Frequency Domain FIR and IIR Adaptive Filters

D. W. Lynn
Department of Electrical Engineering
University of Idaho
Moscow, Idaho

Abstract

A discussion of the LMS adaptive filter relating to its convergence characteristics and the problems associated with disparate eigenvalues is presented. This is used to introduce the concept of proportional convergence. A novel approach is used to analyze the convergence characteristics of block frequency-domain adaptive filters. This leads to a development showing how the frequency-domain FIR adaptive filter is easily modified to provide proportional convergence. These ideas are extended to a block frequency-domain IIR adaptive filter and the idea of proportional convergence is applied. Experimental results illustrating proportional convergence in both FIR and IIR frequency-domain block adaptive filters is presented.

1 The LMS Adaptive Filter

We first present Widrow's LMS adaptive filter, analyzing it's mean convergence following his approach in [5]. The equations describing an FIR filter in both scalar and vector notation are

$$y_j = \sum_{i=0}^{N-1} w_i x_{j-i} = W^T X_j = X_j^T W$$

with $X_j^T = \left[x_j, x_{j-1}, x_{j-2}, \ldots, x_{j-(N-1)}\right]$ and $W^T = [w_0, w_1, \ldots, w_{N-1}]$.

The goal of an adaptive filter is to automatically adjust the weights W so that the difference between the output $\{y_n\}$ and some desired signal $\{d_n\}$ is a minimum in a mean square sense. That is, with $e_j = d_j - y_j = d_j - W^T X_j = d_j - X_j^T W$

$$E\left\langle (e_j)^2 \right\rangle = E\left\langle d_j^2 \right\rangle - 2E\left\langle d_j X_j^T \right\rangle W + W^T E\left\langle X_j X_j^T \right\rangle W$$

where $E\langle\rangle$ represents the expectation operator. With $P_{xd} = E\langle d_j X_j \rangle$ (the cross-correlation between $\{d_n\}$) and $\{x_n\}$ and $R_{xx} = E\left\langle X_j^T X_j \right\rangle$ (the autocorrelation of $\{x_n\}$), the mean square error ξ can be written

$$\xi = E\left\langle d_j^2 \right\rangle - 2P_{xd}^T W + W^T R_{xx} W \qquad (1)$$

Since the MSE (mean square error) is a hyperquadratic function of the weights, in essence a bowl shaped surface with a single global minimum, the minimum can be sought using simple gradient search techniques. That is

$$W_{j+1} = W_j - \mu \nabla_j$$

where μ is a scalar constant that controls the rate of adaptation, and ∇_j is the gradient of the error surface with respect to the weights evaluated at time j.

$$\nabla_j = \frac{\partial E\left\langle e_j^2 \right\rangle}{\partial W} = -2P_{xd} + 2R_{xx}W \tag{2}$$

Setting gradient to 0 yields the optimum choice of W for minimum mse

$$W^\star = R_{xx}^{-1} P_{xd} \tag{3}$$

Since R_{xx} is positive definite, it is non-singular and it's inverse exists. This equation is the Weiner-Hopf equation written in matrix form, thus $W^\star$ represents the optimum Weiner filter.

Obviously, if R_{xx} and P_{xd} are known exactly (3) gives the optimum weights and requires the inversion of the autocorrelation matrix, but this need only be done once. In one approach to adaptive filtering, a large number of data samples are processed to obtain an accurate estimate of R_{xx} and P_{xd}, then R_{xx}^{-1} is found and $W^\star$ is computed and the data are filtered using $W^\star$. This approach requires a large amount of data storage, imposes a significant processing delay and works well only for stationary signals. The LMS approach uses the gradient method presented above but to minimize the storage requirement, the gradient is estimated on the basis of no more data than are present in the filter itself. That is, at iteration j, we estimate $R_{xx} \approx X_j X_j^T$ and $P_{xd} \approx d_j X_j$. With this, the estimated gradient at iteration j becomes $\hat{\nabla}_j = -2e_j X_j$ and the weight update becomes

$$W_{j+1} = W_j + 2\mu e_j X_j \tag{4}$$

Admittedly, these estimates of the gradient are noisy, but if μ is chosen to be small, the error in each estimate will contribute little to the final solution.

To analyze the convergence characteristics of the LMS adaptive filter, consider an ensemble of adaptive processes that all begin with the same initial weight vector W_0. Also, the inputs to each adaptive filter are drawn from the same statistical populations. If we take the expected value over the ensemble of the weight update (4), we have

$$E\left\langle W_{j+1} \right\rangle = E\left\langle W_j \right\rangle + 2\mu\left(P_{xd} - R_{xx}E\left\langle W_j \right\rangle\right) \tag{5}$$

provided that X_j and W_j for each adaptive process are uncorrelated. We first note, that *in the mean*, the gradient estimate equals the true gradient, so, it the adaptive filter converges, the weight vector converges to the Weiner solution. We can simplify (5) if we translate the weight vector coordinates so that the optimum weight vector in the new coordinate system is 0. That is, let the new weight coordinates be represented by the vector V where $V_j = W_j - W^\star$. Letting W_j be $V_j + W^\star$ and recognizing that $P_{xd} = R_{xx}W^\star$, the above expression becomes

$$\begin{aligned} E\left\langle V_{j+1} \right\rangle &= \left(I - 2\mu R_{xx}\right) E\left\langle V_j \right\rangle \\ &= \left(I - 2\mu R_{xx}\right)^{j+1} V_0 \end{aligned} \tag{6}$$

This is a geometric equation whose convergence depends on R_{xx} and μ. If we now rotate the weight space so that the axes fall along the principle axes of R_{xx}, the above equation will be transformed into a set of uncoupled scalar equations. Since R_{xx}

is symmetric, it can be orthogonally diagonalized as in $\Lambda = Q^{-1}R_{xx}Q$ where Λ is a diagonal matrix composed of the eigenvalues of R_{xx}, that is $\Lambda = diag[\lambda_0, \lambda_1, \ldots, \lambda_{N-1}]$ and Q is the orthonormal modal matrix of R_{xx}, that is the columns of Q are the normalized eigenvectors of R, each of which is distinct and orthogonal to the other eigenvectors. Thus $QQ^T = I$ and $Q^{-1} = Q^T$. Now, substituting $R_{xx} = Q\Lambda Q^T$ into (6) and premultiplying both sides of the equation by Q^T gives

$$E\left\langle V_j' \right\rangle = (I - 2\mu\Lambda)^j V_0' \tag{7}$$

where $V' = Q^T V$ is a rotation of the translated weight coordinates into the principle axes of R_{xx}. Now, because Λ is diagonal, the above equation decouples into a set of scalar equations

$$v'_{p_j} = (1 - 2\mu\lambda_p)^j v'_{p_0}$$

with v'_p representing the pth element of V'. We will refer to $\{v_p\}$ as *modes* of the adaptive filter. It is evident that for $E\left\langle V_j' \right\rangle$ and hence $E\langle W_j \rangle$ to converge, we must have

$$|1 - 2\mu\lambda_p| < 1 \; \forall p$$

which will be satisfied if $0 < \mu < 1/\lambda_{max}$ since $0 < \lambda_p \; \forall p$ because R_{xx} is positive definite. It is important to note that this only guarantees convergence *in the mean* over an ensemble of adaptive processes. Note here that choosing $\mu = 1/2\lambda_p$ will give the fastest convergence for mode p. In fact it should converge *in the mean* in one step. but unless $\lambda_p = \lambda_{max}$, other modes will not converge. μ must be typically be chosen $< 1/2\lambda_{max}$ because the gradient estimate is itself noisy. Often, the eigenvalues are not directly available. Since R_{xx} is positive definite, $\lambda_{max} < (\sum_{i=0}^{N-1} \lambda_i = \mathbf{tr}R_{xx})$ and since $\mathbf{tr}R_{xx}$ is just the average power in $\{x_n\}$, μ is conveniently chosen as $1/\mathbf{tr}R_{xx}$.) It should be clear that λ_{max} limits rate at which the filter can converge. The mode associated with λ_{max} converges the fastest and that the mode associated with λ_{min} will be the last to converge. That is, the overall convergence of the LMS algorithm is controlled by the spread in the eigenvalues of R_{xx}. This aspect has attracted a lot of attention among researchers attempting to speed up LMS convergence. If all the eigenvalues were equal, the LMS algorithm could be made to converge at the fastest possible rate, taking into account that the overall rate must still be relatively slow to compensate for the fact that we have only estimated the gradient.

Another related problem we encountered [13] has to do with the non-proportional convergence of the modes of the adaptive filter, particularly in non-stationary environments. The most highly correlated modes of R_{xx} will have the largest eigenvalues and these modes will be resolved first. Since the eigenvalues of R_{xx} are related to the power spectral density of X, this essentially means that the spectral components of X that have the most power will be resolved first. If the problem is non-stationary, the lower power components may never be resolved. This effectively alters the power spectral density of the output process in a way that may be undesirable. For example, consider processing speech for noise cancellation using the line enhancer configuration of the adaptive filter. The spectral components of speech having the highest power are those in the low frequencies, so these will be resolved first. The high frequencies may never be resolved. In this case the non-proportional convergence has effectively added a low-pass filter. This is very undesirable as a large percentage of the intelligibility of speech is carried in the higher frequencies.

One way to achieve both goals of faster and more proportional convergence is to normalize (7) by replacing μ with $\mu\Lambda^{-1}$. This amounts to replacing a scalar-μ by a vector or a matrix μ. If we do this, and follow our development back through the rotation and translation steps, we arrive at the update equation

$$W_{j+1} = W_j + 2\mu R_{xx}^{-1} e_j X_j \tag{8}$$

A number of algorithms have been developed using this approach which is essentially Newton's method for the minimization of a quadratic surface. Most of these are referred to as "self-orthogonalizing" adaptive filters and they attempt to estimate R_{xx}^{-1} at each iteration. [26,6][5,20] This results in a much larger computational load than might be desired. Not only is there the additional load of estimating R_{xx}^{-1}, but also an extra matrix multiply is required between it and the gradient estimate.

Another way to do the same thing, and the approach we pursue in this paper, is to transform the input data into the rotated space, perform the update and output computations in the rotated space and then inverse transform the output data. To see how this might work, let us premultiply both sides of (8) with Q^T. With $R_{xx}^{-1} = Q\Lambda^{-1}Q^T$, this becomes

$$W'_{j+1} = W'_j + 2\mu\Lambda^{-1} e_j X'_j \tag{9}$$

where $W'_j = Q^T W_j$ and $X'_j = Q^T X_j$ represent vectors in the rotated space. Because Λ is diagonal, we shall often refer to $\mu\Lambda^{-1}$ as a *vector-μ*. Note that each of the modes of the adaptive process decouple and we end up with N one-weight adaptive filters with the adaptive gain equal to μ/λ_p for the *pth* mode. This can also be seen in the fact that $\Lambda = E\left\langle X'_j X'^T_j \right\rangle$, so λ_p is just the power in the *pth* component of X'_j averaged over the iterations j. This approach is very attractive in that it requires no more computation than the usual LMS time-domain approach, yet it promises improved convergence rates as well as more proportional convergence. The only obstacle to be overcome is the problem of transforming X into the rotated space. We could attempt to estimate Q, but this is essentially the same as the Newton's methods discussed above. What is needed is an orthogonal transform that decouples the spectral components of X. Several candidates have been studied, the most prominent among them being the Discrete Fourier Transform (DFT). Efforts to improve convergence in this way developed synergisticly with efforts to reduce the number of computations required in the adaptive filter by using the Fast Fourier Transform (FFT) to implement high speed convolution and correlation.

Yet another recent approach [30] uses a DCT to estimate $\{\lambda_p\}$, orders the set by magnitude and then assigns to μ a sequence of values related to the reciprocals of the ordered set $\{\lambda_p\}$. Effectively, once the mode associated with λ_{max} is converged, μ can be increased to speed the convergence of the mode associated with the next largest eigenvalue until it too is converged and so on.

2 The Block Adaptive Filter

A block adaptive filter using FFTs to perform fast convolution and correlation was proposed nearly simultaneously in 1980 by Clark et. al. [9,14,18] and Ferrara [10]. Clark also presented the effect of block processing on the convergence and misadjustment of the adaptive filter as opposed to point by point processing. Waltzman and

Schwartz [1,3] had earlier applied the use of the FFT to the automatic channel equalizer. Not only did they show that the filter could be run with fewer computations, but by adapting the weights in the frequency domain they were able to obtain tighter bounds on λ_{max} and λ_{min} and hence a more accurate setting for μ. The following discussion is based largely on the approach used by Clark et. al.

We note that the convolution $y_j = W^T X_j = X_j^T W$ can be written in matrix form as

$$\begin{bmatrix} y_j \\ y_{j+1} \\ y_{j+2} \\ y_{j+3} \\ \vdots \end{bmatrix} = \begin{bmatrix} X_j^T \\ X_{j+1}^T \\ X_{j+2}^T \\ X_{j+3}^T \\ \vdots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix} \tag{10}$$

or as

$$Y_l = X_l W \tag{11}$$

where Y_l is an L element vector $[y_{lL}, y_{lL+1}, \cdots, y_{(l+1)L-1}]^T$ and X_l is an L by N matrix whose rows are the transposes of the vectors $X_j = [x_j, x_{j-1}, \ldots, x_{j-(N-1)}]^T$ for $j = (lL, lL+1, ..., (l+1)L-1)$. We also similarly define desired and error vectors D_l and E_l so that $E_l = D_l - Y_l = D_l - X_l W$. Thus, instead of computing the error for every input point, we only do so once every L points. For this approach, we are interested in minimizing the block mean square error ξ_B.

$$\xi_B = E\left\langle \frac{1}{L} E_l^T E_l \right\rangle$$

Under the assumption that the inputs are stationary, it is easy to show that the block mean square error will be the same as the mean square error in the unblocked filter. Further, the Weiner optimum weights will be the same in both cases.

