{kBps})\) baseband data transmission over unconditioned, twisted-pair telephone subscriber lines. The data rate is the aggregate of two B (64kBps) data channels, one D (16kBPS) data channel, and a control signalling channel, using the nomenclature of the North American T1-rate standard. A typical subscriber line is modeled. Impairments added to the model include additive white Gaussian noise, an unbalanced matching hybrid, and bridged taps on the line. The top-level topology model for the simulations is shown in Figure 6.1. The transmitter, channel and receiver blocks are discussed in some detail next.
6.2. The Transmitter

In full-duplex transmission, there is a data transmitter at both ends of the channel operating simultaneously. A data transmitter consists of several elements, which are modeled in these experiments as a binary data generator, a line coder, and a Nyquist-type line conditioner. The transmitter module topology "tx" is shown in Figure 6.2, and the individual component stars are discussed below.

6.2.1. Binary Data Generation. These simulations are exclusively concerned with the physical layer of the data transmission system. Higher level control, packetizing, error correction, etc. would occur at a higher protocol layer; hence, only the bulk rate of bit transmission is important. For these experiments, a random sequence of bits is generated, which corresponds to the randomizing
(scrambling) usually done to remove any correlations or patterns in the data stream. The "bdata" block models a pseudo-random binary data generator with selectable seed. This allows different patterns to be transmitted from each end.

6.2.2. Line Coding. Line coding is used to convert the binary sequence supplied by the data generator into a form suitable for physical transmission. This consists of assigning symbols to each bit or groups of bits, and then interpolating and bandlimiting to create an analog waveform. The symbol assignment phase is handled in one block; bandlimiting is performed in a separate block discussed in the next section. Many types of linecoding schemes can be used, the most obvious being to assign the symbols 0/1 to the binary bits. However, symbols are normally chosen to be symmetrically spaced about zero, in order to achieve DC balance on
the physical line. The particular line-code to be used throughout these experi-
ments is known as 2B1Q, which is currently favored as the standard for ISDN. In
2B1Q coding, consecutive pairs of bits are combined to define one of four
equispaced levels, symmetric about zero, represented with the symbols -3/-1/1/3.
This code has the advantage of reducing the system symbol rate to
\[ F_s = \frac{1}{2} F_b = 96 \text{kHz} \]
Additionally, since the number of possible symbols is not
large, noise immunity at the receiver is not compromised.

The "linecode" block was designed to output the 2B1Q code (or others), given
binary input data. It also inserts a specified number of zeroes between each output
symbol, which is preliminary to the interpolation filter stage, discussed next. This
process of increasing the sampling rate above the symbol rate is called oversam-
pling, which is considered in more detail later.

6.2.3. Nyquist Filtering. Before symbols can be launched onto the channel,
they must be conditioned by frequency domain shaping. This filtering is used pri-
marily to bandlimit the spectrum of the transmitted signal, both to confine it to a
more linear portion of the channel, and so that the receiver can limit its noise
acceptance bandwidth. Additionally, if the bandlimiting filter satisfies the Nyquist
criterion [52], no intersymbol interference (ISI) is added to the signal at the deci-
sion time (the center of the symbol interval).‡

‡ ISI is primarily a linear distortion phenomenon. Linear distortion occurs whenever a transmission
channel fails to have both a constant amplitude response and a linear phase response over the bandwidth of
the transmitted signal. This is known as a dispersive channel: different frequency components of the signal are
attenuated and delayed unequally, causing a waveform distortion at the channel output. In symbol
transmission, the affect is to "smear" adjacent symbols, such that their values influence each other. If this
The Nyquist criterion stipulates that the lowpass amplitude response of the filter be anti-symmetric about the frequency \( \frac{1}{2} F_s \). All such filters have an impulse response of the \( \sin x / x \) family, which have zero crossings at the inter-symbol period. Thus, adjacent samples will not influence each others value at the center of the symbol interval. The raised cosine amplitude function, illustrated in Figure 6.3, is one function meeting this condition. The "filtnyq" module was designed using the raised cosine function to define the frequency response. It has selectable amplitude rolloff rate \( \beta, 0 < \beta \leq .5 \). The impulse response is truncated at 12 symbol intervals, and the filter introduces a causality delay of 6 symbol intervals.

Figure 6.3 Nyquist Filter Amplitude Response, Raised Cosine

too large, symbol decision errors will be made at the receiver. Some linecodes, called partial response codes, intentionally introduce a controlled amount of ISI to influence the spectral distribution of signal energy [17].
uses the FFT with overlap-save to implement linear convolution of its impulse response with the input symbols from the linecode module, to form each output sample.

Figure 6.4 shows an eye diagram‡ for a Nyquist filtered four level signal just before entering the channel. This is the output signal of the transmitter. Figure 6.4a shows the results with a rolloff factor $\beta = 0.5$, and 6.4b with $\beta = 0.3$. Sharper filter rolloff, characterized by smaller $\beta$, causes larger response between sampling times. However, note that at sampling time (time 0 in the baud interval), the waveform exactly equals some symbol value. This is the characteristic feature of a zero ISI system. For all following experiments, $\beta$ was set to 0.5.

6.2.3.1. Notes on Oversampling. It is important to understand the fundamental difference between the sequence of abstract symbols defined at sampling time and the continuous analog waveform that actually exists on the channel. Reconciling this difference is the domain of sampling theory, which theoretically allows exact conversion from one aspect to another under certain specific conditions. The sampling rate must be greater than twice the bandwidth of the analog signal to insure that the symbols contain all the information of the waveform.

Conversely, if the symbols represent a properly sampled waveform, then the analog

‡ An eye diagram is a standard representation tool for data transmission systems. It shows a overlay of many successive symbol intervals, with the center of the graph representing the receiver sample time. This is similar to an oscilloscope view, triggered at the symbol rate. The vertical separation of the waveforms about each symbol level at time 0 is called the "eye opening", and is a measure of additive noise immunity when making symbol decisions. The horizontal separation about time 0 is a measure of the timing criticality (jitter) when sampling the waveforms.
signal can be exactly recovered using an ideal \((\sin x / x)\) interpolator.

In practice, symbols are converted to an analog waveform via digital-to-analog (D/A) conversion and lowpass filtering. The analog signal travels through the channel, which distorts it. A receiver then samples this waveform and makes a decision about which symbol was transmitted. Critical elements in this process are determining when to sample (timing recovery), and removing the channel distortions (equalization). For experiments on a digital computer, where even analog signals must be represented as a series of samples, there are two major considerations:

1) an adequate sampling rate must be determined to represent the response of the channel. For these experiments, an oversampling rate (sample rate/symbol rate) of 8 was selected for the analog signal launched onto the channel. This allowed frequency characterization of the channel to 768kHz.

2) some approximation is required by causality when interpolating an analog signal from a sequence of samples, since an infinite number of preceding and following symbols are required for perfect reconstruction. For these experiments, the interpolation filter impulse response was time limited to 12 symbol intervals.

6.3. The Channel

The typical 2-wire twisted pair line, which connects virtually all telephone sets in North America, is often a combination of sections made of different wire
gauges. Additionally, many lines contain so-called bridged taps, which are simply spliced-on lengths of cable, usually with no termination. Bridged taps were often installed anticipating future expansion. They cause difficulties for high rate data transmission, since their resonance at certain frequencies causes spectral absorption nulls. Since equalization is an inverse spectrum estimation problem, spectral nulls make equalization difficult or even impossible.

The channel topology model "channel" is shown in Figure 6.5. It has blocks which perform linear convolution with stored impulse responses to simulate the transmission and echo paths. Additive white gaussian noise is used to represent the effects of sampling errors, thermal noise and crosstalk from adjacent channels.

Figure 6.5 Channel Module Topology
Note that only a portion of the full-duplex channel is simulated, since for these experiments, only receiver performance is considered. For the transmission channel, a typical line with various wire gauges and two bridged taps was simulated to find an impulse response. This was called "Line2". Another line, "Line3", with long bridged taps was simulated, to evaluate more demanding conditions. The physical models of these hypothetical lines are shown in Figure 6.6. The lines were modeled and evaluated using another simulation program called CAPNET (Capture Network). This program allows graphical input of the characteristics of transmission lines, and evaluates their characteristics versus frequency at various nodes on the line. The transmission channel impulse responses for the lines are

![Diagram of Line2 and Line3]

Figure 6.6 Transmission Channel Physical Models, a) Line2  b) Line3
shown in Figure 6.7.

The frequency response of the echo path channel, \( H_e(j\omega) \), is computed from the impedance of the test line, \( Z_t(j\omega) \), (seen by the hybrid) and the impedance of the hybrid balance network, \( Z_h(j\omega) \), by the following expression:

\[
H_e(j\omega) = \frac{Z_t(j\omega) - Z_h(j\omega)}{Z_t(j\omega) + Z_h(j\omega)} 
\]  

(6.1)

The hybrid balance network is taken from [53], and consists of the parallel combination of 143\( \Omega \), and 643\( \Omega \) + 55nF. The impulse responses of the echo channels for the two lines are shown in Figure 6.8.

6.4. The Receiver

The function of the receiver is to periodically sample the channel output, and determine which symbol was transmitted. The receiver sampling rate and phase must be determined, and the effects of the channel must be ameliorated sufficiently so that low error rate decisions can be made. Determination of sampling rate and phase (timing recovery) is itself a vast subject and will not be considered in detail here. The synchronization implicit in the transmitter is maintained at the receiver, and phase uncertainty is limited to \( \pm \frac{1}{8} \) sample (i.e., one eighth of a symbol interval).

The receiver modeled here is comprised of several elements, each a star module. See the block diagram in Figure 6.9. These include a sampler, equalizer, slicer (symbol decision element) and an echo canceller. The sampler module is a
Figure 6.7 Transmission Channel Impulse Response, a) Line2  b) Line3
Figure 6.8  Echo Channel Impulse Response, a) Line2  b) Line3
periodic demultiplexer of the sample stream from the channel. It can select one or more sample "phases", relative to the number of samples per symbol, to input to the equalizer. The function of the equalizer is to reduce channel distortion. It is discussed below. The slicer module compares the equalizer output to possible symbols and chooses the nearest as the most likely transmitted. This selected symbol is the final receiver output. If the equalizer is adaptive, the symbol is also compared with the equalizer output estimate to compute the error used to drive adaptation. The echo canceller is used to reduce the energy from locally transmitted data that enters the receiver. It is discussed below.

Figure 6.9 Receiver Module Topology
6.4.1. Equalization. Although the conditioned symbols launched onto the channel initially suffer no ISI, the non-ideal channel introduces linear and non-linear distortion which are manifest as ISI at the receiver. The function of the equalizer in the receiver is to remove as much of this ISI as possible, in order to facilitate correct decisions about the data symbol transmitted. An equalizer is essentially a filter with a frequency response that is (ideally) the inverse of the channel frequency response. Thus, a perfect equalizer creates a net distortionless channel, i.e., pure delay only. An adaptive equalizer has the ability to invert a channel that is unknown in advance. This is important when it must suffice for the unknown set of channels encountered in a dial-up telephone environment.