We also only update the weights once per block. Following the same approach as for the unblocked case, we approximate the the gradient of the ξ_B using only the information available at that iteration. That is $\xi_B \approx 1/L E_l^T E_l$. (We note that this is L times the information that was available to the unblocked filter so we suspect that this gradient estimate is not as noisy as the estimates generated by an unblocked filter.) This leads to a block weight update equation

$$W_{l+1} = W_l + \frac{2\mu_B}{L} X_l^T E_l \tag{12}$$

where μ_B is the block adaptive gain constant.

By a similar analysis as applied to the unblocked filter, it can be shown that convergence is guaranteed *in the mean* if $0 < \mu_B < 1/\lambda_{max}$ which is the same condition as for the unblocked case. However, we must set $\mu_B = L\mu$ for the blocked and unblocked filters to converge at the same rate. Depending on L and how much smaller than $1/2\lambda_{max}$ μ is chosen to compensate for the larger gradient estimate noise in the unblocked filter, this may or may not be possible. So the blocked filter may be constrained to converge more slowly. This is particularly true in the case of highly disparate eigenvalues. We need to remember here that this discussion applies to a scalar μ_B and that when we eventually introduce a vector-μ, the convergence of the blocked algorithm can be made faster than that of the unblocked filter. There is, however, another factor that may

limit how large L can be. In applications where $\{x_n\}$ is slowly non-stationary, L must be small enough so that over several successive L-point blocks (the number depending on the convergence rate) $\{x_n\}$ is approximately stationary. Otherwise, the filter would, at best, not properly track the non-stationarity or, at worst, become unstable.

Let us next consider how to apply the FFT to reduce the number of computations required in the block adaptive filter.

The N-point DFT of a sequence $x(n)$ is computed

$$\mathcal{X}(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

where $W_N = e^{-j\frac{2\pi}{N}}$ and W_N^n for $n = 0, 1, 2, \ldots, N-1$ are the N roots of unity. Similarly, the inverse transform is given

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} \mathcal{X}(k) W_N^{-nk}$$

These transformations can be written in matrix form

$$\mathcal{X} = \sqrt{N}\mathcal{F}X \text{ and } X = \frac{1}{\sqrt{N}}\mathcal{F}^{*T}\mathcal{X}$$

where $\mathcal{X} = [\mathcal{X}(0), \mathcal{X}(1), \ldots, \mathcal{X}(N-1)]^T$, $X = [x(0), x(1), \ldots, x(N-1)]^T$, $\mathcal{F}$ is a matrix whose $(k,n)th$ element is $W_N^{kn}/\sqrt{N}$, and $*$ represents the complex conjugate and T represents the transpose operation. We note first of all that $\mathcal{F}$ is symmetric and that $\mathcal{F}^* = \mathcal{F}^{-1}$ so that $\mathcal{F}$ is a unitary transform. Also, it can be shown [25,2,21] that if C is a circulant matrix, $\mathcal{F}$ will diagonalize it. Further, if the diagonalization is expressed

$$\Lambda_C = \mathcal{F}C\mathcal{F}^* \tag{13}$$

then the eigenvalues of C and the elements of Λ_C will be given by the DFT of the first column of C.

With this background, let us now illustrate how the DFT can be used to implement the convolution in (11). First we rewrite (11) as follows

$$\begin{bmatrix} \cdot \\ Y_l \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ X_l & \cdot \end{bmatrix} \begin{bmatrix} W_l \\ 0 \end{bmatrix} \tag{14}$$

where the $\cdot$'s represent arbitrary elements which do not affect the equation. More simply,

$$Y_l^a = X_l^a W^a$$

If X_l^a can be made circulant, then the DFT can be used to diagonalize it. This can be done by defining X_l^a to be an $M \times M$ circulant matrix whose first column is the M-point vector $[x_{lL-(N-1)}, \ldots, x_{lL-1}, x_{lL}, x_{lL+1}, \ldots, x_{(l+1)L-1}]$. Each successive column is formed by circularly down shifting the previous column by one sample. The M point augmented weight vector W^a formed by padding the N-point W with $N - M$ zeros. The result will be another augmented vector Y_l^a whose last L points is the vector Y_l. With $I = \mathcal{F}^*\mathcal{F}$, we can write

$$Y_l^a = \frac{1}{\sqrt{M}}\mathcal{F}^*\mathcal{F}X_l^a\mathcal{F}^*\sqrt{M}\mathcal{F}W^a$$

which, with the circulancy of X_l^a, can be written

$$Y_l^a = DFT^{-1}\left\{DFT([x_{lL-(N-1)},\ldots,x_{lL},\ldots,x_{(l+1)L-1}]) \otimes DFT(W^a)\right\} \qquad (15)$$

where $\otimes$ is a point-by-point multiply of two vectors. Consider the following example with $l = 0$.

$$\begin{bmatrix} \hat{y}_{1-N} \\ \hat{y}_{2-N} \\ \vdots \\ y_0 \\ y_1 \\ \vdots \\ y_{L-2} \\ y_{L-1} \end{bmatrix} = \begin{bmatrix} x_{1-N} & x_{L-1} & \cdots & x_{L-N+1} & x_{L-N} & \cdots & x_{3-N} & x_{2-N} \\ x_{2-N} & x_{1-N} & \cdots & x_{L-N+2} & x_{L-N+1} & \cdots & x_{4-N} & x_{3-N} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_0 & x_{-1} & \cdots & x_{1-N} & x_{L-1} & \cdots & x_2 & x_1 \\ x_1 & x_0 & \cdots & x_{2-N} & x_{1-N} & \cdots & x_3 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{L-2} & x_{L-3} & \cdots & x_{L-N-1} & x_{L-N-2} & \cdots & x_{1-N} & x_{L-1} \\ x_{L-1} & x_{L-2} & \cdots & x_{L-N} & x_{L-N-1} & \cdots & x_{2-N} & x_{1-N} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \qquad (16)$$

From the above, we can see that the last $L = M - (N-1)$ points y_0 to y_{L-1} represent points from the linear convolution of $x(n)$ with W. The previous $N-1$ points are incorrect and are discarded. The next segment of $x(n)$ to be processed will overlap the previous by $N-1$ points, ie., $[x_{L-N+1},\ldots,x_{L-1},x_L,x_{L+1},\ldots,x_{2L-1}]$ This will produce another valid L points which are abutted with the previous set. This is the standard "overlap and save" approach to discrete convolution using the DFT.

Waltzman [1,3] and, later, Ferrara [10], observed that the update (12) involved a cross-correlation between X_l and E_l and that this also could be sped up by applying FFTs. To see how this is done we first observe that by taking the complex conjugate transpose of (13), we have

$$\Lambda_C^* = \mathcal{F}C^T\mathcal{F}^*$$

Now, as before, we augment the vectors in (12). We use the same circulant matrix X_l^a and augmented vector W^a. This requires the use of an augmented vector E_l^a, whose first $N-1$ points are 0 and whose last L points form E_l. That is

$$\begin{bmatrix} W_{l+1} \\ \cdot \end{bmatrix} = \begin{bmatrix} W_l \\ \cdot \end{bmatrix} + 2\frac{\mu_B}{L}\begin{bmatrix} \cdot & X_l^T \\ \cdot & \cdot \end{bmatrix}\begin{bmatrix} 0 \\ E_l \end{bmatrix} \qquad (17)$$

or, simply,

$$W_{l+1}^a = W_l^a + 2\frac{\mu_B}{L}X_l^{aT}E_l^a$$

This is fortunate because it gives us a consistent set of augmented vectors. W_l^a and X_l^a are the same for both the update and convolution equations. Also, $E_l^a = D_l^a - Y_l^a$, if we define an appropriate augmented vector for D_l. Next, with the fact that $\mathcal{F}^*\mathcal{F} = I$, the augmented update equation becomes

$$W_{l+1}^a = W_l^a + 2\frac{\mu_B}{L}\frac{1}{\sqrt{M}}\mathcal{F}^*\mathcal{F}X_l^{aT}\mathcal{F}^*\sqrt{M}\mathcal{F}E_l^a$$

Since $\mathcal{F}X_l^{aT}\mathcal{F}^*$ is given by the complex conjugate of the DFT of the first column of X_l^a, the update equation ultimately becomes

$$W_{l+1}^a = W_l^a + 2\frac{\mu_B}{L}DFT^{-1}\left\{\left(DFT([x_{lL-(N-1)},\ldots,x_{lL},\ldots,x_{(l+1)L-1}])\right)^* \otimes DFT(E_l^a)\right\} \qquad (18)$$

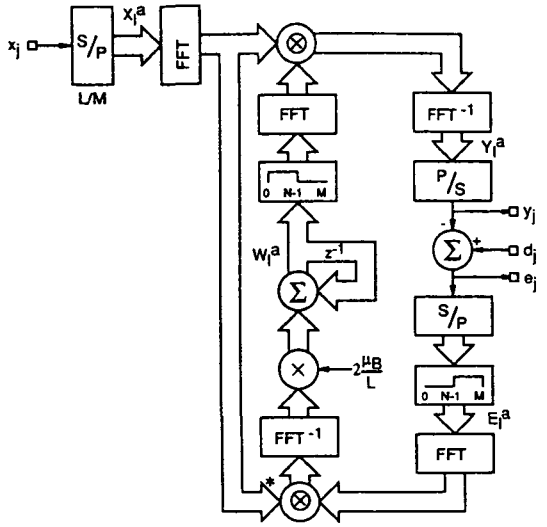

Figure 1: The Block Frequency Domain LMS Algorithm

Remembering that we must set the last $M - N$ points of W_{l+1}^a to zero before performing the next convolution, (15) and (18) describe the block frequency domain LMS algorithm. This process is summarized in Figure 1. Since the FFT is a linear operator, there is considerable flexibility in the structure of the algorithm. If we move the window to a position just before the adaptive gain multiply, we get a structure that reduces the number of multiplies and adds in the weight update calculation. Another possible rearrangement is shown in Figure 2. Here, the adaptive gain takes place in the frequency domain and opens up the possibility for selective gains for each frequency bin. In this structure, the FFT after the window could be moved above the weight recursion to reduce the number of additions.

Choosing the FFT length to be twice the number of weights ($M = 2N$) allows the filter to generate $L = N + 1$ valid output points at each iteration. Taking advantage of symmetries, we can argue [29] that a $2N$ radix 2 point FFT requires $2N\log_2(N) - 4N$ real multiplies and $3N\log_2(N) + 2N - 12$ real additions. Working from Figure 1, we see that there are 5 FFTs. Using the symmetry in the FFT of a real sequence, each $\otimes$ operation requires $N - 1$ complex multiplies and 2 real multiplies. The $\sum$ operations each require N real additions. The adaptive gain requires N real multiplies. Overall, to generate $N + 1$ output points requires $10N\log_2(N) - 11N - 4$ real multiplies and $15N\log_2(N) + 16N - 64$ real additions. To produce $N+1$ points from an N-weight time-domain adaptive filter requires $2N^2 + 3N + 1$ real multiplies and $2N^2$ real additions. Of course, the time domain filter performs a weight update for each output point produced and, although we could alter it to do an update only once every $N + 1$ points as the block filter does, would not result in any computational savings. The ratio of complexity of the frequency-domain block adaptive filter to the time-domain adaptive filter for several values of N is given in the following table.

We need to remember that while we have achieved some computational savings, that is not our only objective. Next we consider the issue of proportional convergence. Much of what follows is motivated by Picchi and Prati's presentation of a self-orthogonalizing adaptive equalizer in [20], however our approach is novel and re-

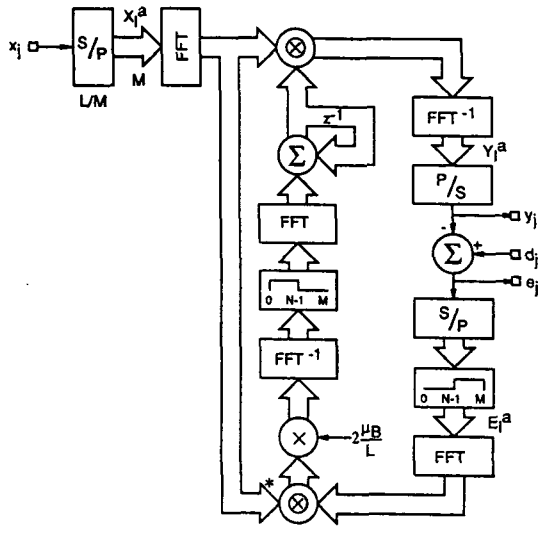

Figure 2: Another Version of The Block Frequency Domain Filter

N	Multiplies	Additions
8	0.967	3.313
16	0.820	2.250
32	0.580	1.391
64	0.374	0.820
128	0.228	0.471
256	0.134	0.265
512	0.077	0.147
1024	0.043	0.081

Table 1: Ratio of Frequency Domain to Time Domain Calculations

quires fewer approximations.

In order to formalize the sectioning and zero-padding operations we define two projection operators Π_N and Γ_L. Γ_L is constructed from an $M \times M$ identity matrix whose first $N-1$ diagonal elements have been set to 0. Thus $\Gamma_L = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}$ where I is $L \times L$. Π_N is similarly constructed but in this case the first N diagonal elements are left at 1 with the remaining $L-1$ elements set to 0. When Γ_L premultiplies a vector, it projects its first $N-1$ points to 0. When premultiplying a matrix, it sets the first $N-1$ rows of the matrix to 0. Postmultiplying a matrix by Γ_L is equivalent to setting the first $N-1$ columns to zero. Π_N has similar properties but applies to the last $L-1$ points of a vector or rows or columns of a matrix. With this (17) can be restated

$$W_{l+1}^a = W_l + 2\frac{\mu_B}{L}\Pi_N X_l^{aT} E_l^a \tag{19}$$

which formalizes the fact that the weight vector must be updated in such a way so that last $M-N$ elements of W_l^a are always 0.

Now, to consider the convergence of the augmented blocked filter, we take the expected value of (19) over an ensemble of adaptive processes just as we did in for the unblocked filter. This gives

$$E\left\langle W_{l+1}^a \right\rangle = E\left\langle W_l \right\rangle + 2\frac{\mu_B}{L}\Pi_N E\left\langle X_l^{aT} E_l^a \right\rangle$$

With $E_l^a = D_l^a - \Gamma_L X_l^a W_l^a$ and with the assumption of independence between X_l^a and W_l the above expands to

$$E\left\langle W_{l+1}^a \right\rangle = E\left\langle W_l \right\rangle + 2\frac{\mu_B}{L}\Pi_N \left(E\left\langle X_l^{aT} D_l^a \right\rangle - E\left\langle X_l^{aT} \Gamma_L X_l^a \right\rangle E\left\langle W_l \right\rangle \right)$$

Observing that

$$E\left\langle X_l^{aT} D_l^a \right\rangle = E\left\langle \begin{bmatrix} \cdot & X_l^T \\ \cdot & X_c^T \end{bmatrix} \begin{bmatrix} 0 \\ D_l \end{bmatrix} \right\rangle = L \begin{bmatrix} P_{xd} \\ 1/LE\left\langle X_c^T D_l \right\rangle \end{bmatrix} = L P_{xd}^a$$

and that

$$E\left\langle \begin{bmatrix} \cdot & X_l^T \\ \cdot & X_c^T \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \cdot & \cdot \\ X_l & X_c \end{bmatrix} \right\rangle = L \begin{bmatrix} R_{xx} & 1/LE\left\langle X_l^T X_c \right\rangle \\ 1/LE\left\langle X_c^T X_l \right\rangle & 1/LE\left\langle X_c^T X_c \right\rangle \end{bmatrix}$$

or, in augmented matrix notation,

$$E\left\langle X_l^{aT} \Gamma_L X_l^a \right\rangle = L R_{xx}^a$$

where we have chosen X_c to represent a part of the circulant matrix X_l^a. We can now write

$$E\left\langle W_{l+1}^a \right\rangle = E\left\langle W_l^a \right\rangle - 2\mu_B \Pi_N \left(R_{xx}^a E\left\langle W_l^a \right\rangle - P_{xd}^a \right) \tag{20}$$

This is just the blocked and augmented analog of (5) and the projection operator and the quantity in parentheses is the augmented form of the gradient (2). To find the optimum weight vector, we set the gradient to 0, but we note that this imposes no constraint on the augmented components of the gradient since Π_N already forces them

to zero. All that is required is $P_{xd} = R_{xx}W^*$. That is, the projection operator Π_N guarantees that W^{*a} is W^* (the Weiner optimum weight vector) padded with $L-1$ zeros, which is precisely what we desire. If Π_N were not present, as is done in Mansour and Gray's unconstrained filter [15,16], setting the gradient to zero would also require that

$$E\left\langle X_c^T X_l \right\rangle W^* = E\left\langle X_c^T D_l \right\rangle \tag{21}$$

which is only true if $D_l = X_l W^* + \Psi$ where Ψ represents samples of a white gaussian noise sequence. This will be true if $\{x_j\}$ and $\{d_j\}$ are related by a simple FIR transfer function whose order G is less than or equal to N. Provided this is the case, we can delete Π_N which also allows us to drop two FFTs. The only drawback is that while the last $L-1$ points of W^a start out as 0, gradient noise will allow small non-zero quantities to enter in which will lead to a slightly higher mean square error (MSE). When $M \geq G > N$, $N-G$ of the weights will tend toward non-zero values and the filter will be able to achieve a lower mean-square error than a constrained N weight filter. The lower MSE comes at the expense of introducing circular correlation products; however, if $N-G$ is small with respect to L this may be tolerable. The amount of error introduced has yet to be studied extensively. The same is true when the transfer function between $\{d_n\}$ and $\{x_n\}$ is IIR where the unconstrained filter will attempt to use all M weights. The unconstrained filter also fails when it is configured as a line enhancer where $x_j = z^{-\delta} d_j$. Because the filter is unconstrained it can find the non-causal solution for which the mean square error is zero – unfortunately, this is a useless solution.

While we could continue to pursue the mean convergence of augmented blocked algorithm, it is exactly the same as that of the blocked algorithm. What is important to note here is that, although the above expectations have explicitly determined the augmented portions of P_{xd}^a and R_{xx}^a, these portions contribute nothing to $E\left\langle W_{l+1}^a \right\rangle$ as a result of the projection Π_N and the zero padding of W_l^a (also due to Π_N). This will later allow us to augment R_{xx} in such a way that R_{xx}^a is easily invertible.

As we saw in Section 1, we can both speed up and orthogonalize the convergence of the LMS adaptive filter by replacing μ with μR_{xx}^{-1}. Writing (8) using augmented vectors consistent with those we have already defined gives

$$W_{l+1}^a = W_l^a + 2\frac{\mu_B}{L}\Pi_N (R_{xx}^{-1})^a \Pi_N X_l^{aT} E_l^a$$

where $(R_{xx}^{-1})^a$ is an $M \times M$ augmented matrix whose upper left corner is composed of the $N \times N$ matrix R_{xx}^{-1}. The remaining elements of the matrix are arbitrary. Pre-and post-multiplying $(R_{xx}^{-1})^a$ by Π_N guarantees that they do not enter into the computation. Now, if we again take expectations over an ensemble, we will have

$$E\left\langle W_{l+1}^a \right\rangle = E\left\langle W_l \right\rangle - 2\mu_B \Pi_N (R_{xx}^{-1})^a \Pi_N \left(R_{xx}^a E\left\langle W_l \right\rangle - P_{xd}^a\right) \tag{22}$$

But,

$$\begin{aligned} \Pi_N (R_{xx}^{-1})^a \Pi_N R_{xx}^a &= \Pi_N \\ &= \Pi_N (R_{xx}^a)^{-1} R_{xx}^a \end{aligned}$$

and, while we have $\Pi_N P_{xd}^a = \Pi_N R_{xx}^a W^{*a}$, since the augmented portion of P_{xd}^a is not

constrained, we can let $P_{xd}^a = R_{xx}^a W^{*a}$. With this we have

$$\begin{aligned}\Pi_N(R_{xx}^{-1})^a\Pi_N P_{xd}^a &= \Pi_N(R_{xx}^{-1})^a\Pi_N R_{xx}^a W^{*a} \\ &= \Pi_N W^{*a} \\ &= \Pi_N(R_{xx}^a)^{-1}R_{xx}^a W^{*a} \\ &= \Pi_N(R_{xx}^a)^{-1}P_{xd}^a\end{aligned}$$

Thus, *in the mean*, $\Pi_N(R_{xx}^{-1})^a\Pi_N$ can be replaced by $\Pi_N(R_{xx}^a)^{-1}$ and the update equation becomes

$$W_{l+1}^a = W_l^a + 2\frac{\mu_B}{L}\Pi_N(R_{xx}^a)^{-1}X_l^{aT}E_l^a \tag{23}$$

Now, we pre-multiply with $\sqrt{M}\mathcal{F}$ and insert $\mathcal{F}^*\mathcal{F} = I$ at appropriate points. Thus,

$$\mathcal{W}_{l+1}^a = \mathcal{W}_l^a + \mathcal{F}\Pi_N\mathcal{F}^*2\frac{\mu_B}{L}\mathcal{F}(R_{xx}^a)^{-1}\mathcal{F}^*\mathcal{F}X_l^{aT}\mathcal{F}^*\mathcal{E}_l^a$$

where $\mathcal{W}$ and $\mathcal{E}$ represent DFTs of W^a and E^a respectively. Next, if R_{xx}^a can be made circulant, $\mathcal{F}$ will diagonalize it. That is $\Lambda^a = \mathcal{F}R_{xx}^a\mathcal{F}^*$ and $\Lambda^{a-1} = \mathcal{F}(R_{xx}^a)^{-1}\mathcal{F}^*$. With this the update equation becomes

$$\mathcal{W}_{l+1}^a = \mathcal{W}_l^a + \mathcal{F}\Pi_N\mathcal{F}^*2\frac{\mu_B}{L}\Lambda^{a-1}\mathcal{F}X_l^{aT}\mathcal{F}^*\mathcal{E}_l^a \tag{24}$$

This results in the structure shown in Figure 3. This filter [28] is exactly the same as the fast implementation of the block LMS adaptive filter shown in Figure 2, except μ

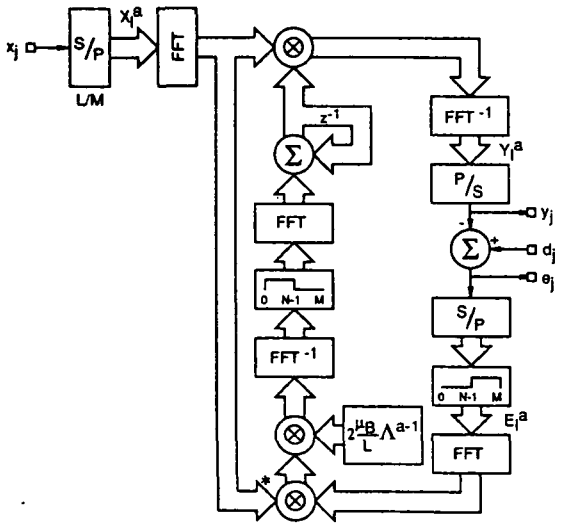

Figure 3: The Block Frequency Domain LMS Algorithm with a Vector μ

has been replaced by $\mu(\Lambda^a)^{-1}$. Since this simply represents a point-by-point multiply of two vectors, we see that this algorithm requires no more multiplies nor adds to implement than does Figure 2[1]. The only additional computational load arises in estimating $(\Lambda^a)^{-1}$.

[1] We need to note that Figure 2 requires N more real multiplies than did Figure 1.

As we indicated above, we must augment R_{xx} in such a way as to make R^a_{xx} circulant. If we let $r_n = r_{-n} = E\langle x_k x_{k+n}\rangle$ represent samples of the autocorrelation of $x(n)$, then

$$R_{xx} = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{N-1} \\ r_1 & r_0 & r_1 & \cdots & r_{N-2} \\ r_2 & r_1 & r_0 & \cdots & r_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{N-1} & r_{N-2} & r_{N-3} & \cdots & r_0 \end{bmatrix}$$

Next, following the approach used by Picchi and Prati in [20], we define R^a_{xx} as a circulant matrix whose first column is given as

$$[r_0, r_1, r_2, \ldots, r_{N-1}, r_{N-1}, r_{N-2}, \ldots, r_2, r_1]$$

and whose successive columns are given by circularly down-shifting each preceding column by one sample. It is easy to see that the resulting $2N-1 \times 2N-1$ matrix contains the $N \times N$ matrix R_{xx} in its upper left corner. Defining R^a_{xx} this way imposes restrictions on L, namely we require $L = N$. L can be chosen greater than N by defining the first column of R^a_{xx} and inserting $L-N$ zeros as follows:

$$[r_0, r_1, r_2, \ldots, r_{N-1}, 0, \ldots, 0, r_{N-1}, r_{N-2}, \ldots, r_2, r_1]$$

However, the only way L can be smaller than N is if $r_n = 0 \ \forall\, n \geq G$, for some $G \leq N$. In this case the first column is given as

$$[r_0, r_1, r_2, \ldots, r_{G-1}, 0, \ldots, 0, r_{G-1}, r_{G-2}, \ldots, r_2, r_1]$$

In the above there are $G-N$ zeros, so the overall length is $M = N+G-1$, and thus $L = G$.

Provided we can satisfy these restrictions on N and L we can easily diagonalize R^a_{xx} using the DFT. That is,

$$\lambda^a_i = \sum_{n=-\frac{M}{2}}^{n=\frac{M}{2}} r_n e^{-j\frac{2\pi ni}{M}} \tag{25}$$

where λ^a_i are just the diagonal elements of Λ^a. From this expression we see that λ^a_i are just samples of the power spectral density of $\{x_j\}$, $P_{xx}(\omega)$ convolved with the Fourier transform of a rectangular window of length M. Thus, the problem of estimating Λ^a is simply one of spectrum estimation.