Equalizers can be linear, non-linear, or a combination [17, 54]. These experiments consider linear equalizers of the transversal form. Linear equalizers are unable to handle non-linear distortions; however, the multichannel implementation of the decision feedback equalizer, discussed next, can mitigate this limitation. The length (number of taps) of a transversal equalizer determines its capacity for linear distortion removal: in general, the longer filter is more effective. However, longer filters also have lengthened convergence times. Some experiments will consider this phenomenon, and also the advantages of the multichannel form in reducing the number of taps required to achieve a given level of distortion removal. Additive noise also has an effect on convergence time and ultimate distortion removal ability. This is also shown experimentally.
6.4.1.1. **Decision Feedback Equalization.** In a decision feedback equalizer (DFE), previously decoded symbols also contribute to the current symbol estimate. Past symbols are fed into another adaptive filter, whose output is summed with that of the channel data filter, to produce the ultimate symbol estimate. This was illustrated in Figure 6.9. If the feedback filter coefficients are adaptive and determined simultaneously with the forward filter coefficients, this condition matches the multichannel case derived earlier. This was also the subject of several experiments.

6.4.1.2. **Fractionally Spaced Equalization.** A fractionally spaced equalizer (FSE) is known to be effective for alleviating the problems of uncertain sampling phase (jitter) at the remote receiver [17]. In FSE, samples are collected from the channel at a rate which is some integer multiple of the symbol rate. These sampling phases represent sub-channels which are filtered separately; each filter output is summed to contribute to the symbol estimate. This is illustrated in Figure 6.10, which also shows echo cancellation for each sampling phase. It has been shown that an oversampling rate of 2 gives results that are significantly better than unity sampling; higher rates provide little addition improvement [17,50]. A rate of 2 was used for all FSE experiments described here.

FSE implies a multiple input channel configuration. In most applications, it is not possible to prefer a particular sampling phase in advance; thus, the general order option for two (or more) channels probably does not apply. However, when
Figure 6.10 Fractionally Spaced Equalizer with Echo Cancellers
FSE is combined with DFE, it is often appropriate for the decision feedback channel to have a different order. This situation is the subject of several experiments which test convergence rate and quality under several permutations of feedforward and feedback channel orders.

6.4.2. Echo Cancellation. In a full-duplex data transmission system, the two-wire channel is shared between data transmitted in both directions. Each end of the channel has a transmitter/receiver pair which are joined to the channel by a transformer called a hybrid. Even well-balanced hybrids allow a significant amount of energy from the local transmitter to leak through to the receiver. This is called near-end echo. An addition source of echo is discontinuities in the transmission path, including the remote hybrid. This is typically termed far-end echo.

The purpose of echo cancellation in a full-duplex transmission system is to create a replica of the echo and subtract it from the received data signal. An echo canceller estimates the impulse response of the (unknown) echo path, which is a function of both the hybrid and characteristics of the channel. The transmitted symbols are fed into an adaptive filter, whose output is an estimate of the echo at the receive port. The estimation error is then used to drive adaptation of the echo canceller filter. Note that the input to the echo canceller consists of the actual transmitted symbols, rather than linecoded values. The echo canceller thus also compensates for the response of the linecoder/filter.
Since the data from the remote transmitter is attenuated by the channel, the echo power to received data power ratio can average +50dB. It is normally estimated that the received signal to noise ratio must be +20dB or better to assure low-error transmission. This implies that an echo canceller must achieve echo suppression of at least 70dB.

When echo cancellers are used in conjunction with FSE, the echo must be cancelled for each of the sub-channels, since they represent different sampling phases of the echo signal. The input data for either channel equalizer are the same transmitted symbols, however. This implies a single input channel, multiple output channel predictor, as was shown in Figure 6.10.

6.5. Predictors

The equalizer and the echo canceller described above are both based on adaptive linear transversal filters. Three types of predictor modules were designed to implement these filters. The first, an LMS predictor, is an algorithm commonly used in applications today. The other two are both fast Least Squares predictors. One of these is the multichannel FTF algorithm developed in Chapter 3; the other is the normalized form, as developed in Chapter 5.4.

All three predictor modules were designed to handle general order, multichannel inputs and multichannel outputs,‡ as derived earlier in this work. For each

‡ This is straightforward in the case of the LMS algorithm, since each filter weight is essentially adjusted independently. The LMS algorithm used here was power normalized, which means that the adaptation rate constant (λ) is selected based on the input signal power and the number of filter taps. For the multichannel case, each input channel has a separate adaptation constant.
output channel, a solution filter is simultaneously found over all input channels. Although each output filter is independent, their orders are equal and predetermined as the sum of the individual orders of the input channels.

Each predictor can be arbitrarily specified in terms of the number of input and output channels, and the input channel orders. In practice, this is accomplished via a specification file that also initializes the values of each tap. A similar file is written upon experiment completion which shows the final values of all taps. This file can be used as a specification file for subsequent experiments if desired, allowing converged values to be initialized. This technique is particularly useful for testing algorithm response to initial conditions, or presetting the slower converging LMS algorithm.

6.5.1. Algorithm Implementation. The sequencing of an adaptive predictor algorithm is not always obvious in practice. The simulation program used here enforces the causality constraints caused by feedback paths, also faced when designing an actual hardware implementation. The problem can be illustrated by considering the sequence of events during each step in adaptive equalization. The equalizer output is an estimate of the transmitted symbol at the sampling time. This estimate is then compared to a set of thresholds, and the most likely (closest) transmitted symbol is selected. A problem now occurs because the adjustment algorithm for the prediction filter depends on the error between the filter estimate and the decoded output (decision) which represents the 'desired' signal. Clearly
this error cannot be available until a decision is made, which in turn requires the filter estimate output. The joint process equations, repeated here for reference, are normally written after the linear prediction kernel equations in the sequence of algorithm equations (e.g. see Table 3.3).

\[ a) \quad e(n|n-1) = d(n) - w_N^T(n-1)z_N(n) \]
\[ b) \quad e(n|n) = \gamma(n)e(n|n-1) \]
\[ c) \quad w_N(n) = w_N(n-1) + c_N(n)e(n|n) \]  

The equation for estimation error (6.2a) is composed of two terms, the decision value and the estimate generated by inner product between the filter and the input data. It is written assuming that the decision sample is already available. Thus, this equation must be 'split' during actual implementation. The estimate is first generated and output; then, the predictor module waits until a decision is made and presented to it, before an error (and concomitant filter modifications) are computed.

This sequence of events is further complicated in the BLOSIM simulation environment, where data arrival at various module ports is not guaranteed to be sample synchronous. For example, there can be a large number of channel data samples available for processing by the predictor module, yet only one or no decision samples available. To create a universally applicable predictor module, one must provide unambiguous operation for all circumstances of data availability, at both the input and desired data ports. This design goal was achieved in the implementation of the three predictor modules. Additionally, the modules are mutually
replaceable. The module programs are shown in Appendix 6A.

6.5.2. **Predictor Training.** Adaptive predictors use an error signal to drive the weight convergence process. In the full-duplex environment, both the equalizer and echo canceller application are disrupted by interfering signals added to the error signal. For the equalizer, this interference consists of the echo from the locally transmitted data. For the echo canceller, the interference results from the remotely transmitted data. This interference can cause problems with algorithm convergence time, since the error is no longer an uncorrelated signal. The problem is often handled by "training" the predictors, using a known sequence of symbol values, transmitted while the corresponding interferer is disabled. First, a known sequence of local data is transmitted, allowing the echo canceller to converge. Then a remote sequence is transmitted (with values known to the receiver) allowing the equalizer to converge. This is one advantage of the faster converging LS algorithms, since the required training sequence is shorter. A "trainer" module is shown in Figure 6.13. It primarily switches between local and remote data. It also includes an error counter, used to evaluate performance of the equalizer.

After the training period, full duplex transmission ensues. If it is assumed that the echo canceller is well adapted, the interference to the equalizer is now quite small. It can continue to adapt during data transmission, The echo canceller error is still corrupted by remotely transmitted data, however. One approach to this problem is called adaptive reference echo cancellation (AREC) [15]. This
algorithm uses an additional predictor which forms a direct model of the transmission channel. The received symbols estimates (containing few errors) are sent through this filter to generate an estimate of the received (no additive echo) signal. This estimate is subtracted from the received signal to make the error signal for the echo canceller.
6.6. The Experiments

The experiments are divided into three subgroups: equalization, echo cancellation, and combined. Equalization and echo cancellation are first considered individually, in order to isolate the contributions of the two predictors. This is actually a half-duplex problem and is considerably less demanding for adaptive predictors.

In all the following experimental descriptions, the set of parameters given in Table 6.1 are used.

6.6.1. Experiment E1. This equalization experiment summarizes, in Figure 6.11, the convergence capability of the mcftf algorithm versus sampling phase $\phi$ for two values of the algorithm initialization constant, $\delta$. It is intended to demonstrate that

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta^2$</td>
<td>variance, additive white gaussian noise</td>
</tr>
<tr>
<td>$N$</td>
<td>total number of adaptive taps</td>
</tr>
<tr>
<td>i/j/k</td>
<td>DFE taps/FE taps, phase1/FE taps, phase2</td>
</tr>
<tr>
<td>$n$</td>
<td>number of samples processed by algorithm</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>data forgetting factor</td>
</tr>
<tr>
<td>$\delta$</td>
<td>initial value of forward prediction error power</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>bulk delay of training symbols</td>
</tr>
<tr>
<td>$\phi$</td>
<td>sampling phase, relative to center of training symbol</td>
</tr>
<tr>
<td>nlms</td>
<td>power normalized LMS algorithm</td>
</tr>
<tr>
<td>mcftf</td>
<td>multichannel Fast Transversal Filter algorithm</td>
</tr>
<tr>
<td>Nmcftf</td>
<td>normalized, multichannel FTF algorithm</td>
</tr>
</tbody>
</table>
1) quality of equalization is affected by sampling phase for symbol rate equalizers.

2) quality of equalization can be affected by the initialization constant

The magnitude of the equalizer prediction error power is given in dB, with smaller error power indicating better equalization quality. The error power is automatically output by the predictor module, and is internally low-pass filtered. Each point on the curves was determined after \( n = 1000 \) samples were processed by the predictor. The known correct symbol is used at each step used to generate the error used by the predictor (training). The channel being equalized is Line2 described in a previous section. The additive noise variance is \( \eta^2 = 1e-12 \), which corresponds to an SNR at the receiver input of +99dB (extremely low noise). The
equalizer model used is described 4/20/0, which means four DFE taps, 20 forward
taps for phase1, and zero taps (not connected) for phase2.

It can be seen that in general, the quality of equalization is much higher for
$\delta = 1e - 4$, at least after only 1000 iterations. With either choice of $\delta$, equalization
is strongly affected by the choice of sampling phase, $\phi$. Note the peaks in the
graphs. The equalization quality is periodic at the symbol width. The position of
the peaks depends on the bulk delay of the channel. It is well known (and intu-
tively reasonable) that for a symbol-rate equalizer, one should sample the received
signal near the "center" of the transmitted symbol waveshape, since the transmit
shaping filter is symmetric. Note that it is purely coincidental that the symbol
center occurs near $\phi = 0$ in this experiment; Line2 happens to have a length which
delays the signal an integer number of symbol periods.