A well known method for spectrum estimation is Bartlett's procedure [23]. Using this approach gives

$$\hat{\Lambda}^a = diag\left[\frac{1}{K+1}\sum_{l=0}^{K} \mathcal{X}^a_l \otimes \mathcal{X}^{a*}_l\right] \tag{26}$$

where $\mathcal{X}^a_l$ is the DFT of the first column of X^a_l. That is, at block K, $\hat{\lambda}^a_i$ is just the average over $K+1$ blocks of the power in the ith bin of the DFT of the reference input. $E\left\langle \hat{\lambda}^a_i \right\rangle$ is a sample of $P_{xx}(\omega)$ convolved with the Fourier transform of a triangular window of length M evaluated at $\omega = \frac{2\pi i}{M}$. Although this is not exactly equivalent to

(25), it provides a simple estimator. The estimator given in (26) was first proposed by Ogue et. al. in [19] and later by Picchi and Prati in [20] for use in an adaptive channel equalizer.

For a general adaptive filter, especially in the case of slowly non-stationary inputs, an exponential average might be more appropriate. That is,

$$\hat{\Lambda}_{l+1}^{a} = \frac{1}{1+\beta}\hat{\Lambda}_{l}^{a} + \frac{\beta}{1+\beta}\, diag\left[\mathcal{X}_{l}^{a} \otimes \mathcal{X}_{l}^{a*}\right] \tag{27}$$

where β is chosen to reflect the rate of non-stationarity. We need to be careful that any of the $\hat{\lambda}_{i}^{a}$ do not become too small or they will cause the digital filter to overflow when inverted. One way to prevent this is to constrain them to be larger than some lower limit. This is equivalent to adding a noise floor to x before the FFT is taken leading to the $\hat{\Lambda}$ estimate but it doesn't require as much computation.

In summary, we have found a way to use the FFT to easily estimate the eigenvalues of R_{xx} and used this to orthogonalize the adaptive process. If we had taken a purely heuristic approach, viewing the frequency-domain filter as a set of M independent one-weight adaptive filters operating on the bins of M-point DFTs of $x(n)$ and $d(n)$ we would have arrived at the same estimates as (26) and (27). That is, the adaptive gain of each filter is determined only by the power in the associated bin. Of course, adjacent bins of the DFT are not necessarily independent of each other. Hodgkiss and Nolte [4] argued that the covariance between two Fourier series coefficients of a process is approximately zero if the power spectral density of the process is relatively smooth. This argument can be extended to the variance between DFT coefficients, since these are just aliased Fourier series coefficients [24]. In this context, adjacent DFT coefficients are approximately independent of each other if the power spectral density changes slowly over the bandwidth of the DFT bins which will be the case if the process contains no isolated periodic components and we choose M large enough.

These arguments correspond to arguments made above. If N, and hence M, is greater than G, that is, if $r_n = 0 \forall n \geq G$, then the rectangular window disappears and the $\hat{\lambda}_{i}^{a}$ in (25) are exactly samples of $P_{xx}(\omega)$. Further, the difference between using a rectangular window and a triangular one diminishes as M increases and the estimate in (26) approaches that in (25).

We also note at this point that the filter in Figure 3 degenerates to Narayan and Peterson's Transform Domain filter (TDF) [11,17] if the overlap is set so that at each iteration only one output point is produced. That is $L = 1$ and $M = N$. In this case, the inverse DFT used to generate Y_{l}^{a} only produces one usable point. Since this one point is the first point in the vector, its computation only involves the average of the points in $\mathcal{Y}_{l}$, by definition of the DFT. Although the TDF requires more computation than the LMS adaptive filter, it has the advantage of proportional convergence. Narayan [17] also proposed other transforms including the DCT to be used in place of the DFT.

Another frequency-domain adaptive filter is the Dentino filter [8]. The Dentino filter simply implements (8) using the DFT to orthogonalize the input data. It ignores the fact that the DFT actually implements a circular convolution, so none of the vectors are augmented and there are no projection operations. Though the filter output suffers from circular convolution products, occasionally the meaningful information can be extracted from the converged weight vector. Maiwald et. al. [7] observed that for the

case where the input is truly cyclic with period equal to the length of the filter, as in an adaptive equalizer operating on a repeated training pulse, the frequency-domain weights of an M-point Dentino filter converge to the DFT of the optimal M-point transversal filter. After the weights have converged, they may be used to process data using standard overlap and-save-techniques. This works well in an equalizer where no output is generated during the training sequence. Another example of the Dentino filter used simply for the information in the weights is given by Reed et. al. [22], who use the filter to perform bearing estimation.

3 Proportional Convergence

In this section we present the results of two experiments that illustrate the proportional convergence of the general frequency-domain adaptive filter. In the first experiment we present a signal consisting of two sinusoids to to a general 32-weight general frequency domain adaptive filter configured as an adaptive line enhancer. In this configuration, $x_j = z^{-\delta}d_j$. This configuration is commonly used to improve signal to noise ratios in noisy "single microphone" data. The filter will enhance input signals that have a strong autocorrelation, while rejecting those whose autocorrelation is narrower than δ. The first sinusoid has a frequency 0.25 times the sample frequency and the second has a frequency of 0.3125 times the sample frequency. This puts the first sinusoid in the 17th bin of a 64 point FFT, and the second in the 21st bin. The first sinusoid has an amplitude of 2.5 and the second 1.25 which gives a two to one relationship in amplitude. We also set the first 128 samples of the signal to zero. Thus we will have some zero output points before the sinusoids reach the filter. Finally, we analyze the output data by looking at the average magnitude in the two processed sinusoids as a function of output points. This is done for two cases. The first is the scalar-μ case, which is implemented with a vector-μ containing all ones. The second uses a vector-μ tailored to the power in the two sinusoids. It has the 17th and 21st points (as well as the symmetric points) in the vector-μ set to 0.000156 and 0.000626 respectively which is based on the inverse of the average power in these bins. All the other points in the vector-μ are set to 0.002. The power in these bins is actually very close to zero, which would ordinarily result in a much larger μ at these points, but this is undesirable since it may cause the filter to overflow. Instead we simply limit the minimum bin power. The result of these two experiments are shown in Figure 4. The two thin lines represent the scalar-μ case, while the two thicker ones represent the output of the vector-μ case. The μ was chosen for the vector-μ case so that the higher-power sinusoid was resolved at the same rate as in the scalar-μ case. We can easily see that in the scalar-μ case the lower-powered frequency was not resolved at the same rate as the higher-powered one. However, the vector-μ allows the filter to follow a much more proportional convergence.

For the second demonstration we use a frequency domain filter configured as a canceler. We apply speech spectrum noise to the reference (x_j) input and the same noise filtered through a 32 point FIR filter applied to the desired (d_j) input. This is a standard system identification problem. To minimize the mean square error, the adaptive filter weights will converge to the same values as the system filter. Using speech spectrum noise will give a disparate set of eigenvalues in R_{xx}. In this case we analyze both the average spectral energy in the error output as well as the mean square

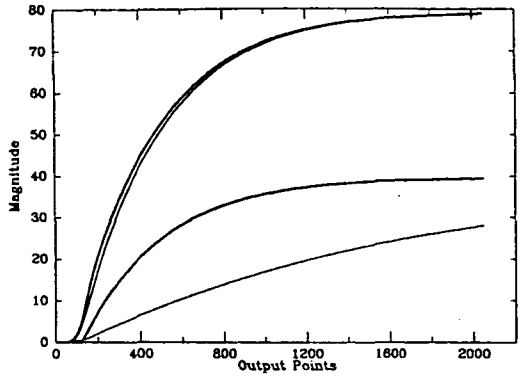

Figure 4: Output of a Two Sinusoid Enhancer

error. As before, we do this for the case of a scalar-μ, as well as a vector-μ tailored to the spectral energy in the reference input. This latter was chosen by inverting the average power in the bins of a 64-point FFT over several overlapping 32 point sections of the reference input. Analyzing the mean squared error in this case of a scalar-μ gives the result shown in Figure 5. This is probably more accurately described as a 'smoothed'

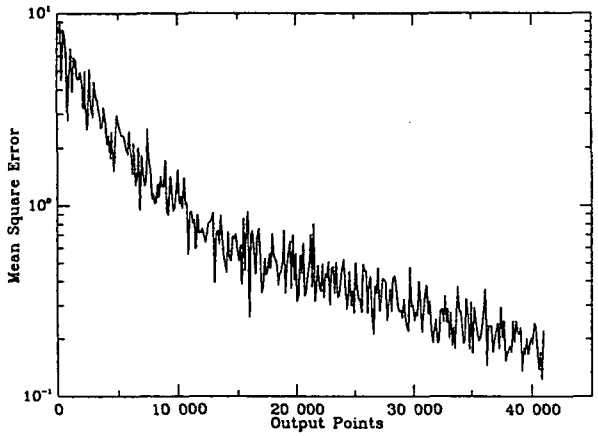

Figure 5: Mean Squared Error for a Scalar μ

square error curve since it is generated by squaring the error and then smoothing it by averaging each point with a few neighboring points. By plotting this against a logarithmic scale, we see at least two different rate of convergence. This illustrates the fact that the filter resolves the weights associated with the larger eigenvalues of R_{xx} first. Having only a scalar-μ limits the convergence rates of the weights associated with the smaller eigenvalues.

Figure 6 shows a series of 4096 point spectral averages of the error output of the filter. Each average is based on 5 4096 point FFTs. The thin line represents the spectral energy in the reference input and each successively thicker line represents averages starting at 12800, 44800, and 172800 points into the output file respectively. We should point out that the mean square error plot stops about where the third trace begins. If we were to continue the mean square error curve we should be able to identify several more progressively decreasing rates of convergence.

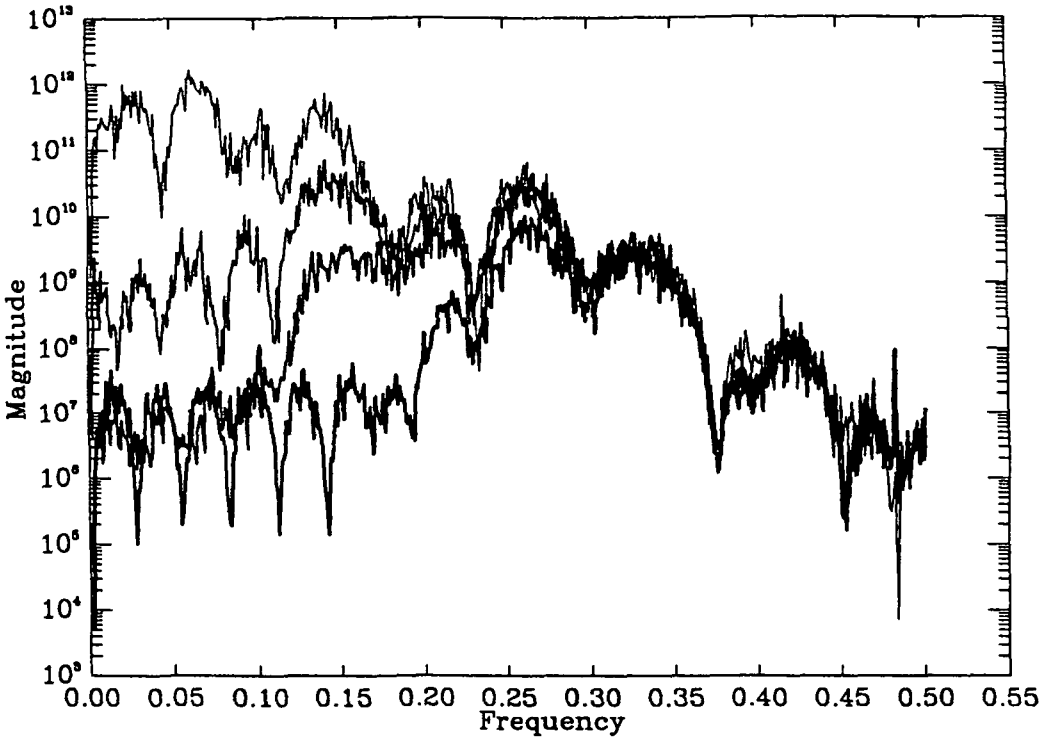

Figure 6: Scalar μ Processing

By comparison, the mean square error and spectral average plots for a vector-μ are given in Figures 7 and 8 respectively. In Figure 8 the averages are taken beginning

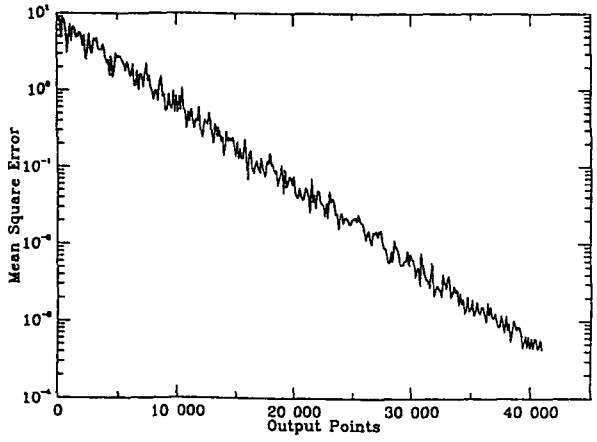

Figure 7: Mean Squared Error for a Vector μ

at 12800, 38400, and 128000 points into the output file respectively. We see from both these figures that the vector-μ gives us a much more proportional convergence.

Both these demonstrations dealt with stationary problems, and used pre-determined vector-μs. For non-stationary problems the vector-μ could be estimated using an exponential average as in (27). Again, as we pointed out in the first demonstration,

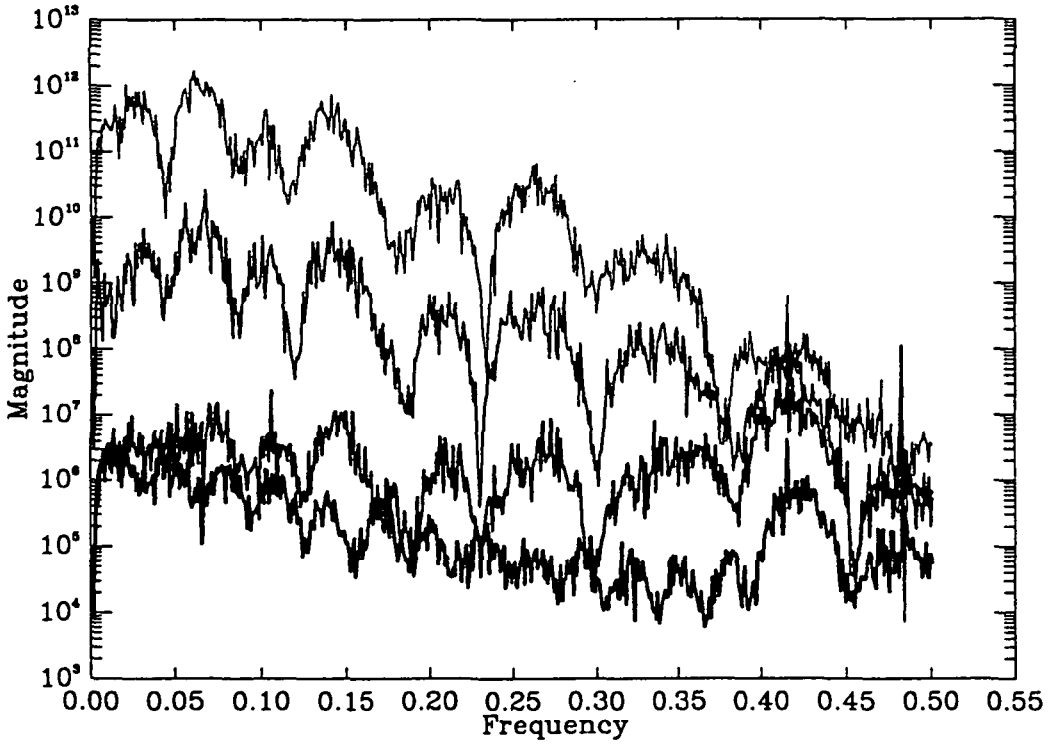

Figure 8: Vector μ Processing

occasionally we have to limit the largest values in the vector-μ, because even though the *average* power in those bins will be very small, a very large corresponding vector-μ element may cause the filter to overflow.

We also need to guard against choosing the block length too large since, as explained in Section 2, this may slow the overall convergence of the filter. Also, the filter will be less able to track any non-stationarities in the input. On the other hand, choosing longer FFTs allows us to better estimate Λ^a and provide more proportional convergence. One way to balance these two requirements is to increase the number of points overlapped in each section. This decreases the number of points output at each iteration and increases the computational overhead but provides the advantage of more proportional convergence.

4 A Frequency-Domain IIR Adaptive Filter

Next, we consider extending the frequency-domain techniques developed for the transversal adaptive filter to the recursive (or IIR) filter. This will include both the application of the DFT to perform fast block IIR convolution and the use of a vector-μ to make the adaptive process converge proportionally. First we present the LMS IIR adaptive filter as it was first developed in 1975 by White [31]. We will not concern ourselves here with questions of stability beyond those addressed by White.

In the recursive filter, the output is a weighted sum not only of the current and past inputs but also of past outputs.

$$y_j = \sum_{i=0}^{N_a-1} a_i x_{j-i} - \sum_{i=0}^{N_b-1} b_i y_{j-i-1} \tag{28}$$

The z-transform of the transfer function of the IIR filter is simply

$$\frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N_a-1} a_i z^{-i}}{1 + \sum_{i=0}^{N_b-1} b_i z^{-i-1}} \tag{29}$$

As with the transversal LMS adaptive filter, we form an error sequence $\{e_j\}$ using the difference between the output $\{y_j\}$ and a desired signal $\{d_j\}$. Then we use the expected value of the square of the error as a performance criteria upon which to base decisions about how to adjust the forward and backward weights, a_i and b_i respectively. Unfortunately, the recursive nature of the equations make analyzing $E\left\langle e_j^2 \right\rangle$ the way we did with the FIR filter nearly impossible. Instead we simply proceed with the implementation. As before, we approximate $E\left\langle e_j^2 \right\rangle$ with e_j^2, then we find the gradient of the squared error with respect to the weights and use this to adjust the weights. First, we observe $\nabla(e_j^2) = 2e_j \nabla e_j$ and

$$\nabla e_j = \nabla(d_j - y_j) = -\left[\frac{\partial y_j}{\partial a_0}, \frac{\partial y_j}{\partial a_1}, \cdots, \frac{\partial y_j}{\partial a_{N_a-1}}, \frac{\partial y_j}{\partial b_0}, \cdots, \frac{\partial y_j}{\partial b_{N_b-1}}\right]^T$$

Now we need to find $\frac{\partial y_j}{\partial a_i}$ and $\frac{\partial y_j}{\partial b_i}$. But

$$\frac{\partial y_j}{\partial a_i} = \frac{\partial}{\partial a_i}\left(\sum_{k=0}^{N_a-1} a_k x_{j-k} - \sum_{k=0}^{N_b-1} b_k y_{j-k-1}\right) \tag{30}$$
$$= x_{j-i} - \sum_{k=0}^{N_b-1} b_k \frac{\partial y_{j-k-1}}{\partial a_i}$$

and similarly,

$$\frac{\partial y_j}{\partial b_i} = -y_{j-i} - \sum_{k=0}^{N_b-1} b_k \frac{\partial y_{j-k-1}}{\partial b_i} \tag{31}$$

Thus the gradient estimate may itself be computed recursively. With suitable adaptive gains μ_f and μ_b, the weight update equations are written

$$a_{i_{j+1}} = a_{i_j} + 2\mu_f e_j \frac{\partial y_j}{\partial a_i} \tag{32}$$

$$b_{i_{j+1}} = b_{i_j} + 2\mu_b e_j \frac{\partial y_j}{\partial b_i} \tag{33}$$

We observe that we can reduce the number of computations required to implement a recursive filter simply by performing FFT implemented block convolution on the forward part of the filter and then computing the feedback contribution a sample at a time for each sample in the block. However, our object here is to find a way to bring to bear the power of the FFT in order to reduce the number of computations in the feedback portion of the filter as well as the feedforward. Several ways to perform recursive convolution in a block fashion have been developed [36,38,39,40,41]. We will use an approach first proposed by Voelcker and Hartquist [37], but we will follow a simpler development and make an additional observation. If we consider (28) it first appears that there is no obvious way to perform block recursive convolution when the block length L is greater than 1, since the feedback portion of the filter will require inputs before they have been computed. This is true unless the backward weights $b_0, b_1 \ldots, b_{L-1}$ (with $L < N_b$) are identically zero. It would seem that this requirement would limit the types of problems to which such a filter structure could be applied. However, we will show that it is possible to add extra poles to the transfer function in such a way as to set b_0 through b_{L-1} to zero. The effect of the extra poles can be compensated by adding equivalent zeros to the transfer function. To formalize these ideas, let us adopt a slightly different notation. First, we rewrite the filter equation as

$$y_j = \sum_{i=0}^{N_a-1} a_i x_{j-i} - \sum_{i=0}^{N_b-1} b_i y_{j-i-L} \tag{34}$$

where we now represent the first non-zero backward weight as b_0. Then, the transfer function becomes

$$\frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N_a-1} a_i z^{-i}}{1 + z^{-L}\sum_{i=0}^{N_b-1} b_i z^{-i}} \tag{35}$$

To implement the feedback portion of this filter using block methods, the largest block that can be computed at a time will be L points long. Suppose we want to replace a recursive filter with N_p forward weights $(p_0, p_1, \ldots, p_{N_p-1})$ and N_q backward weights $(q_0, q_1, \ldots, q_{N_q-1})$ with a block filter that has a feedback block length L. Both systems will have to be equivalent, that is

$$\frac{\sum_{i=0}^{N_p-1} p_i z^{-i}}{1 + z^{-1} \sum_{i=0}^{N_q-1} q_i z^{-i}} = \frac{\sum_{i=0}^{N_a-1} a_i z^{-i}}{1 + z^{-L} \sum_{i=0}^{N_b-1} b_i z^{-i}} \tag{36}$$

The order of the blocked denominator is $N_b + L - 1$, while that of the unblocked denominator is N_q, so we will have to add N_e poles and zeros to the unblocked system where

$$N_e = N_b + L - N_q - 1 \tag{37}$$

That is, we need to find $(e_0, e_1, \ldots, e_{N_e-1})$ and $(b_0, b_1, \ldots, b_{N_b-1})$ such that

$$\left(1 + z^{-1} \sum_{i=0}^{N_q-1} q_i z^{-i}\right) \left(1 + z^{-1} \sum_{i=0}^{N_e-1} e_i z^{-i}\right) = 1 + z^{-L} \sum_{i=0}^{N_b-1} b_i z^{-i} \tag{38}$$

Let us rewrite this equation as

$$(1 + z^{-1}Q(z))E(z) = 1 + z^{-L}B(z) \tag{39}$$

where $Q(z) = \sum_{i=0}^{N_q-1} q_i z^{-i}$, $B(z) = \sum_{i=0}^{N_b-1} b_i z^{-i}$ and $E(z) = 1 + z^{-1} \sum_{i=0}^{N_e-1} e_i z^{-i})$. Then we write

$$E(z) = 1 + z^{-L}B(z) - z^{-1}Q(z)E(z) \tag{40}$$

If we recognize that (40) describes $E(z)$ as the output sequence of an all-pole recursive filter with input sequence $1+z^{-L}B(z)$, we can see that the first L points of the sequence $(1, e_0, \ldots, e_{N_e-1})$ correspond to the first L points of the impulse response of an all-pole filter with transfer function $H(z) = 1/(1+z^{-1}Q(z))$, which is just the feedback portion of the filter we are trying to model. Since $Q(z)$ is general, we see that $N_e + 1 \geq L$. To keep $E(z)$ finite, we must choose $B(z)$ to clear the filter. Conceptually we can do this by setting b_j to the negative of the output of the filter e_{L+1+j} for $0 \leq j \leq N_q - 1$. This will introduce N_q zeros into the filter after which, provided there is no further input (ie: $b_j = 0 \; \forall j > N_q - 1$), the filter produces no further output. This would lead to $N_b = N_q$ and $N_e = L - 1$. We could let $N_e > L - 1$ by not chosing the $B(z)$ to immediately clear the filter. This gives us some flexibility in choosing the initial terms of $B(z)$.

We can also make these same observations by expanding the left side of (38) and

equate the coefficients of the two polynomials. In matrix notation this is

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ q_0 & 1 & 0 & \cdots & 0 \\ q_1 & q_0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{N_e-1} & q_{N_e-2} & q_{N_e-3} & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{N_q-1} & q_{N_q-2} & q_{N_q-3} & \cdots & q_{N_q-N_e-2} \\ 0 & q_{N_q-1} & q_{N_q-2} & \cdots & q_{N_q-N_e-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & q_{N_q-1} \end{bmatrix} \begin{bmatrix} 1 \\ e_0 \\ e_1 \\ \vdots \\ e_{N_e-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ b_0 \\ b_1 \\ \vdots \\ b_{N_b-1} \end{bmatrix} \tag{41}$$

We note that the matrix is $N_q + N_e + 1$ by $N_e + 1$ and that the rightmost vector has $N_b + L$ elements which equals $N_q + N_e + 1$ using (37). Next, we observe that only the first $N_e + 1$ rows of the equation are needed to solve for $(e_0, e_1, \ldots, e_{N_e-1})$. Also, since the determinant of the submatrix formed from the first $N_e + 1$ rows of the matrix is 1, a unique solution exists. If $L > N_e + 1$ however, rows $N_e + 2$ through row L of the matrix equation will not, in general, satisfy the equality. By using (37), this constraint can be written as $N_q > N_b$, that is, the equation is guaranteed to have a solution if $N_b \geq N_q$. If $N_b = N_q$, $(b_0, b_1, \ldots, b_{N_b-1})$ are explicitly determined, but in the case that $N_b > N_q$, $(b_0, \ldots, b_{N_b-N_q-1})$ can be chosen arbitrarily[2]. Also notice from (41) that L can be chosen larger than N_q.

Thus we can determine the e_j either by using matrix methods on the first $N_e + 1$ rows of (41) or, as we observed above, the first N_e points of the impulse response of a recursive system whose feedforward weights are given by the first $N_e + 1$ elements of the solution vector and whose feedback weights are given by the first $N_e + 1$ elements of the first column of the matrix excluding the leading 1.[3] The undetermined b terms are then found by performing the matrix operations indicated in the last N_q rows of (41).