This experiment also demonstrates that it is still possible to recover the
transmitted signal, even if one samples exactly "between" transmitted symbol
centers (in this case, $\phi = \pm 0.5$), if the SNR is great enough.

6.6.2. Experiment E2. This equalization experiment shows prediction error
power versus additive noise for several values of initialization factor in Figure 6.12.
It demonstrates that

---

Sampling phase is expressed as a fraction of the width of a symbol, with phase zero representing the
center of the training symbol. For these experiments, where the oversampling rate is 8, $\phi = k/8$, $k$ integer.
Figure 6.12 Exp. E2: Convergence with Additive Noise

1) large additive noise will override the importance of the initialization factor.

2) even with extremely small additive noise, the initialization factor can not usefully be made ever smaller. In fact, experimentally it was found that extremely small values for $\delta$ caused the algorithm to become momentarily† unstable. These values were collected after 1000 samples; behavior as a function of time is considered later.

† Incipient instability of the algorithm is detected via "rescue" variable [40][32] becoming out of bounds. The algorithm is then restarted, which consists of zeroing the linear prediction kernel filters, while the joint process filter maintains its tap values. This process is flagged for the user by the prediction modules, but its effect on the prediction error and the convergence process is surprisingly minute.
6.6.3. Experiment E3. Figure 6.13 shows prediction error power versus sampling phase for two FSE configurations, with and without a DFE channel. The prediction error power is plotted over a range of training symbol delay which spans the entire forward transversal predictor, i.e., 8 taps. The experiment demonstrates that

1) fractionally spaced equalizers are insensitive to phase (between unit symbol delays).

2) forward equalizers must place the center of the detected symbol near the center of the transversal filter.

![Figure 6.13 Exp. E3: Sampling Phase Sensitivity for FSE](image-url)
3) adding DFE taps expands the range of good performance. Since DFE sections can only work with past signal values, they can only equalize post-cursor distortion. Thus, the performance improvement is toward the end of the transversal filter, where the forward equalizer is forced to equalize pre-cursor distortion.

6.6.4. Experiment E4. This experiment shows equalizer convergence versus training delay for several DFE filter configurations. Figure 6.14a shows filters with 8 total taps, and Figure 6.14b with 10 total taps. Note that these are not FSE configurations, and are thus sensitive to sampling phase. This was fixed at $\phi = 0$, which was previously found to be ideal for Line2. Thus, a single data point was collected for each integral training delay. The experiments demonstrate that

1) given a fixed allotment of filter taps, it is worthwhile to assign at least one to a DFE section.

2) larger FE filters have a wider range of excellent convergence capability. The absolute size required for the equalizer is of course, determined by the channel to be equalized, and the requirement for equalization quality, which affects transmission error rate.

6.6.5. Experiment E5. This experiment shows DFE equalizer convergence versus time, expressed in symbols estimated. The equalizer was fixed at 12 taps, arranged 4/8. The three plots in Figures 6.15a,b,c compare the mcftf and nlms
Figure 6.14 Exp. E4: Advantage of DFE Taps with Fixed Resources
algorithms, versus the level of additive noise. To show appropriate detail, the time scales span 4000, 2000, and 1000 symbols respectively. These plots were generated by a logarithmic power meter module connected to the output of the equalizer predictor. The meter was set to average 10 consecutive values to generate each output, in order to smooth the plots. The experiments demonstrate that

1) the mcftf algorithm is always superior to the nlms algorithm in convergence speed, in this configuration. The LS algorithms can converge in an amazingly short time.

2) the ultimate convergence level for the two algorithms tends to become the same. The amount of time for this to happen can be very long if the noise level is low.

3) the ultimate convergence level is compromised by the amount of additive noise (the receiver input SNR).

6.6.6. Experiment E6. This experiment shows 4/8 DFE equalizer convergence versus time, for the normalized and the standard multichannel FTF algorithms. Figure 6.16 shows comparisons for two noise conditions, and two initialization conditions. The experiments demonstrate that

1) the performance of the normalized algorithm is identical to that of the standard algorithm, if the initialization constants are chosen properly. This is theoretically correct since both are LS algorithms.
Figure 6.15  Exp. E5: Convergence of MCFTF and NLMS
2) the initialization constant for the normalized algorithm is the square root of that for the standard algorithm.
3) the normalized algorithm can provide superior convergence with an initialization constant \( \delta = 1e^{-4} \). If the standard algorithm used this value squared, it is highly unstable. This indicates that given a value for \( \delta \), the normalized algorithm is more stable over a wider range of input power than the standard algorithm. This has been noted before [42] but only for the single channel case.

6.6.7. Experiment E7. This experiment, Figure 6.17, shows 4/8 DFE mcftf equalizer convergence versus time, for Line3 compared to Line2. The additive noise is very small. Line3 is constructed to be poor, having long bridged taps. (Line2 has been featured in all previous experiments.) This experiment

![Graph showing convergence of MCFTF for Line2 and Line3](image)

**Figure 6.17 Exp. E7: Convergence of MCFTF for Line2 and Line3**
demonstrates that

1) Line3 is more difficult to equalize than Line2, as expected. The error power is about 10dB higher for Line3 after 1000 iterations. More taps are required to achieve equivalent error power.

2) the initial convergence rate is roughly the same for either line. LS algorithms have the property that convergence rate is related to filter length, rather than signal characteristics, for high SNR.

6.6.8. Experiment F1. This echo cancellation experiment summarizes, in Figure 6.18, the convergence capability of the single channel (all zero) ftf algorithm versus filter order $N$, for several values of additive white gaussian noise. The level
of echo cancellation is defined here as the ratio of prediction error power to the receiver input power. (The input does not have interference from the remote transmitter in this experiment.) Each point on the graphs represents \( n = 1000 \) iterations. The experiment demonstrates that

1) for a given additive noise level, the ultimate convergence level is limited, no matter how long the filter.

2) as the noise level is decreased, a limit to convergence capability is reached, no matter how long the filter. This is probably also an effect of computation precision. All modules in these experiments are implemented in single precision floating point.

Note the distinct downward curvature at \( N = 38 \). This occurs because the echo impulse response used in these experiments was time limited to 256 samples or 32 symbols. The nyquist shaping filter also adds a 6 symbol delay to the echo. Thus, an all-zero filter which is \( \geq 38 \) taps will completely span the echo response, allowing cancellation to the limit of additive noise. In actual practice, of course, the echo response is not time limited, and such an artifact would not appear.

6.6.9. Experiment F2. Figure 6.19 diagrams the configuration used to test pole/zero echo cancellation. The receiver input signal is input through a unit delay to a second channel of the echo canceller. Although only one output channel (per sampling phase) is diagrammed, an arbitrary number can be accommodated. Note that it is optional whether a phase will contribute an additional input
Figure 6.19 Pole-Zero Echo Canceller

channel. Figure 6.20 shows echo cancellation versus the number of feedback (pole) taps, \( L \), for several fixed filter sizes, expressed as the total number of taps available, \( N \). This is a low noise experiment. The experiment demonstrates that

1) given a fixed allotment of taps, using one to four of the taps for a pole predictor is worthwhile. Note the dramatic improvement over the all zero filters \((L = 0)\), as used in experiment F1. Apparent exceptions to this statement occur for the filters with total order \( \geq 40 \); however, there the zero portion of the filter can span the entire impulse response, as discussed earlier, and is thus misleading.

2) there is a limit to the improvement available from adding pole taps. In fact, this can be counter-productive in the fixed allotment situation.
Figure 6.20 Exp. F2: Adding Poles for Echo Cancellation

6.6.10. Experiment F3. Figure 6.21 shows mctf and nlms echo canceller convergence for 4000 iterations (symbols estimated). The equalizers were fixed at 25 taps, arranged 4/21 (4 pole taps, 21 zero taps). These plots were generated with the logarithmic power meter module which was set to average 10 consecutive values to generate each output. The power was referenced to the receiver input. The experiment demonstrates that

1) the mctf algorithm is superior to the nlms algorithm in convergence speed, in this configuration. The LS algorithm converges very rapidly in this high SNR situation.
6.6.11. **Experiment F4.** This experiment tests a new concept for implementing a pole/zero echo canceller. In the preceding experiments, the receiver input was connected (through a unit delay) to a second channel of the echo canceler to implement a pole/zero response. Although this is possible in an actual implementation, a serious problem arises after the training mode is complete. Then, the remotely transmitted signal will also pass through the pole-channel of the echo canceller filter, which will distort it. One approach to this problem is to use an inverse filter in the data path after the echo canceller to restore the received signal; however, this requires an additional filter implementation.

It is proposed here to switch the input of the pole predictor portion of the echo canceller to its own output, after convergence has occurred (after training).
This configuration is shown in Figure 6.22. This approach has the advantage of not filtering the received data, and thus no extra inverse filtering is required.

A potential problem with this idea is that now a true feedback path exists within the filter, which admits the possibility of filter instability. The criterion for stability is that the roots of the inverse filter polynomial must remain within the unit circle. In general, this is a condition that is difficult to enforce with adaptive filters, since stability testing of intermediate filters would require polynomial root computation. However, since the filter is originally converged in a non-feedback mode, such a computation would only have to be performed once just prior to switching. The filter coefficients are then frozen, and no further adaptation occurs.

Figure 6.22 Feedback Switched Pole-Zero Echo Canceller
(This is feasible for the rapidly converging LS algorithms.) Another solution is to limit the order of the pole channel to one or two, since previous experiments have shown that it is worthwhile to include one or two pole taps. Then the stability condition as a direct function of the filter coefficients is easy to describe, and simple to check.

In these simulations, however, unstable filters never occurred. This included cases with pole order $>2$, and even when algorithms were allowed to continue convergence after switching to feedback mode. Examples with a 4/21 implementation (four feedback taps) are shown in Figure 6.23, which compares the switched and unswitched algorithms. (The graphs average 100 error power samples per data point, in order to separate the signals.) The switched algorithm changes to feedback mode after 1000 samples. The two graphs show results for different additive noise levels. They demonstrate that

1) in the low noise case, some disruption to convergence level occurs after switching. This effect decreases over time.

2) in moderate noise, there is no detectable degradation due to switching.

Although the filter no longer has access to true receiver input data after switching to feedback mode, this is balanced by the fact that there is no additive noise in the filter output.

Notice that both of these simulations are perfectly stable, even after 4000 symbols; extremely long term performance is not known.
Figure 6.23 Exp. F4: Switched Pole/Zero Echo Canceller
6.6.12. Experiments G1 and G2

The concluding set of experiments combines an equalizer and echo canceller into a complete full-duplex receiver. A DFE/FS equalizer was used (4/8/8) with a switched pole-zero echo canceller (4/21). Some experimental convergence results are shown in Figures 6.24 and 6.25. Experiment G1 used low additive receiver noise, $\eta^2 = 10^{-12}$, and G2 used moderate noise, $\eta^2 = 10^{-6}$. The simulation was divided into three epochs: equalizer convergence, samples 0-999; echo canceller convergence, samples 1000-1999; and full-duplex, samples 2000-4000. During equalizer convergence, the local transmitter was off (outputting zeroes) so there would be no interfering echo signal. Similarly, during echo canceller convergence, the remote transmitter was off. At the end of convergence (sample 2000), the echo canceller was switched to feedback mode, and both predictors were prevented from further adaptation.