Once the terms $(e_0, e_1, \ldots, e_{N_e-1})$ have been determined, the block forward weights $(a_0, a_1, \ldots, a_{N_a-1})$ can be found by adding an identical set of zeros to the system. That is,

$$\sum_{i=0}^{N_a-1} a_i z^{-i} = \left(\sum_{i=0}^{N_p-1} p_i z^{-i} \right) \left(1 + z^{-1} \sum_{i=0}^{N_e-1} e_i z^{-i} \right) \tag{42}$$

Hereafter, we will refer to $1 + z^{-1} \sum_{i=0}^{N_e-1} e_i z^{-i}$ as the auxiliary equation and it's roots as auxiliaries. From this we see that the order of the feedforward portion of the block filter will have order $N_a - 1$ where $N_a = N_p + N_e$. With (37) we have $N_a = N_p + N_b - N_q + L - 1$. Although we can choose L to be smaller than N_b we will achieve greatest efficiency when $L \approx N_b$. In such a case, $N_a \approx 2N_b + N_p - N_q - 1$ and, with the constraint $N_b \geq N_q$, we require $N_a \approx\geq N_b + N_q - 1$. In most cases of interest, therefore, N_a will be larger than N_b. In the case that $N_p \approx N_q$ and, with the fact that implementing the convolution with FFTs requires that N_a and N_b must be powers of 2, then most likely

[2]Other considerations may determine how we select values for these terms.

[3]We note that these two approaches are mathematically equivalent.

we will need $N_a = 2N_b$. Often, it is undesirable to implement a filter that requires two different length FFTs. One approach is to use the larger length FFT for both convolutions but to use twice the amount of overlap in the feedback convolution as in the feedforward. A more efficient approach is based on the observation that a $2N$-point convolution can be performed using two N-point convolutions. That is,

$$y_j = \sum_{i=0}^{N_a-1} a_i x_{j-i} = \sum_{i=0}^{N_a/2-1} a_i x_{j-i} + \sum_{i=N_a/2}^{N_a-1} a_i x_{j-i}$$

Notice that each convolution involves half of the weights and the input to the second convolution is just the input to the first convolution delayed by $N_a/2$. If $N_a/2$ is chosen to be equivalent to the block length L, then the input to the second convolution is the same as the input to the first except that it is delayed by one block. Thus, one FFT can be used for the input and, excluding the initial FFTs required for the weights, only three FFTs per block are required to implement a fixed weight recursive filter. The choice of $L = N_a/2$ forces us to use 50% overlap but, by saving an extra FFT, it is likely the most efficient approach. We also have assumed in the preceding argument that we want to use the same block length for both of the forward and the backward block convolutions. Although it is possible to have a different block length for each convolution, the use of two different block lengths increases the storage requirements of the filter as well as its complexity.

Presuming that we have chosen a suitable block length L which is larger than or equal to the larger of the required number of backward weights and half the required number of augmented forward weights, that is $L \geq N_b$ and $2L \geq N_a$, then we pad N_a to $2L$ and N_b to L by adding zeros to each. From these we form three L-point weight vectors

$$Aa_l = [a_0, a_1, \ldots, a_{L-1}]^T, \quad Ab_l = [a_L, a_{L+1}, \ldots, a_{2L-1}]^T, \quad \text{and} \quad B_l = [b_0, b_1, \ldots, b_{L-1}]^T$$

Now defining the matrix X_l as we did for (11) (but with $N = L$) and the *matrix* Y_l the same way, we can write the block recursion as

$$\vec{Y}_l = X_l Aa_l + X_{l-1} Ab_l - Y_{l-1} B_l$$

where $\vec{Y}_l$ is the lth L point output vector (defined as Y_l in (11). Following the procedure developed in Section 2 we augment these vectors and matrices making Y^a_{l-1} and X^a_l circulant. This gives

$$\begin{bmatrix} \cdot \\ \vec{Y}_l \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ X_l & \cdot \end{bmatrix} \begin{bmatrix} Aa_l \\ 0 \end{bmatrix} + \begin{bmatrix} \cdot & \cdot \\ X_{l-1} & \cdot \end{bmatrix} \begin{bmatrix} Ab_l \\ 0 \end{bmatrix} - \begin{bmatrix} \cdot & \cdot \\ Y_{l-1} & \cdot \end{bmatrix} \begin{bmatrix} B_l \\ 0 \end{bmatrix} \qquad (43)$$

or, simply,

$$\vec{Y}^a_l = X^a_l Aa^a_l + X^a_{l-1} Ab^a_l - Y^a_{l-1} B^a_l$$

Then, applying DFTs we have

$$\vec{Y}^a_l = DFT^{-1} \left\{ \begin{array}{l} DFT([x_{(l-1)L}, \ldots, x_{(l+1)L-1}]) \otimes DFT(Aa^a_l) \\ +DFT([x_{(l-2)L}, \ldots, x_{lL-1}]) \otimes DFT(Ab^a_l) \\ -DFT([y_{(l-2)L}, \ldots, y_{lL-1}]) \otimes DFT(B^a_l) \end{array} \right\} \qquad (44)$$

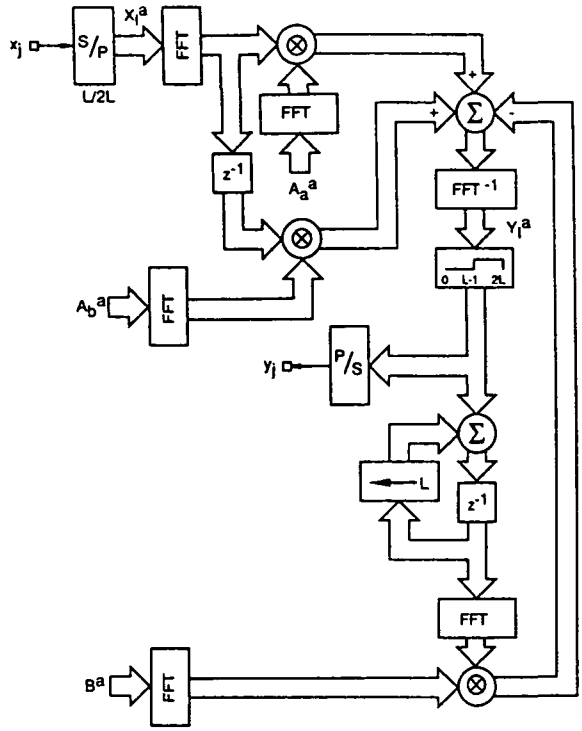

Figure 9: The Block Recursive Filter

which results in the structure diagramed in Figure 9

Next, we need a method to adaptively update the coefficients of the block filter.

As we did for the block FIR filter, we first form an L-point error vector E_l from the difference of the lth output block $\vec{Y}_l$ and the lth set of L points from the desired input, D_l. Then we form the block mean square error and estimate it using only the information available at the lth iteration, yielding

$$\xi_B = E\left\langle \frac{1}{L} E_l^T E_l \right\rangle \approx \frac{1}{L} E_l^T E_l$$

and the block gradient estimate becomes

$$\hat{\nabla}_l = \frac{1}{L} \nabla E_l^T E_l = \frac{-2}{L} (\nabla \vec{Y}_l)^T E_l \tag{45}$$

where

$$(\nabla \vec{Y}_l)^T = \begin{bmatrix} \left(\frac{\partial \vec{Y}_l}{\partial a_0}\right)^T \\ \left(\frac{\partial \vec{Y}_l}{\partial a_1}\right)^T \\ \vdots \\ \left(\frac{\partial \vec{Y}_l}{\partial a_{(l+2)L-1}}\right)^T \\ \left(\frac{\partial \vec{Y}_l}{\partial b_0}\right)^T \\ \left(\frac{\partial \vec{Y}_l}{\partial b_1}\right)^T \\ \vdots \\ \left(\frac{\partial \vec{Y}_l}{\partial b_{(l+1)L-1}}\right)^T \end{bmatrix} = \begin{bmatrix} \frac{\partial y_{lL}}{\partial a_0} & \frac{\partial y_{lL+1}}{\partial a_0} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial a_0} \\ \frac{\partial y_{lL}}{\partial a_1} & \frac{\partial y_{lL+1}}{\partial a_1} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial a_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{lL}}{\partial a_{(l+2)L-1}} & \frac{\partial y_{lL+1}}{\partial a_{(l+2)L-1}} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial a_{(l+2)L-1}} \\ \frac{\partial y_{lL}}{\partial b_0} & \frac{\partial y_{lL+1}}{\partial b_0} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial b_0} \\ \frac{\partial y_{lL}}{\partial b_1} & \frac{\partial y_{lL+1}}{\partial b_1} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial b_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{lL}}{\partial b_{(l+1)L-1}} & \frac{\partial y_{lL+1}}{\partial b_{(l+1)L-1}} & \cdots & \frac{\partial y_{(l+1)L-1}}{\partial b_{(l+1)L-1}} \end{bmatrix} \tag{46}$$

From (34) we can find

$$\frac{\partial y_j}{\partial a_i} = x_{j-i} - \sum_{k=0}^{L-1} b_k \frac{\partial y_{j-k-L}}{\partial a_i} \quad \text{and} \quad \frac{\partial y_j}{\partial b_i} = -y_{j-i-L} - \sum_{k=0}^{L-1} b_k \frac{\partial y_{j-k-L}}{\partial b_i} \tag{47}$$

As in the scalar case, (47) can be solved recursively. By comparing these equations to (34) and (35), we note that the recursions can be computed using block methods directly as diagramed in Figure 10. Then we compute the gradient estimate and update

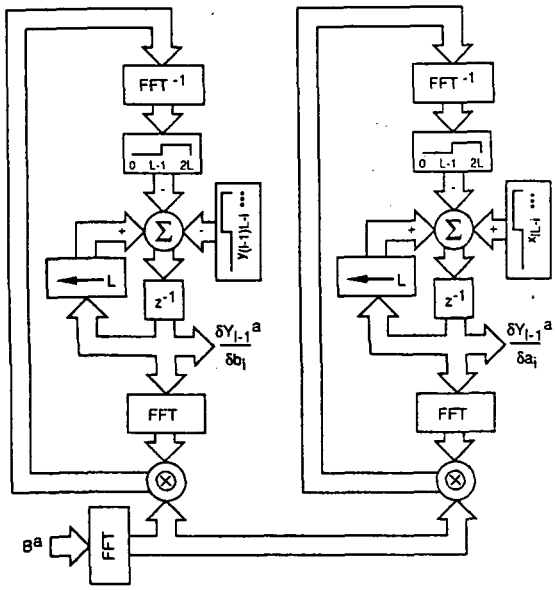

Figure 10: The Block Recursive Filter Weight Partials

the weights

$$a_{i_{l+1}} = a_{i_l} + \frac{2\mu_f}{L} \sum_{k=0}^{L-1} \frac{\partial y_{lL+k}}{\partial a_{i_l}} e_{lL+k} \quad b_{i_{l+1}} = b_{i_l} + \frac{2\mu_b}{L} \sum_{k=0}^{L-1} \frac{\partial y_{lL+k}}{\partial b_{i_l}} e_{lL+k} \tag{48}$$

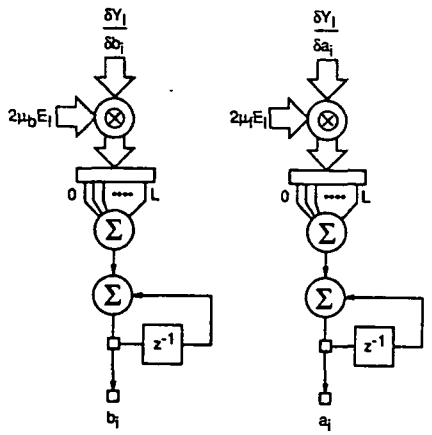

Figure 11: The Block Recursive Filter Weight Update

where μ is the adaptive gain. Thus, the weight update can be performed as shown in Figure 11.

At long last, we have a complete block IIR filter implemented using the FFT to perform fast convolution. To summarize, we present the complete algorithm. To do so, we define the vectors:

$$\begin{aligned} X_l^T &= [x_{lL}, x_{lL+1}, \ldots, x_{(l+1)L-1}] \\ D_l^T &= [d_{lL}, d_{lL+1}, \ldots, d_{(l+1)L-1}] \\ Y_l^T &= [y_{lL}, y_{lL+1}, \ldots, y_{(l+1)L-1}] \\ E_l^T &= [e_{lL}, e_{lL+1}, \ldots, e_{(l+1)L-1}] \\ Aa^T &= [a_0, a_1, \ldots, a_{L-1}] \\ Ab^T &= [a_L, a_{L+1}, \ldots, a_{2L-1}] \\ B^T &= [b_0, b_1, \ldots, b_{L-1}] \end{aligned} \tag{49}$$

Also, we define Γ to be a projection operator that sets the first L points of a vector to zero. With the FFT and it's inverse represented by $\mathcal{F}$ and $\mathcal{F}^{-1}$, we first compute the feedforward contribution and subtract from that the feedback contribution computed at the previous iteration to generate L points of output.

$$\begin{aligned} \mathcal{X}_l^a &= \mathcal{F}[X_{l-1}^T \| X_l^T]^T \\ \mathcal{Y}_l'^a &= \mathcal{X}_l^a \otimes \mathcal{A}a^a + \mathcal{X}_{l-1}^a \otimes \mathcal{A}b^a - \mathcal{V}_{l-1}^a \\ [0^T \| Y_l^T]^T &= \Gamma \mathcal{F}^{-1} \mathcal{Y}_l'^a \\ E_l &= D_l - Y_l \end{aligned} \tag{50}$$

Next we update the forward weights

$$\left.\begin{aligned} \mathcal{G}_{a_i l}^a &= \mathcal{F}[G_{a_i(l-1)}^T \| G_{a_i l}^T]^T \\ \mathcal{Z}_{a_i}^a &= \mathcal{G}_{a_i l}^a \otimes \mathcal{B} \\ [0^T \| Z_{a_i}^T]^T &= \Gamma \mathcal{F}^{-1} \mathcal{Z}_{a_i}^a \\ G_{a_i(l+1)} &= [x_{lL-i}, \ldots, x_{(l+1)L-1-i}]^T - Z_{a_i} \\ a_i &= a_i + \tfrac{2\mu_f}{L} E_l^T G_{a_i(l+1)} \end{aligned}\right\} \forall i\,;\ 0 \le i \le 2L-1 \tag{51}$$

where $G_{a,l} \cong \frac{\partial Y_l}{\partial a_i}$ at each iteration and $\mathcal{Z}$ and Z are intermediate results. We then update the backward weights

$$\left.\begin{array}{l} \mathcal{G}^a_{b_i l} = \mathcal{F}[G^T_{b_i(l-1)} \| G^T_{b_i l}]^T \\ \mathcal{Z}^a_{b_i} = \mathcal{G}^a_{b_i l} \otimes \mathcal{B} \\ [0^T \| Z^T_{b_i}]^T = \Gamma \mathcal{F}^{-1} \mathcal{Z}^a_{b_i} \\ G_{b_i(l+1)} = -[y_{(l-1)L-i}, \cdots, y_{lL-1-i}]^T - Z_{b_i} \\ b_i = b_i + \frac{2\mu_b}{L} E_l^T G_{b_i(l+1)} \end{array}\right\} \forall i \,;\, 0 \le i \le L-1 \qquad (52)$$

where $G_{b,l} \cong \frac{\partial Y_l}{\partial b_i}$ at each iteration. Finally, we transform the weights back into the frequency domain and compute the feedback contribution for the next iteration:

$$\begin{array}{l} \mathcal{A}a = \mathcal{F}[Aa^T \| 0^T]^T \\ \mathcal{A}b = \mathcal{F}[Ab^T \| 0^T]^T \\ \mathcal{B} = \mathcal{F}[B^T \| 0^T]^T \\ \mathcal{Y}^a_l = \mathcal{F}[Y^T_{l-1} \| Y^T_l]^T \\ \mathcal{V}_l = \mathcal{Y}^a_l \otimes \mathcal{B} \end{array} \qquad (53)$$

Equations (50), (51), (52) and (53) constitute a frequency domain IIR adaptive filter.

Now we turn to the question of efficiency. As with the FIR adaptive filter, we restrict our focus to filters whose inputs are real sequences.

With N_p forward and N_q backward weights, a time-domain IIR filter requires $N_p + N_q$ real multiplies and real additions including the error calculation. There are $N_p + N_q$ recursions required to compute the weight updates and each requires $N_q + 1$ multiplies and $N_q + 1$ additions including the sum for the actual weight update. Apart from the recursions we also need two multiplies to form $\mu_f e_j$ and $\mu_b e_j$. Altogether, the time-domain IIR filter requires $(N_p + N_q)(N_q + 2) + 2$ real multiplies and $(N_p + N_q)(N_q + 2)$ real additions for each output point. If we use $2L$ point FFTs in the frequency domain filter, the feedback filter actually implements L feedback weights and $2L$ feedforward weights. An equivalent time-domain filter would use the same number of feedback weights. From (37) using $N_q = L$ gives $N_p = L + 1$. We also need to consider that the frequency domain filter produces L output points at each iteration. To produce L output points the equivalent time-domain filter requires $2L^3 + 5L^2 + 4L$ real multiplies and $2L^3 + 5L^2 + 2L$ real additions.

The filter section of the frequency-domain filter (excluding the weight FFTs) uses 3 FFTs, 3 $\otimes$ operations and one three input $\sum$ operation. Using the same FFT complexity estimates as we used for the frequency-domain FIR filter, the IIR filter section uses $6L\log_2(L) - 2$ real multiplies and $9L\log_2(L) + 18L - 42$ real additions. There are $3L$ recursions of the type in Figure 10. Each uses two FFTs, one $\otimes$ operation and one two input $\sum$ operation. Here, the sum occurs in the time domain and only requires L real additions. So weight partials require $12L^2\log_2(L) - 12L^2 - 6L$ real multiplies and $18L^2\log_2(L) + 21L^2 - 78L$ real additions. There are also $3L$ weight recursions of the type shown in Figure 11. The $\otimes$ operations involve L point real vectors in addition to the multiply by μ_b or μ_f, which can be performed after the $\sum$. Each recursion requires $L + 1$ multiplies and L additions for each of the $3L$ weights. Adding in the cost of the three weight FFTs gives the cost of the weight updates as $3L^2 + 6L\log_2(L) - 9L$ real multiplies and $3L^2 + 9L\log_2(L) + 6L - 36$ real additions. Thus, the overall cost of computing L output points using the block recursive LMS

adaptive filter is $12(L+L^2)\log_2(L) - 9L^2 - 15L - 2$ real multiplies and $24L^2 + 18(L+L^2)\log_2(L) + 90L - 78$ real additions.

Table 2 gives the ratio of frequency-domain to time-domain operations for several choices of L. By comparison, the ratios are a little worse than but still of the same

L	Multiplies	Additions
8	1.376	4.460
16	1.102	2.850
32	0.758	1.731
64	0.480	1.015
128	0.289	0.581
256	0.169	0.327
512	0.096	0.181
1024	0.054	0.099

Table 2: Ratio of Frequency Domain to Time Domain Calculations for the IIR Filters

order as those in Table 1. Here, we see that we need a filter length ≥ 64 before we realize a computational savings. While this filter length may be reasonable for a FIR adaptive filter, one of the reasons for adopting an IIR approach is the hope of shorter filters. However, the use of block methods may allow faster and more proportional convergence even in smaller filters, justifying the added computational expense.

4.1 Proportional Convergence

The forward and backward filters are not strictly independent from each other and we cannot find a simple non-recursive expression for the optimum weights as we did for the FIR adaptive filter. If we take a somewhat heuristic approach and treat the filters as independent from each other, we can apply some of the same ideas that lead to the use of a vector-μ. Following the same development that lead to (24)

$$\begin{aligned} \mathcal{A}a^a_{l+1} &= \mathcal{A}a^a_l + \mathcal{F}\Pi\mathcal{F}^* \tfrac{2\mu_l}{L} \Lambda_x^{a^{-1}} \mathcal{F} \left(E_l^T \nabla_{Aa} \vec{Y}_l\right)^a \\ \mathcal{A}b^a_{l+1} &= \mathcal{A}b^a_l + \mathcal{F}\Pi\mathcal{F}^* \tfrac{2\mu_l}{L} \Lambda_x^{a^{-1}} \mathcal{F} \left(E_l^T \nabla_{Ab} \vec{Y}_l\right)^a \\ \mathcal{B}^a_{l+1} &= \mathcal{B}^a_l + \mathcal{F}\Pi\mathcal{F}^* \tfrac{2\mu_b}{L} \Lambda_y^{a^{-1}} \mathcal{F} \left(E_l^T \nabla_B \vec{Y}_l\right)^a \end{aligned} \tag{54}$$

where we have used $\nabla_{Aa}\vec{Y}_l$ to be the first L columns of $\nabla\vec{Y}_l$ and similarly for ∇_{Ab} and ∇_B. The notation ∇_W indicates the vector gradient with respect to each of the components of W. Π represents a projection operator that selects the first L points of a vector and the the final term in each of the above equations is augmented by padding each with L zeroes. Noting that $\nabla_{Aa} \equiv [G_{a_0(l+1)}, G_{a_1(l+1)}, \ldots, G_{a_{(L-1)}(l+1)}]^T$, $\nabla_{Ab} \equiv [G_{a_L(l+1)}, G_{a_{(L+1)}(l+1)}, \ldots, G_{a_{(2L-1)}(l+1)}]^T$, and $\nabla_B \equiv G_{b_0(l+1)}, G_{b_1(l+1)}, \vdots, G_{b_{(L-1)}(l+1)}]^T$,

we adjust the weight updates in (51) and (52) as follows

$$\left.\begin{aligned} h_{B_i} &= \tfrac{2\mu_b}{L} E_l^T G_{b_i(l+1)} \\ h_{Aa_i} &= \tfrac{2\mu_f}{L} E_l^T G_{a_i(l+1)} \\ h_{Ab_i} &= \tfrac{2\mu_f}{L} E_l^T G_{a_{(i+L)}(l+1)} \end{aligned}\right\} \forall i\ ;\ 0 \le i \le L-1$$
$$\begin{aligned} Aa^a_{l+1} &= Aa^a_l + \Pi \mathcal{F}^{-1} \Lambda_x^{a^{-1}} \mathcal{F}[H^T_{Aa} \| 0^T]^T \\ Ab^a_{l+1} &= Ab^a_l + \Pi \mathcal{F}^{-1} \Lambda_x^{a^{-1}} \mathcal{F}[H^T_{Ab} \| 0^T]^T \\ B^a_{l+1} &= B^a_l + \Pi \mathcal{F}^{-1} \Lambda_y^{a^{-1}} \mathcal{F}[H^T_{B} \| 0^T]^T \end{aligned} \tag{55}$$

where $H_{Aa} = [h_{Aa_0}, h_{Aa_1}, \ldots, h_{Aa_{L-1}}]$ and H_{Ab} and H_B are defined similarly. The Λ's can be estimated as they were in either (26) or (27). An exponential average is preferred for Λ^a_y since it is difficult to predict beforehand the inputs to the backward filter. If we use exponential averages, then

$$\begin{aligned} \hat{\Lambda}^a_{x_{l+1}} &= \frac{1}{1+\beta_f} \hat{\Lambda}^a_{x_l} + \frac{\beta_f}{1+\beta_f} diag\left[\mathcal{X}^a_l \otimes \mathcal{X}^{a^*}_l\right] \\ \hat{\Lambda}^a_{y_{l+1}} &= \frac{1}{1+\beta_b} \hat{\Lambda}^a_{y_l} + \frac{\beta_b}{1+\beta_b} diag\left[\mathcal{Y}^a_l \otimes \mathcal{Y}^{a^*}_l\right] \end{aligned} \tag{56}$$

Since the input to the backward filter is less predictable, one would usually choose β_b to be larger than β_f, and thus provide for a shorter window average. Notice that despite the fact that we have broken the forward convolution into two parts, we still use only one Λ^a_x. Even if we used separate estimators for both forward filters, they would be very similar since, except for a block delay, the inputs are the same to both forward filters. In Section 5 we present the results of a frequency domain IIR filter using both forward and backward vector-μs. Using a vector-μ for the forward filter provides dramatic improvement in the case where the input to the filter is characterized by a large eigenvalue spread. The improvement resulting from using a vector-μ in the backward filter is more difficult to assess but it appears to be useful.

4.2 Feasibility

Although adding auxiliary poles and zeroes allowed us to do the block recursion, and although we showed that we could always find the auxiliary roots, we cannot guarantee that the auxiliary poles and zeros will always lie on top of each other since we separately adjust the forward and backward weights[4]. The plots in Section 5 indicate in fact that they do not begin to coincide until the filter is almost converged. If the auxiliary root happens to lie outside the unit circle and if the auxiliary pole follows a path (in time or space) different from the one the auxiliary zero takes when outside the unit circle, then the filter will become unstable. Another possibility is that the adaptive filter may prevent the pole from moving outside the unit circle if the error surface gradient estimate is steep enough in the direction associated with this motion. Though this avoids instability, it prevents the filter from converging to the optimal solution. We say a solution is *feasible* if the all the poles, including the auxiliaries lie inside the unit circle. Voelcker and Hartquist [37] showed that if L is chosen large enough, all the auxiliaries will fall inside the unit circle. We have noted above that choosing N_b larger than

[4]Even in a fixed filter, roundoff errors will cause the auxiliary poles and zeros not to coincide.

N_q allows $(b_0, \ldots, b_{N_b - N_q - 1})$ to be chosen arbitrarily. We give an example to illustrate that this choice can also affect whether or not a solution is feasible. Suppose we choose $N_b = N_q = L = 3$, in particular a system with poles at $(-.7, 0)$ and $(-.5, \pm.5)$. The denominator of the system function is in this case $1 + 1.7z^{-1} + 1.2z^{-2} + .35z^{-3}$. Setting up (41) for this case

$$\begin{bmatrix} 1 & 0 \\ 1.7 & 1 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \end{bmatrix} = \begin{bmatrix} -1.7 \\ -1.2 \end{bmatrix} \tag{57}$$

and solving for e_0 and e_1 gives that the auxiliary equation is $1 - 1.7z^{-1} + 1.69z^{-2}$ which has roots outside the unit circle at $(0.85, \pm.983616)$. If we set up the same problem, but with $N_b = 4$ we have

$$\begin{bmatrix} 1 & 0 & 0 \\ 1.7 & 1 & 0 \\ 1.2 & 1.7 & 1 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} -1.7 \\ -1.2 \\ b_0 - .35 \end{bmatrix} \tag{58}$$

where we are free to choose b_0. We notice that the choice of b_0 will only affect e_2. e_0 and e_1 will always be -1.7 and 1.69 respectively. Choosing $e_2 = -.75$ gives the auxiliary equation $1 - 1.7z^{-1} + 1.69z^{-2} - .75z^{-3}$ which has roots just inside the unit circle at $(0.464943, \pm 0.870465)$ and $(0.770114, 0)$. To get this value for e_2 requires that we set $b_0 = .433$.