The two graphs in each experiment show equalizer and echo canceller error power, respectively. These powers are referenced to unity, and each point is an average of ten values.

Some observations on these results are

1) the echo canceller error is only interesting during the EC convergence epoch. At all other times, it is dominated by the level of the received signal, which is approximately -20dB. (The received signal is still far greater than the additive noise: in G1, receiver SNR = 40dB; in G2, SNR = 100dB.) The degree
Figure 6.24 Exp. G1: Full-Duplex Performance, Low Noise
Figure 6.25  Exp. G2: Full-Duplex Performance, Moderate Noise

Line2, Echo2
\( \eta^2 = 1e^{-6} \)
DFE: 4/8/8
\( \lambda = 1.00 \)
\( \Delta = 12 \)
\( \phi = 0 \)
\( \delta = 1e^{-4} \)
EC: 4/21, switched
\( \lambda = 1.00 \)
\( \phi = 0 \)
\( \delta = 1e^{-4} \)
of convergence is the same in both of these experiments, since the echo signal is far stronger than the added noise.

2) the equalizer convergence is affected by the additive noise level. In both cases, however, it is superior to the ultimate full-duplex level, and is thus adequate.

3) the level of equalizer error power, during echo canceller convergence, represents the effect of channel noise and residual echo through the converged equalizer filter. (There is no remote transmitted data during this interval.) In both experiments, this level is roughly as large as the ultimate equalizer error. Note that the equalizer must introduce approximately +20dB gain (for this line model), in order to restore the received signal. The gain is also applied to the channel noise and residual echo. This seems to determine the limitation to final receiver performance, as indicated by the equalizer error level. In both of these experiments, the -20dB equalizer error power level gave error free performance, after symbol 2000.
CHAPTER 7
CONCLUSION

This work has considered the application of fast transversal filters to the two problems of equalization and echo cancellation. The unifying classification of system identification was described and applied to these two applications. Fast LS algorithms, with a computational requirement per update that is a linear function of the filter size, were presented as a viable mechanism for system identification.

A new general order, multichannel form of the fast transversal least squares algorithm was then derived. Although a multichannel version of this algorithm was previously known, it was constrained to have equal orders in all input channels. The multichannel version presented here is completely arbitrary in the order of each input channel. As one general application, it was used with a special two channel case of system identification. This case, in which the unit delayed joint process signal is used as an additional input channel, was shown to be an ARMA system identification configuration called the equation error form. A new and significant computational savings of \((2N+1)\) operations was described and proven for this configuration. The reduction was also newly shown to be available with the other known fast recursive algorithms (FK and FAEST). This improved algorithm was then successfully applied to the identification of an IIR bandpass filter. The general order capability was not previously known using the multichannel FTF algorithm in this application.
Several extensions of the new algorithm were then derived, including exponential decaying data memory, the multichannel output case, the complex data case, and the normalized variable version. Normalized forms have been shown to provide superior numerical stability properties compared to unnormalized algorithms; this is particularly important in practical implementations with limited-precision machines.

Several application simulations for the algorithm were performed, using Integrated Services Digital Network (ISDN) parameters. The BLOSIM simulation program was used to build an end-to-end data transmission system, complete with adaptive equalizer and echo canceller implementing the new algorithms, and a large library of software modules was developed for future efforts in data transmission simulations.

The superior convergence capacity of LS algorithms over the ubiquitous LMS algorithm makes them attractive for implementing equalizers and echo cancellers. This work has also demonstrated that pole/zero modeling can reduce the model order and computational burden required to implement a fast LS algorithm, compared to currently popular all-zero models. In real-time situations, the general order capability will allow allotment of limited computational time to individual channels, particularly when implementing decision feedback equalizers and pole-zero echo cancellers.
It is well established that the fast LS algorithms have long term stability problems, and this has limited their practical application. These problems were not addressed in this work; algorithm stability was maintained through the use of rescue variables. Although some progress has been made recently, this is still an important research topic.
CHAPTER 8

REFERENCES


CHAPTER 9

APPENDIX: Simulation Modules and Topologies

This appendix gives program listings for several important BLOSIM modules used in the simulations. It also details the interconnections (topologies) used in the various experiments.
This star implements a multichannel input/output FIR predictor, which is
adapted using the least squares Fast Transversal Filter algorithm.
It can be used as an equalizer, FSE, DFE, or echo canceller.
An arbitrary number p input channels are transversal filtered
to produce an arbitrary number q output estimate signals.
Note: each output buffer connected to this star implies a separate
output channel, and identically numbered error input channel.
Input signal channels are then connected to higher numbered buffers.
It is assumed that the estimate error is computed externally.
Do NOT implement an external (causality) unit-delay from
output estimate to input error; this delay is handled automatically.

Param. 1 - Name of ASCII input specification file. Filter orders
and initial tap values are given. default => prfile

The proper specification file format is:
(int) # output channels, q
(int) # input channels, p

(int) order of in ch.#1 ... (int) order of in ch.#p

(float) ch.#1, tap 1 ... (float) ch.#1, tap last
   {output ch.1}
   ...

(float) ch.#p, tap 1 ... (float) ch.#p, tap last
   ...

(float) ch.#1, tap 1 ... (float) ch.#1, tap last
   {output ch.q}
   ...

(float) ch.#p, tap 1 ... (float) ch.#p, tap last

Param. 2 - Name of output file, for final adapted filter values.
default => prfileo. The file is written in proper
input-file format. This file can then be used to initialize
the filter for the next run, if desired.

It is assumed that each output prediction filter will create one
estimate output for EACH input sample/error sample pair.
Any decimation, etc. must occur externally.

Param. 3 - (float) lambda. data forgetting factor. default => 1.0
   Lambda = 1.0 implies no long term adaptation occurs.
4 - (float) delta. initial value, forward prediction energy.
default => 1e-4

5 - (int) wait. number of samples to skip before starting
   adaptation. The predictor still inputs samples, and
   outputs a zero estimate. default => 0
6 - (int) adapt. number of samples to adapt filter. After
   this number, filter taps are fixed, and estimates are still
   produced. default => -1 (implies always adapt)
Programmer: L.J. Faber
Date: April 1988
Modified: May 1988 add multichannel output
Modified: June 1988 estimate-referenced prediction energy
Modified: Aug 1988 est. input power. new parameter delta.
Modified: Sept 1988 add parameters 5,6 and associated.

#define Mch 10 /* maximum 10 or 0 channels */

parameters
file ifile_name = "prfile";
file ofile_name = "prfileo";
float lambda = 1.0;
float delta = 1e-4;
int wait = 0;
int adapt = -1;

end

states
int counter; /* routine entry counter */
float* inpower; /* avg. input power */
float* epower; /* avg. jp error power */
int p; /* number of input data channels */
int q; /* number of output data channels */
FILE* fp;
int* orders; /* #taps for each input channel */
int N; /* total number of taps */
float* z;
float* Z;
float* F;
float* B;
float* C;
float* Cp;
float* W;
float* Ef;
float* Eb;
float gamma;

end

declarations
int i,j,k;
char *calloc();
float error[Mch]; /* convenience counter */
int index; /* last p elements of Cp */
float cb[Mch]; /* auxilliary (temp) vector */
float a[Mch]; /* small working vector */
float temp[Mch+1]; /* working accumulator */
float sum; /* forward est. error */
float ef[Mch]; /* forward pred. error */
float efp[Mch]; /* backward est. err. */
float ebp[Mch]; /* b. prediction error */
float gmp; /* extended gain estimation error */

end

initialization_code
if(no_outputBuffers() < 1) {
    fprintf(stderr,"pred: no output data channels\n*); return(1);
}

if(no_inputBuffers() < no_outputBuffers() + 1) {
    fprintf(stderr,"pred: not enough input buffers\n*); return(2);
}

if((fp = fopen(ifile_name, "r")) == NULL) {
    fprintf(stderr,"pred: can't find filter specification file\n*); return(3);
}

fscanf(fp, "%d", &q);
fscanf(fp, "%d", &p);
if(p > Mch || q > Mch) {
    fprintf(stderr,"pred: more than %d i/o channels\n*, Mch); return(4);
}

if(q != no_outputBuffers() || p != no_inputBuffers() - q) {
    fprintf(stderr,"pred: spec file %s does not agree with topology\n*, ifile_name); return(5);
}

if( (orders = (int*)calloc(p,4)) == NULL
    || (t = (float*)calloc(p,4)) == NULL
    || (inpower = (float*)calloc(p,4)) == NULL
    || (epower = (float*)calloc(q,4)) == NULL
    || (Ef = (float*)calloc(p*p,4)) == NULL
    || (Eb = (float*)calloc(p*p,4)) == NULL) {
    fprintf(stderr,"pred: can't allocate space\n*); return(6);
}

N = 0;
for(j=0; j<p; j++) {
    if(fscanf(fp, "%d", &orders[j]) != 1) {
        fprintf(stderr,"pred: problem in input file %s\n*, ifile_name); return(7);
    }
    N += orders[j];
}

if( (W = (float*)calloc(N*q,4)) == NULL
    || (Z = (float*)calloc(N,4)) == NULL
    || (F = (float*)calloc(p*N,4)) == NULL
    || (B = (float*)calloc(p*N,4)) == NULL
    || (C = (float*)calloc(N,4)) == NULL
    || (Cp = (float*)calloc(N+p,4)) == NULL) {
    fprintf(stderr,"pred: can't allocate space\n*); return(6);
}

for(k=0; k<q; k++) {
    for(i=0; i<N; i++) {
        if(fscanf(fp, "%.f", &W[i*q+k]) != 1) {
            fprintf(stderr,"pred: problem in input file %s\n*, ifile_name); return(7);
        }
    }
}
```c
Close(fp);

/* set up unit delay in error channels */
for (k=0; k<q; k++)
    set_dmln_in(k,1);

gamma = 1.0;
for (l=0; l<p; l++)
    { 
        Efl[l*p+i] = 1./delta;
        Eb[l*p+i] = delta;
        input[l] = delta;
        for (j=0; j<l; j++)
            { 
                Efl[l*p+j] = Efl[j*p+i] = 0.0;
                Eb[l*p+j] = Eb[j*p+i] = 0.0;
            }
    }

end

main_code

if (min_avall() == 0) return(0);

if (wait > 0) {
    wait--;
    for (k=0; k<q; k++)
        { 
            it_in(k);
            it_out(k);
            output(k,0) = 0;
        }
    for (j=0; j<p; j++)
        it_in(q+j);
    return(0);
}

/**** Normal Operation ****/

/* input new data and create vector */
for (j=0; j<p; j++)
    { 
        it_in(q+j);
        z[j] = inf(q+j,0);
        /* update input power state */
        input[j] *= .96;
        input[j] += .04 * z[j] * z[j];
    }

/* input unit-delayed error(s) */
for (k=0; k<q; k++)
    { 
        it_in(k);
        error[k] = inf(k,1);
        /* update error power state */
        error[k] *= .96;
        error[k] += .04 * error[k] * error[k];
    }

if (adapt == 0) goto resume;

/******* ftf predictor ****************************/
 Joint Process

for(k=0; k<q; k++) {
    /* e(n-1-k-2) is computed externally */
    /* find e(n-1-k-1) (14) */
    /* update W(n-2) -> W(n-1) (15) */
    for(l=0; l<N; l++)
        W[l*q+k] += C[l] * error[k] * gamma;
}

/* calculate e sub f (nh-1) (1) */
/* calculate e sub f (nh) (2) */
for(j=0; j<p; j++) {
    sum = 0;
    for(i=0; i<N; i++)
        sum += F[i*p+j] * Z[i];
    efp[j] = z[j] - sum;
    ef[j] = efp[j] * gamma;
}

/* Update auxiliary vector a(n) (3a) */
for(j=0; j<p; j++) {
    sum = 0;
    for(i=0; i<N; i++)
        sum += Ef[i*p+j] * efp[i];
    a[j] = sum / lambda;
}

/* update gamma sub + (n) (3b) */
sum = 0;
for(j=0; j<p; j++)
    sum += ef[j] * a[j];
gmp = gamma / (1. + sum);

/* update epsilon sup -1 (n) (4) */
for(j=0; j<p; j++) {
    Ef[i*p+j] /= lambda;
    temp[j] = gmp * a[j];
    Ef[i*p+j] -= temp[j] * a[j];
    for(i=0; i<j; i++)
        Ef[i*p+j] -= temp[j] * a[i];
    Ef[i*p+j] = Ef[i*p+j];
}

/* update C sub N+p sup f (n) (5a) */
for(j=0; j<p; j++)
    Cp[j] = a[j];
for(l=0; l<N; l++)
    sum = 0;
    for(j=0; j<p; j++)
        sum += F[l*p+j] * a[j];
    Cp[l+p] = C[l] - sum;
}

/* form Cbp = Sb*Sf*Cfp (re-arrange Cp) (5b) */
index = p-1;
for(j=0; j<p-1; j++) {
    index += orders[j];
    temp[j] = Cp[index];
    Cp[index] = Cp[index+1];
for(i=1; i<N; i++)
\quad Cp[i] = Cp[i+p-1];
for(j=0; j<p-1; j++)
\quad Cp[N+j] = temp[j];

/* define cb sub sup b (n) */
for(j=0; j<p; j++)
\quad cb[j] = Cp[N+j];

/* update F(n) */
for(i=0; i<N; i++)
\quad for(j=0; j<p; j++)
\quad \quad F[i*p+j] += C[i] * ef[j];

/* calculate e sub b (nh) */
for(j=0; j<p; j++)
\quad eb[j] = gamma * ebp[j];

/* update epsilon sub b */
for(i=0; i<p; i++)
\quad Eb[i*p+i] *= lambda;
\quad Eb[i*p+i] += ebp[i] * eb[i];
\quad for(j=0; j<i; j++)
\quad \quad Eb[i*p+j] *= lambda;
\quad Eb[i*p+j] += ebp[i] * eb[j];
\quad Eb[j*p+i] = Eb[i*p+j];
for (i=0; i<N; i++) {
    sum = 0;
    for (j=0; j<p; j++)
        sum += B[i*p+j] * cb[j];
    C[i] = Cp[i] + sum;
}

for (i=0; i<N; i++)
    for (j=0; j<p; j++)
        B[i*p+j] += C[i] * eb[j];

/* create z sub N (n) by shifting z sub N (n-1) */
for (i=N-1; i>0; i--)
    Z[i] = Z[i-1];
index = 0;
for (j=0; j<p; j++) {
    Z[index] = z[j];
    index += orders[j];
}

for (k=0; k<q; k++)
    /\* compute estimate with old filter */
    estimate = 0;
    for (i=0; i<N; i++)
        estimate += W[i*q+k] * Z[i];

    /\* output estimate and wait for error */
    it_out(k);
    outf(k,0) = estimate;
}

counter++;
if (adapt > 0) adapt--;
return(0);

end

wrapup_code

for (j=0; j<p; j++)
    fprintf(stderr, "predItf: input power(\%d) = \%g\n", j, inpower[j]);
for (k=0; k<q; k++)
    if (epower[k] <= 0) epower[k] = 1e-30;
    fprintf(stderr, "pred error power(\%d) = \%g = \%g dB\n",
            k, epower[k], 10*log10(epower[k]));

if ((fp = fopen(ofile_name, "w")) == NULL) {
    fprintf(stderr, "pred: can't open output file\n");
    return(1);
}

fprintf(fp, "\%d\n", q);
fprintf(fp, "\%d\n", p);
for(j=0; j<p; j++)
    fprintf(fp, *%d \t*, orders[j]);
fprintf(fp, "\n");

for(k=0; k<q; k++) {
    index = 0;
    for(j=0; j<p; j++) {
        fprintf(fp, "\n");
        for(i=0; i<orders[j]; i++) {
            fprintf(fp, *%g \t*, W[index*q+k]);
            if((i+1)%5 == 0) fprintf(fp, "\n");
            index++;
        }
    }
    fprintf(fp, "\n");
}
fprintf(fp, "\n");
fclose(fp);
free(inpower);free(epower);
free(orders);free(z);free(Z);
free(F);free(B);free(C);free(W);
free(Cp);free(Ef);free(Eb);

end
This star implements a multichannel input/output FIR predictor, which uses the least squares NORMALIZED Fast Transversal Filter algorithm. It can be used as an equalizer, FSE, DFE, or echo canceller. An arbitrary number p input channels are transversal filtered to produce an arbitrary number q output estimate signals. Note: each output buffer connected to this star implies a separate output channel, and identically numbered error input channel. Input signal channels are then connected to higher numbered buffers. It is assumed that the estimate error is computed externally. Do NOT implement an external (causality) unit-delay from output estimate to input error; this delay is handled automatically.

Param. 1 - Name of ASCII input specification file. Filter orders and initial tap values are given. default => prfile

The proper specification file format is:
(int) # output channels, q
(int) # input channels, p
(int) order of in ch.#1 ... (int) order of in ch.#p
(float) ch.#1, tap 1 ... (float) ch.#1, tap last
.... {output ch.1}
....
(float) ch.#p, tap 1 ... (float) ch.#p, tap last
....
....
(float) ch.#1, tap 1 ... (float) ch.#1, tap last
.... {output ch.q}
....
(float) ch.#p, tap 1 ... (float) ch.#p, tap last

Param. 2 - Name of output file, for final adapted filter values. default => prfileo. The file is written in proper input-file format. This file can then be used to initialize the filter for the next run, if desired.

It is assumed that each output prediction filter will create one estimate output for EACH input sample/error sample pair. Any decimation, etc. must occur externally.

Param. 3 - (float) lambda. data forgetting factor. default => 1.0
Lambda = 1.0 implies no long term adaptation occurs.
4 - (float) delta. initial value, forward prediction energy. default => 1e-4

5 - (int) wait. number of samples to skip before starting adaptation. The predictor still inputs samples, and outputs a zero estimate. default => 0
6 - (int) adapt. number of samples to adapt filter. After this number, filter taps are fixed, and estimates are still produced. default => -1 (implies always adapt)
Programmer: L.J. Faber
Date: April 1988
Modified: May 1988 add multichannel output
Modified: June 1988 estimate-referenced prediction energy
Modified: Aug 1988 est. input power. new parameter delta.
Modified: Sept 1988 add parameters 5,6 and associated.
*/

#define Mch 10 /* maximum i or o channels */

parameters
file ifile_name = *prfile*;
file ofile_name = *prfileo*;
float lambda = 1.0;
float delta = 1e-4;
int wait = 0;
int adapt = -1;
end

states
int count = 0; /* star entry counter */
float* epower; /* avg. jp error power */
int p; /* number of input data channels */
int q; /* number of input data channels */
FILE* fp;
int* orders; /* size of each input channel */
int N; /* total number of taps */
float* z;
float* Z;
float* F;
float* B;
float* G;
float* T;
float* Gp;
float* W;
float* Ef[t;
float* Ebt;
float* Mf;
float* Mbi;
float* gamma;
end

declarations
int i,j,k;
float error[Mch];
int index; /* convenience counter */
float estimate;
float gb[Mch]; /* last p elements of Gp */
float mu,f, mub; /* mu sub f, mu sub b */
float s[Mch]; /* auxilliary (temp) vector */
float temp[Mch+1]; /* small working vector */
float sum; /* working accumulator */
float ef[Mch]; /* forward est. error */
float efp[Mch]; /* forward pred. error */
float eb[Mch]; /* backward est. err. */
float ebp[Mch]; /* b. prediction error */
float gmp; /* extended gain estimation error */
end
initialization_code

if(no_output_buffers() < 1) {
  fprintf(stderr,"pred: no output data channels\n");
  return(1);
}

if(no_input_buffers() < no_output_buffers() + 1) {
  fprintf(stderr,"pred: not enough input buffers\n");
  return(2);
}

if((fp = fopen(iflle_name, "r")) == NULL) {
  fprintf(stderr,"pred: can't open spec file %s\n",
          iflle_name);
  return(3);
}

fscanf(fp, "%d", &q);

fscanf(fp, "%d", &p);

if(p > Mch | q > Mch) {
  fprintf(stderr, "pred: more than %d i/o channels\n",Mch);
  return(4);
}

if( q != no_output_buffers() | p != no_input_buffers() - q ) {
  fprintf(stderr,
          "pred: spec file %s does not agree with topology\n",
          iflle_name);
  return(5);
}

if( (orders = (int*)calloc(p,4)) == NULL |
    (z = (float*)calloc(p,4)) == NULL |
    (epower = (float*)calloc(q,4)) == NULL |
    (Mf = (float*)calloc(p*p,4)) == NULL |
    (Mbi = (float*)calloc(p*p,4)) == NULL |
    (Efl = (float*)calloc(p*p,4)) == NULL |
    (Ebt = (float*)calloc(p*p,4)) == NULL | { |
    fprintf(stderr,"pred: can't allocate space\n");
    return(6);
}

N = 0;

for(j=0; j<p; j++) {
  if((fscanf(fp, "%d", &orders[j])) != 1) {
    fprintf(stderr,"pred: problem in input file %s\n",
             iflle_name);
    return(7);
  }
  N += orders[j];
}

if( (W = (float*)calloc(N*q,4)) == NULL |
    (Z = (float*)calloc(N,4)) == NULL |
    (F = (float*)calloc(p*N,4)) == NULL |
    (B = (float*)calloc(p*N,4)) == NULL |
    (T = (float*)calloc(p*N,4)) == NULL |
    (G = (float*)calloc(N,4)) == NULL |
    (Gp = (float*)calloc(N+p,4)) == NULL ) { |
  fprintf(stderr,"pred: can't allocate space\n");
  return(8);
}

for(k=0; k<q; k++) {
  for(i=0; i<N; i++) {
if((fscanf(fp, "%f", &W[i*q+k])) != 1) {
    fprintf(stderr, "pred: problem in input file %s\n",
            file_name);
    return(7);
}
}
fclose(fp);

/* set unit delay for error channels */
for(k=0; k<q; k++)
    set_dmin_ln(k, 1);

gamma = 1.0;
diagmat(Elt, 1./delta, p);
diagmat(Ebt, delta, p);

end

main_code

if(min_avail() == 0) return(0);

if(wait > 0) {
    wait--;
    for(k=0; k<q; k++) {
        it_ln(k);
        it_out(k);
        outf(k,0) = 0;
    }
    for(j=0; j<p; j++)
        it_in(q+j);
    return(0);
}

/**** Normal Operation ****/
/* input new data and create vector */
for(j=0; j<p; j++)
    it_in(q+j);
    z(j) = inf(q+j,0);

/* input unit-delayed error(s) */
for(k=0; k<q; k++)
    it_in(k);
    error[k] = inf(k,1);
    /* update error power state */
    epower[k] *= .96;
    epower[k] += .04 * error[k] * error[k];
}
if(adapt == 0) goto resume;

/******* normalized stf predictor ***********************/

/**** Joint Process ****/
for(k=0; k<q; k++)
    /* e(n-ln-2) is computed externally */
    /* find e(n-ln-1) (14) */
    /* update W(n-2) -> W(n-1) (15) */
for(i=0; i<N; i++)
    W[i*q+k] += G[i] * error[k] * gamma;
/* calculate $e_{\bar{r}}^{(n+1)}$ (1) */
/* Update auxiliary vector $a(n)$ (2a) */
multmatf(Z,F, 1,N,p, temp);
multmatf(t,Eflt, 1,p,p, efp);
for(j=0; j<p; j++) {
    efp[j] -= temp[j];
    a[j] = gamma * efp[j];
}

/* compute $\mu_f^{(n)}$ (2b) */
multmatf(a,a, 1,p,1, &sum);
muf = 1./sqrt(1.+ sum);
/* compute $e_{\bar{r}}^{(n)}$ (2c) */
for(j=0; j<p; j++)
    ef[j] = muf * a[j];
/* compute $M_f^{(n)}$ (2d) */
sum = 1.+ muf;
for(j=0; j<p; j++)
    temp[j] = -ef[j] / sum;
multmatf(temp,ef, p,1,p, Mf);
for(j=0; j<p; j++)
    Mf[j*p+j] += 1.;

/* update $\gamma^{+sup \, n/2}$ (3) */
gmp = muf * gamma;
/* update $\epsilon_f^{sup -T \, n}$ (4) */
multmatf(Eflt,Mf, p,p,p, Eflt);
/* update bold $f$ bar sub $N$ (n) (5) */
multmatf(F,Mf, N,p,p, F);
multmatf(G,ef, N,1,p, T);
addmatf(F,T, N,p, F);

/* update bold $g$ bar sub $N+p$ sup $f$ (n) (6a) */
multmatf(Eflt,a, p,p,1, Gp);
multmatf(F,a, N,p,1, T);
for(i=0; i<N; i++)
    Gp[i+p] = G[i]/muf - T[i];
/* form Gbp = $S_b^*S_f^*Gfp$ (re-arrange Gp) (6b) */
index = p-1;
for(j=0; j<p-1; j++) {
    index += orders[j];
    temp[j] = Gp[index];
    Gp[index] = Gp[j+1];
}
for(i=1; i<N; i++)
    Gp[i] = Gp[i+p-1];
for(j=0; j<p-1; j++)
    Gp[N+j] = temp[j];
/* define $g_f^{sup + \, n}$ (6c) */
for(j=0; j<p; j++)
    gb[j] = Gp[N+j];
/* calculate $e_b^{(n-1)}$ (7) */
multmatf(Ebt,gb, p,p,1, eb);
/* calculate $\mu_b^{(n)}$ (8a) */
multmatf(eb,eb, 1,p,1, &sum);
if(sum < 0 | 1.-sum <= 0) {
    /* Rescue: restart algorithm kernel */
    fprintf(stderr,"prednftf: rescue @%d\n",count);
    gamma = 1.0;
    for (i=0; i<N; i++) {
        G[i] = 0;
        for(j=0; j<p; j++)
            F[i*p+j] = B[i*p+j] = 0;
    }
    diagmatf(Ebt, 1./delta, p);
    diagmatf(Ebt, delta, p);
    goto resume;
}

mub = sqrt(1.-sum);

    /* calculate M b -1 (8b) */
    sum = mub * (1. + mub);
    for(j=0; j<p; j++)
        temp[j] = eb[j] / sum;
    multmatf(temp,eb, p, p, Mbi);
    for(j=0; j<p; j++)
        Mbi[j*p+j] += 1.;

    /* update gamma (n) (9) */
    gamma = gmp / mub;
    /* update epsilon b (n) (10) */
    multmatf(Mbi,Ebt, p,p,p, Ebt);
    multmatf(B,Mbi, N,p,p, B);
    for (j=0; j<p; j++)
        temp[j] = eb[j] / mub;
    multmatf(Gp,temp, N,l,p, T);
    addmatf(B,T, N,p, B);

    /* update g bar b N (n) (12) */
    multmatf(B,eb, N,p,1, T);
    for(i=0; i<N; i++)
        G[i] = mub * Gp[i] + T[i];

/*---------------------------------------------------------------*/
resume:

    /* create z N (n) by shifting z N (n-1) */
    for(i=N-1; i>0; i--) 
        Z[i] = Z[i-1];
    index = 0;
    for(j=0; j<p; j++)
        Z[index] = z[j];
    index += orders[j];
}

for(k=0; k<q; k++) {
    /* compute estimate with old filter */
    estimate = 0;
    for(i=0; i<N; i++)
estimate += \( W[i*q+k] * Z[i] \);

/\* output estimate and wait for error */
it_out(k);
outf(k,0) = estimate;
}
count++;
if(adapt > 0) adapt--;
end

wrapup_code

for(k=0; k < q; k++) {
    epower[k] = 1e-30;
    fprintf(stderr, "prednftf: pred error power(%d) = %#.3g dB\n", k, epower[k], 10*log10(epower[k]));
}

if((fp = fopen(ofile_name, "w")) == NULL) {
    fprintf(stderr, "can’t open output file\n");
    return(1);
}

fprintf(fp, "%d\n", q);
fprintf(fp, "%d\n", p);
for(j=0; j < p; j++)
    fprintf(fp, "%d \til, orders[j]);
    fprintf(fp, "\n");
for(k=0; k < q; k++) {
    index = 0;
    for(j=0; j < p; j++) {
        fprintf(fp, "\n");
        for(i=0; i < orders[j]; i++) {
            fprintf(fp, "%g \t", W[index*q+k]);
            if((i+1)%5 == 0) fprintf(fp, "\n");
            index++;
        }
        fprintf(fp, "\n");
    }
    fprintf(fp, "\n");
}
fclose(fp);
free(orders); free(epower); free(z); free(Z);
free(F); free(B); free(G); free(W); free(Gp);
free(Eft); free(Ebt); free(T); free(Mf); free(Mbi);
end
This star implements a multichannel input/output FIR predictor, which is adapted using the power normalized LMS algorithm. It can be used as an equalizer, FSE, DFE, or echo canceller. An arbitrary number \( p \) input channels are transversal filtered to produce an arbitrary number \( q \) output estimate signals.

Note: each output buffer connected to this star implies a separate output channel, and identically numbered error input channel. Input signal channels are then connected to higher numbered buffers. It is assumed that the estimate error is computed externally. Do NOT implement an external (causality) unit-delay from output estimate to input error; this delay is handled automatically.

Param. 1 - Name of ASCII input specification file. Filter orders and initial tap values are given. default => profile

The proper specification file format is:

\[
\begin{align*}
(\text{int}) & \text{ # output channels, } q \\
(\text{int}) & \text{ # input channels, } p \\
(\text{int}) & \text{ order of in ch.}#1 \ldots (\text{int}) & \text{ order of in ch.}#p \\
(\text{float}) & \text{ ch.#1, tap 1} \ldots (\text{float}) & \text{ ch.#1, tap last} \\
& \quad \{\text{output ch.1}\} \\
& \quad \ldots \\
(\text{float}) & \text{ ch.#p, tap 1} \ldots (\text{float}) & \text{ ch.#p, tap last} \\
& \quad \ldots \\
(\text{float}) & \text{ ch.#1, tap 1} \ldots (\text{float}) & \text{ ch.#1, tap last} \\
& \quad \{\text{output ch.q}\} \\
& \quad \ldots \\
(\text{float}) & \text{ ch.#p, tap 1} \ldots (\text{float}) & \text{ ch.#p, tap last}
\end{align*}
\]

Param. 2 - Name of output file, for final adapted filter values. Default => profileo. The file is written in proper input-file format. This file can then be used to initialize the filter for the next run, if desired.

It is assumed that each output prediction filter will create one estimate output for EACH input sample/error sample pair. Any decimation, etc. must occur externally.

Param. 3 - (float) lambda. It is a multiplicative factor to control adaptation rate. Default => 1.0

Param. 4 - (float) delta. Tap leakage factor. default => 1.0 Default implies no tap leakage occurs.

Param. 5 - (int) wait. number of samples to skip before starting adaptation. The predictor still inputs samples, and outputs a zero estimate. default => 0

Param. 6 - (int) adapt. number of samples to adapt filter. After this number, filter taps are fixed, and estimates are still produced. default => -1 (implies always adapt)
#define Mch 10 /* maximum i or o channels */

parameters
file ifile_name = "profile";
file ofile_name = "profileo";
float lambda = 1.0;
float delta = 1.0;
int wait = 0;
int adapt = -1;
end

states
int* orders; /* #taps for each input channel */
float* W; /* estimate filter taps */
int N; /* total number of taps */
int p; /* number of input data channels */
int q; /* number of output data channels */
float* Z; /* previous data storage */
FILE* fp;
float* xpower; /* avg. power in each input channel */
float* epower; /* avg. power in jp error channel */
end

declarations
int i,j,k;
char *calloc();
float xdata;
float error[Mch];
float z[Mch];
int index;
float estimate; /* prediction to output */
float xalpha; /* lms adaptation variable */
end

initialization_code
if(no_outputBuffers() < 1) {
    fprintf(stderr,"pred: no output data channels\n");
    return(1);
}
if(no_inputBuffers() < no_outputBuffers() +1) {
    fprintf(stderr,"pred: not enough input buffers\n");
    return(2);
}
if((fp = fopen(ifile_name, "r")) == NULL) {
    fprintf(stderr,"pred: can't open spec file %s\n", ifile_name);
    return(3);
}
fscanf(fp, "%d", &q);
fscanf(fp, "%d", &p);
if(p > Mch | q > Mch) {
    fprintf(stderr, "pred: more than %d i/o channel\n", Mch);
    return(4);
}

if( q != no_output_buffers() | p != no_input_buffers() - q ) {
    fprintf(stderr, "pred: spec file %s does not agree with topology\n",
            iflle_name);
    return(5);
}

if( (orders = (int*)calloc(p,4)) == NULL |
    (xpower = (float*)calloc(p,4)) == NULL |
    (epower = (float*)calloc(q,4)) == NULL ) {
    fprintf(stderr, "pred: can't allocate filter space\n");
    return(6);
}

N = 0;
for(j=0; j<p; j++) {
    if((fscanf(fp," %d",
                &orders[j])) != 1) {
        fprintf(stderr, "pred: improper input file %s\n",
                iflle_name);
        return(7);
    }
    N += orders[j];
}

if( (W = (float*)calloc(N*q,4)) == NULL |
    (Z = (float*)calloc(N,4)) == NULL ) {
    fprintf(stderr, "pred: can't allocate filter space\n");
    return(8);
}

for(k=0; k < q; k++) {
    for(i=0; i<N; i++) {
        if((fscanf(fp,"%f",&W[i*q+k])) != 1) {
            fprintf(stderr, "pred: improper input file %s\n",
                    iflle_name);
            return(7);
        }
    }
}

fclose(fp);

/* set up unit delay in error channels */
for(k=0; k < q; k++)
    set_dmin_in(k,1);

/* initialize input power variables */
for(j=0; j < p; j++)
    xpower[j] = 1e-1;
end

if(min_avail() == 0) return(0);

if(wait > 0) {
    wait--;
    for(k=0; k < q; k++) {
        it_in(k);
    }
}
predlms

```c
    it_out(k);
    outf(k,0) = 0;
}
for(j==0; j<p; j++)
    it_in(q+j);
return(0);
}

/*** Normal Operation ***/
/* Input new data and create vector */
for(j==0; j<p; j++) {
    it_in(q+j);
    z[j] = inf(q+j,0);
}
/* Input unit-delayed error(s) */
for(k==0; k<q; k++) {
    it_in(k);
    error[k] = inf(k,1);
    /* Update error power state */
    epower[k] *= .96;
    epower[k] += .04 * error[k] * error[k];
}
if(adapt == 0) goto resume;

/*** Lms algorithm *****
for(k==0; k<q; k++) {
    /* Update filter taps */
    index = 0;
    for(j==0; j<p; j++) {
        xalpha = error[k]/(2 * (1+orders[j]) * xpower[j]);
        xalpha *= lambda;
        /* LMS Update taps */
        for(i==0; i<orders[j]; i++) {
            W[index*q+k] += xalpha * Z[index];
            W[index*q+k] *= delta;
            index++;
        }
    }
}
resume:
/* Update data array */
for(i=N-1; i>0; i--)
    Z[i] = Z[i-1];
index = 0;
for(j==0; j<p; j++) {
    Z[index] = z[j];
    index += orders[j];
}
/* Compute, output estimate for each output channel */
for(k==0; k<q; k++) {
    estimate = 0;
    for(i==0; i<N; i++)
        estimate += W[i*q+k] * Z[i];
    it_out(k);
    outf(k,0) = estimate;
}
```
if(adapt > 0) adapt--; 

end

wrapup_code

for(k=0; k<q; k++) {
    if(epower[k] <= 0) epower[k] = 1e-30;
    fprintf(stderr, "predlms: pred error power(%d) = %#g = %#.3g dB\n", k, epower[k], 10*log10(epower[k]));
}

if((fp = fopen(ofile_name, "w")) == NULL) {
    fprintf(stderr, "pred: can't open output file %s\n", ofile_name);
    return(1);
}

fprintf(fp, "%d\n", q);
fprintf(fp, "%d\n", p);
for(j=0; j<p; j++) {
    fprintf(fp, "%d\n", orders[j]);
}

fprintf(fp, "\n");
for(k=0; k<q; k++) {
    index = 0;
    for(j=0; j<p; j++) {
        fprintf(fp, "\n");
        for(i=0; i<orders[j]; i++) {
            fprintf(fp, "\n");
            fprintf(fp, "%g\t", W[index*q+k]);
            if((i+1)%5 == 0) fprintf(fp, "\n");
            index++;
        }
        fprintf(fp, "\n");
    }
    fprintf(fp, "\n");
}
fclose(fp);
free((char *)orders); free((char *)W);
free((char *)xpower); free((char *)epower);
end
Description:
This star inputs 0/1 binary data and outputs various line codes. Line codes are selectable by the first input parameter 'code_type':
- 0 - Binary (NRZ) \(1 = +1, 0 = -1\) (Default; 1 phase)
- 1 - Biphase (Manchester) \(1 = -1,+1; 0 = +1,-1\) (2 phase)
- 2 - 2BIQ \(00 = -3, 01 = -1, 10 = +1, 11 = +3\) (1 phase)

The code output oversampling rate (samples per baud interval) is selected by the second parameter 'smplbd'. Note that multi-phase codes require oversampling rates which are integer multiples of the number of phases!

I/O buffers are float to be compatible with most stars.

Programmer: L.J. Faber
Date: 11/25/86
Modified: 4/18/88 add 2BIQ
Modified: 5/88 add output1 as coded symbols, not oversampled
Modified: 6/88 change oversampling method to zero fill

```c
parameters

    int code_type = 0;
    int smplbd = 8;
end

input_buffers

    delay_max = 1;
    float blndata;
end

output_buffers

    float lcode;
    float symbol;
end

states

    float* template; /* one's pattern */
    int even = 1; /* even # of input bits */
end

declarations

    int i;
    int sign;
    int mag;
    int code_val;
    char *calloc();
end

initialization_code
if((template = (float*)calloc(smplbd, 4)) == NULL) {
    fprintf(stderr, "linecode: cannot allocate space\n");
    return(1);
}

if(code_type == 0) {  /* Binary */
    template[0] = 1.;
} else if(code_type == 1) { /* Biphasic */
    if(smplbd%2 != 0) {
        fprintf(stderr, "linecode: oversample rate not compatible with code type\n");
        return(1);
    }
    template[0] = -1.;
    template[smplbd/2] = 1.;
} else if(code_type == 2) { /* 2BIQ */
    template[0] = 1.;
} else {
    fprintf(stderr,"linecode: unrecognized code type\n");
    return(2);
}

end

main_code
while( it_in(0) ) {
    if(code_type == 2) {
        even = ++even % 2;
        if(even) {
            mag = 1;
            if(bindata(0) == bindata(1)) mag = 3;
            sign = -1;
            if(bindata(0) > 0.5) sign = 1;
        } else continue;
    } else { /* other code types */
        mag = 1;
        sign = -1;
        if(bindata(0) > 0.5) sign = 1;
    }

    code_val = sign * mag;
    for(i=0; i<smplbd; i++) {
        it_out(i);
        lcode(0) = code_val * template[i];
    }
}

end

wrapup_code
    free((char*)template);
end
This star performs Nyquist pulse shaping for a baseband transmitter. See Carlson, Communications Systems, page 381, equation 17b. The Nyquist criterion in the frequency domain is to have an amplitude rolloff which is symmetric about Fb/2 (half baud frequency).

First, a frequency-domain amplitude response is created using a raised cosine shape. This computation is affected by:

- **Param:**
  - 1 - (int) smplbd: samples per baud interval. default: 8
  - 2 - (int) expfft: 2^expfft = fft length to use. default: > 8
  - 3 - (float) beta: filter rolloff factor, 0 < beta < 0.5 default: > 0.5

The amplitude response is changed to impulse response via inverse FFT. The impulse response is made causal by right shifting (filter delay), and is time limited to "IMPBAUD" baud intervals (set by definition). (This filter will cause a delay of IMPBAUD/2 baud intervals.)

Finally, the impulse response is transformed back to a frequency response, which is used in subsequent linear convolution with the input, which is implemented by the Fast Fourier Transform overlap-save method.

The fft length must be greater than the impulse response length; for efficiency, a factor of two or more in length is desirable. This implies that 2^expfft > smplbd * IMPBAUD.

Nyquist shaping has no meaning if smplbd = 1; this implies that each sample would go through the filter unchanged!

Programmer: L.J. Faber
Date: Jan. 14, 1987
Modified: 6/88 ljfaber. expand comments, fix flaws

```c
#include <math.h>
#define PI 3.1415926
#define IMPBAUD 12  /* # baud widths of impulse response to save */

parameters
  int smplbd = 8;
  int expfft = 8;
  float beta = .5;
end

states
  float* nyqfresp; /* freq. response of nyquist filter */
  float* temp;
  float* save;
  int fftl = 1 << expfft; /* FFT length */
  int_impl = IMPBAUD * smplbd; /* impulse response length */
  int pcount = impl; /* count samples */
end

input_buffer
  float x;
end

output_buffer
  float y;
```
end

declarations

int i;
int j;
float freq; /* fractional norm. frequency */
float val; /* temporary */
end

initialization_code

if(smplbd < 2) {
    fprintf(stderr, "filtnyq: do not use if smplbd < 2\n");
    return(1);
}
if(fftl <= impl) {
    fprintf(stderr, "filtnyq: ft length too short\n");
    return(2);
}
if(beta <= 0. | beta > .5) {
    fprintf(stderr, "filtnyq: beta param out of range\n");
    return(3);
}
if((nyqfresp = (float*)calloc(fftl,4)) == NULL |
    (temp = (float*)calloc(fftl,4)) == NULL |
    (save = (float*)calloc(impl,4)) == NULL ) {
    fprintf(stderr, "filtnyq: can't allocate work space\n");
    return(4);
}

/* Compute nyquist frequency response */
/* store in 'folded, real fft' form */
/* note: frequency is normalized to unity at baud rate */
for(i=0; i<fftl/2; i++) {
    freq = i * ((float) smplbd / fftl);
    temp[2*i+1] = 0.; /*imaginary part*/
    if(freq < .5 - beta) temp[2*i] = smplbd;
    else if(freq > .5 + beta) temp[2*i] = 0.;
    else { /* frequency in rolloff region */
        val = cos(PI * (freq -.5 + beta)/(4.* beta));
        temp[2*i] = smplbd * val * val;
    }
}

/* Construct realizable, truncated impulse response */
/* (add delay by getting samples from end of imp. resp.) */
rfti(temp, fftl);
for(i=0; i<fftl; i++) {
    if(i < impl)
        nyqfresp[i] = temp[(i + fftl - impl/2)%fftl];
    else
        nyqfresp[i] = 0;
}

/* Back to frequency response form */
rfft(nyqfresp, fftl);
for(i=0; i<impl; i++)
    temp[i] = 0.;
end

main_code

while(it_in(0)){
    temp[pcount++] = x(0);
    if(pcount == fttl){
        for(i=0; i<impl; i++)
            save[i] = temp[fftl - impl + i];
        rfft(temp, fttl);
        cmultft(temp, nyqfresp, fttl);
        rfft(temp, fttl);
        for(i=impl; i<fftl; i++)
            it_out(0);
                y(0) = temp[i];
    }
    pcount = impl;
    for(i=0; i<impl; i++)
        temp[i] = save[i];
}
end

wrapup_code

free((char*)temp);
free((char*)nyqfresp);
free((char*)save);
end
//*************************************************************

lconv()

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

"Linear Convolution":
This star convoloves its input signal with an impulse response to
generate the output signal.
 Param. 1 - (int) impl: length of impulse response in samples.
        2 - (file) impf_name: ASCII file which holds impulse response.
        3 - (int) fftexp: log2(fft length).

Convolution is performed by the fft overlap-save method (described in
Oppenheim & Schafer, Digital Signal Processing, pp. 113).

The FFT length must be greater than the impulse response length.
For efficiency, it should probably be more than twice as long.

Programmer: M. R. Civanlar
Date: November 16, 1986
Modified: ljfaber, Dec86, Feb87
Modified: 6/88 ljfaber update comments, efficiency
*/

#include <math.h>

parameters
    int impl;
    file impf_name = *imp.dat*;
    int fftexp;
end

states
    float* temp;
    float* iresp;
    float* save;
    int pcount = impl;
    int fftl = 1 << fftexp;
end

input_buffer
    float X;
end

output_buffer
    float y;
end

declarations
    int i;
    char *callocO;
    FILE* fp;
end

initialization_code

    if(impl < (2.89 * log((float)fftl) + 1))
        fprintf(stderr,"lconv: direct convolution may be faster\n");
if(impl >= fftl) {
    fprintf(stderr,"lconv: FFT length too small\n");
    return(1);
}
if( (temp = (float*)calloc(fftl,4)) == NULL
| (iresp = (float*)calloc(fftl,4)) == NULL
| (save = (float*)calloc(impl,4)) == NULL ) {
    fprintf(stderr,"lconv: can't allocate work space\n");
    return(2);
}
if((fp = fopen(lmpf_name,"r")) == NULL) {
    fprintf(stderr,"lconv: can't open file %s, lmpf_name;\n");
    return(3);
}
for(i=0; i<impl; i++) {
    if( (fscanf(fp,"%f", &iresp[i])) != 1) {
        fprintf(stderr, "lconv: file %s too short\n",
            lmpf_name);
        return(4);
    }
}
fclose(fp);

main_code
while(it_in(0)) {
    temp[pcount++] = x(0);
    if(pcount == fftl) {
        for(i=0; i<impl; i++)
            save[i] = temp[fftl-impl+i];
        rfft(temp, fftl);
        cmultfft(temp, iresp, fftl);
        rfti(temp, fftl);
        for(i=impl; i<fftl; i++){
            it_out(0);
            y(0) = temp[i];
        }
        pcount = impl;
        for(i=0; i<impl; i++)
            temp[i] = save[i];
    }
}

end

wrapup_code
free((char*) temp);
free((char*) iresp);
free((char*) save);
end
This star simulates a decision element for a data receiver. It compares the incoming signal to a set of thresholds, which are computed from the user-specified set of output levels. Thresholds are always exactly half way between specified output levels.

Output levels are specified via a topology file parameter array. Parameter arrays are float, so there are no restrictions on specified output values. Only 10 levels are allowed, and must be listed in ascending magnitude order.

Examples—binary decision (threshold = 0.0)
param array 2 -1.0 1.0

quaternary decision (thresholds -2/0/2)
param array 4 -3. -1. 1. 3.

The number of output channels is determined at run-time (auto-fanout).

Programmer: L.J. Faber
Date: April, 1988
Modified:
*/

input_buffers
   float x;
end

parameters
   array level;
end

states
   int obufs;
   float thresh[10];
end

declarations
   int i,j;
end

initialization_code
   if((obufs = no_outputBuffers()) < 1) {
      fprintf(stderr,"sUcer: no output buffers\n");
      return(2);
   }
   if(n_level < 2) {
      fprintf(stderr,"slicer: improper parameters\n");
      return(1);
   }
   for(i=0; i<n_level-1; i++) {
      if(level[i] >= level[i+1]) {
         fprintf(stderr,
"slicer: level parameters must be in ascending order!\n");
      }
   }
return(3);
}

thresh[i] = (level[i] + level[i+1])/2.;

end

main_code

while(it_ln(0)) {
    for(i=0; i<n_level-1; i++) {
        if(x(0) < thresh[i]) break;
    }
    for(j=0; j<obufs; j++) {
        it_out(j);
        outf[j,0] = level[i];
    }
}

return(0);
end
# utop.t
# This universe uses transmitter tx.t, receiver rx.t
# channel channel.t with noise add, local echo.

param int 2000
param int 8
param int 1
galaxy txrem tx.t

# noise variance
param float 1e-12
param file ..:/line2.imp
param file ..:/echo2.imp
galaxy chan channel.t

param int 8
param int 1000
param int 1000
param int 1000
param int 1000
param int 1000
param int 0
galaxy rxloc Rx.t

param int 2000
param int 8
param int 2
galaxy txloc tx.t

connect txrem 0 chan 0
connect txrem 1 rxloc 2
connect chan 0 rxloc 0
connect txloc 0 chan 1
connect txloc 1 rxloc 1
# tx.t
# This galaxy simulates a data transmitter system.
# Binary data is generated, 2B1Q linecoded, oversampled
# and nyquist filtered.
# creates dsp output file tx.out of signal launched.
# arg 0 - (int) number of bits to generate (2*symbols)
# arg 1 - (int) oversampling rate.
# arg 2 - (int) seed for random data generator
# inputs none.
# output 0 - line conditioned data
# output 1 - coded symbols at symbol rate

param arg 0
param arg 2
star bdata data

param int 2
param arg 1
star coder linecode

param arg 1
param int 8
param default
star txfilt fltnyq

connect bdata 0 coder 0
connect coder 0 txfilt 0
connect coder 1 output 1
connect txfilt 0 output 0
# channel.t

# This galaxy simulates a half-duplex transmission channel,
# including additive noise and echo channel.
# input 0 - remote linecoded tx symbols
# input 1 - local tx symbols
# output 0 - local channel output
# arg 0 - (float) variance of additive noise
# arg 1 - (file) name of end-to-end impulse response file
# arg 2 - (file) name of echo impulse response file

param int 256
param arg 1
param int 9
star chfilt1 lconv

star add1 ladd

param arg 0
param int 137
star noisel laddnoise

param int 256
param arg 2
param int 9
star echo lconv

connect input 0 chfilt1 0
connect chfilt1 0 add1 0
connect add1 0 noisel 0
connect noisel 0 output 0

connect input 1 echo 0
connect echo 0 add1 1
# trainer.t
# This galaxy models a receiver trainer. It switches from input 0
# and input 1 as the source of output bits, after a specified time
# interval (number of samples). Input 0 is for training symbols,
# and input 1 is for local (slicer generated) symbols.
# It also counts slicer errors, after the training period.
# The error counter is set to always give error messages, even during
# the training period, in order to monitor convergence.
# arg 0 - (int) channel delay, in symbols
# arg 1 - (int) # of initial training symbols

param arg 0
star dly delay

star fk1 fork
star fk2 fork

param arg 1
star switch toggle.o

param arg 1
param int 0
star err ecount

connect input 0 dly 0
connect dly 0 fk1 0
connect fk1 0 switch 0
connect fk1 1 err 0

connect input 1 fk2 0
connect fk2 0 switch 1
connect fk2 1 err 1
connect switch 0 output 0
# Rx.t
# This is a complete 2-channel receiver topology
# It uses modular echo canceller and equalizer
# arg 0 - oversampling rate
# arg 1 - ec training symbols
# arg 2 - ec symbols to wait
# arg 3 - ec switching symbol
# arg 4 - eq training symbols
# arg 5 - eq symbols to wait
# input 0 - received waveform, oversampled
# input 1 - local transmitted symbols
# input 2 - remote transmitted symbols
# outputs none

param arg 0
# sampling phases
param array 2 0 4
star decim demux

param arg 1
param arg 2
param arg 3
galaxy ec EC.t

param arg 4
galaxy eq EQ.t

connect input 0 decim 0
cconnect decim 0 ec 0
cconnect decim 1 ec 1
cconnect input 1 ec 2

cconnect ec 0 eq 0
cconnect ec 1 eq 1
cconnect input 2 eq 2
DFE, FSE Equalizer

# EQ12.t
# This galaxy simulates a data channel equalizer, with training.
# predfft adapt. equalizer (b)
# DFE, FSE (x2) => '12'
# 2BIQ decoding
# arg 0 - number of eq training symbols
# arg 1 - number of symbols to wait
# input 0 - channel data, phase1
# input 1 - channel data, phase2
# input 2 - remote transmitter codes (for training)
# no outputs

param float .01
star gain multiply

param file eqfile
param file eqfileo
param float 1.0
param float 1e-4
param arg 1
param arg 0
star equalizer predfft.o

param array 2 1 -1
star summer sum.o

param array 4 -3 -1 1 3
star slicer slice.o

param int 1
star dly delay

star fk1 fork
star fk2 fork

param file powfile
param int 10
star meter powmeter

# channel bulk delay
param int 12
param int 2000
galaxy sw trainer.t

connect input 0 equalizer 2
connect input 1 equalizer 3
connect equalizer 0 fk1 0
connect fk1 0 slicer 0
connect fk1 1 summer 1
connect slicer 0 sw 1
connect sw 0 fk2 0
connect fk2 0 summer 0
connect fk2 1 dly 0
connect dly 0 gain 0
connect gain 0 equalizer 1
connect summer 0 meter 0
connect meter 0 equalizer 0
connect input 2 sw 0
# EC22s.t
# This is an echo canceller with switched feedback (off phase 0)
# 2 input channels (pole/zero), 2 output channel
# arg 0 - # of training symbols
# arg 1 - # of symbols to wait
# arg 2 - symbol to switch on
# input 0 - channel data, phase1
# input 1 - channel data, phase2
# input 2 - local transmitter symbols
# output 1 - phase1 minus echo
# output 2 - phase2 minus echo

param arg 2
star switch toggle
star fkl fork
star fkl2 fork

par 2 1 -1
star sum0 sum
par 2 1 -1
star sum1 sum

param file ecfile
param file ecfileo
param float 1.0
param float 1e-4
param arg 1
param arg 0
star ec predftf.o

param float .01
star gain multiply

param int 1
star dly delay

param file powec
param int 10
star meter powmeter

connect input 0 fkl 0
connect fkl 0 sum0 0
connect sum0 0 output 0
connect sum0 1 ec 0
connect sum0 2 meter 0
connect input 1 sum1 0
connect sum1 0 output 1
connect sum1 1 ec 1
connect fkl 1 switch 0
connect switch 0 dly 0
connect dly 0 ec 2
connect input 2 gain 0
connect gain 0 ec 3
connect ec 0 fkl2 0
connect fkl2 0 sum0 1
connect fkl2 1 switch 1
connect ec 1 sum1 1
Figure 6.4 Transmitter Eye Diagram, a) $\beta = .5$  b) $\beta = .3$