Choosing N_b still larger the N_q gives more degrees of freedom (and also introduces more auxiliaries). Whether this can be done in such a way as to guarantee that all solutions for problems of a particular order will be feasible, or whether an adaptive filter left to choose these "free" feedback weights will always choose a feasible solution if one is available, are questions that require further research.

5 Frequency-Domain IIR Experiments

Here we present the results of an experiment to demonstrate both the frequency-domain IIR filter itself, and the application of proportional convergence to the filter.

In this set of experiments, we color a white noise source using a first order Butterworth filter designed with a cutoff of $0.1 f_{sample}$ and implemented with an impulse invariant transform. This colored noise is then applied to a time-domain IIR filter. The output of this filter is applied to the desired input of a frequency-domain block IIR adaptive filter implemented with 8 forward and 4 backward weights using 16 point FFTs. The colored noise is also applied to the reference input of the frequency-domain filter making this a system identification problem and the time-domain IIR filter the system model. A random number generator forms the white noise source. The coloring filter has one forward and two backward weights which are [0.253195] and $[-1.158046, 0.411241]$, respectively. The model filter has four forward and three backward weights which are $[1, -1, 1, -1]$ and $[-.5, 0.25]$ respectively. In each experiment we have used $\mu = .002$ for both the forward and backward sections of the filter, and at each time we process 2000 eight point blocks, saving the weights every 40th block.

The filter will attempt to converge to the blocked version of the model weights. Here we have $N_p = 4$, $N_q = 3$, and $N_b = L = 4$. By (37), we have $N_e = 4$, so there will be 4 auxiliaries. Also, $N_b - N_q = 1$, so we are free to choose one of the auxiliaries. One

possible set is $[(0.271844, 0), (-.385923, \pm.557571), (0, 0)]$. The poles should converge to $[(.5, \pm.5), (-.5, 0)]$, while the zeros should converge to $[(0, \pm 1), (1, 0)]$.

There are three separate experiments, each using a different set of values for β_f and β_b, which are used to estimate the forward and backward vector-μs as given in (56). If the $\hat{\Lambda}^a$s are initialized to the identity matrix, setting $\beta = 0$ is equivalent to using a constant scalar-μ. In Case A, both β_f and β_b are set to 0. In Case B, we set β_f to 0.03 which gives an exponential average of about 17 blocks. Finally in Case C, we set both β_f and β_b to 0.03. Plots of the pole-zero tracks and final pole-zero positions after 2000 blocks for each case are shown in the following figures. We also plot the forward and backward weights after each 40 output blocks.

We see that Case C, with both a forward and backward vector-μ, converges in 2000 blocks. Case B, with only a forward vector-μ, also has the poles and zeroes converged, but the auxiliaries are still moving. Case A has one zero converged and some other poles and zeroes near convergence. The difference in the forward weight plots between Cases A and B illustrates how the vector-μ improves the convergence, making it more proportional. Comparing the backward weight plots for Cases B and C shows that a backward vector-μ may have caused some improvement, but it is difficult to judge. Still, it appears that the backward vector-μ improved the overall rate of convergence. This is an area where more experimentation needs to be done. As we indicated in Section 4, a larger β_b might be expected to perform better.

6 Summary

We have presented a general frequency-domain FIR adaptive filter that not only requires fewer computations than the time-domain LMS adaptive filter, but also provides faster and more proportional convergence which preserves the signal's spectral structure. We have also shown that the frequency-domain adaptive filters previously proposed by Dentino [8], Clark [9], Ferrara [10], Narayan and Peterson [11,17] and Mansour and Gray [15,16] are special cases of this general frequency-domain adaptive filter. We have extended these ideas to the recursive LMS adaptive filter and shown that it is possible to reduce the number of computations from order L^3 to order $L^2 \log_2(L)$. At the same time, we have presented a way to use the FFT to perform fast IIR convolution through the addition of auxiliary poles and zeros. We have shown that the convergence properties of the recursive LMS adaptive filter can be improved by using an appropriately chosen vector-μ. It is possible to show [29] that we can use these fast convolution techniques to reduce the number of computations in the CHARF [33], SHARF [34], and Feintuch's recursive LMS filter [32]. Both the general frequency-domain adaptive filter and the frequency-domain LMS recursive filter were implemented and tested. The results presented in this paper show the advantage of using a vector-μ.

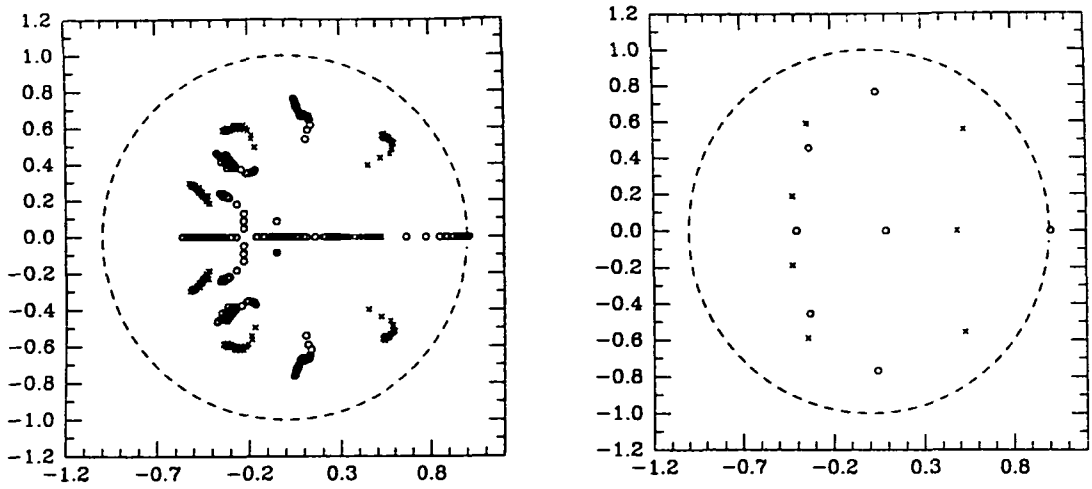

Figure 12: Pole Zero Tracks and Final Pole Zero Plot for Case A

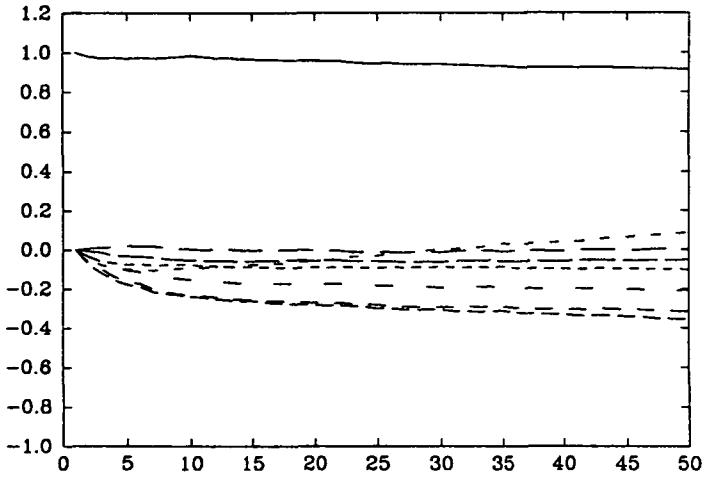

Figure 13: Forward weights for Case A

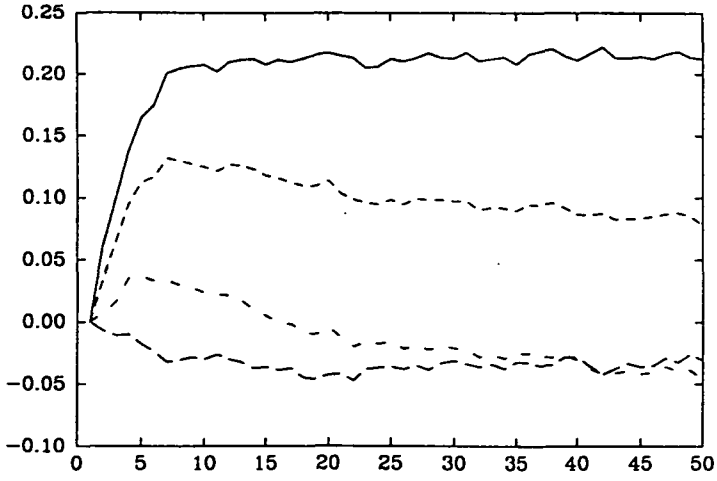

Figure 14: Backward Weights for Case A

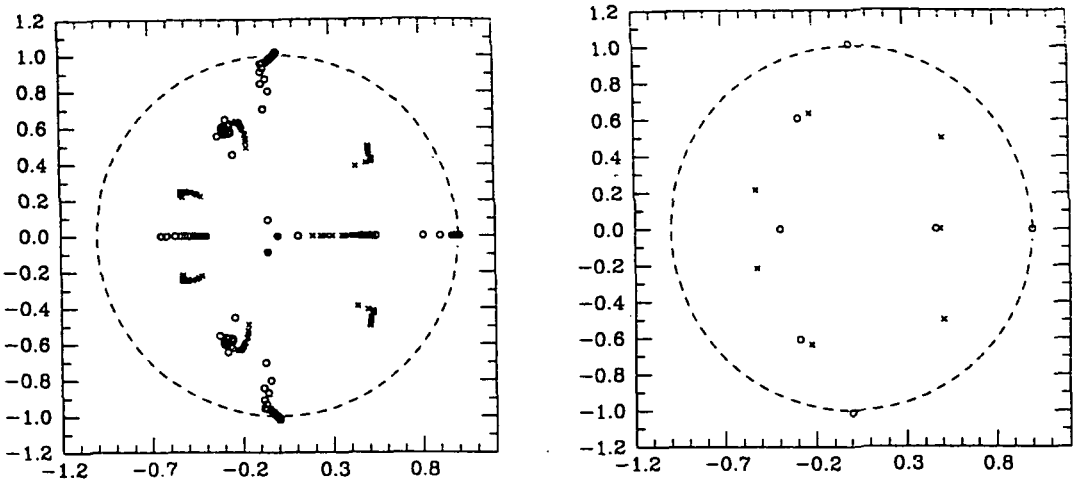

Figure 15: Pole Zero Tracks and Final Pole Zero Plot for Case B

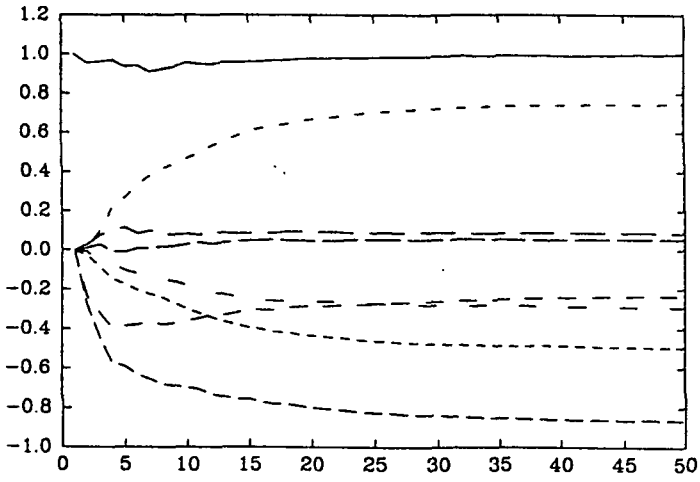

Figure 16: Forward weights for Case B

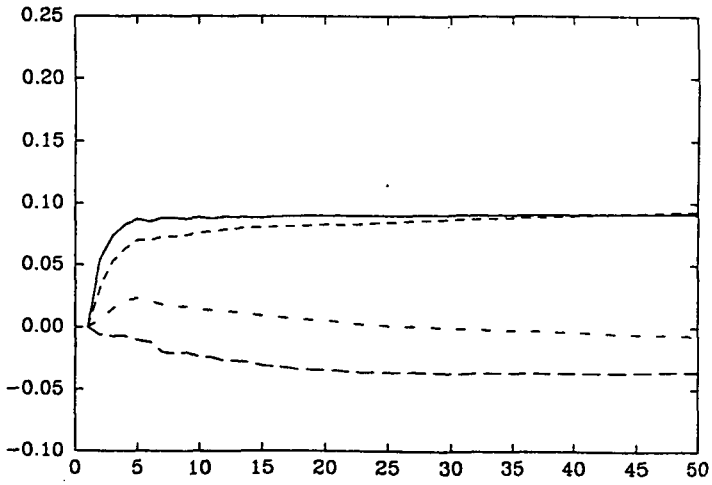

Figure 17: Backward Weights for Case B

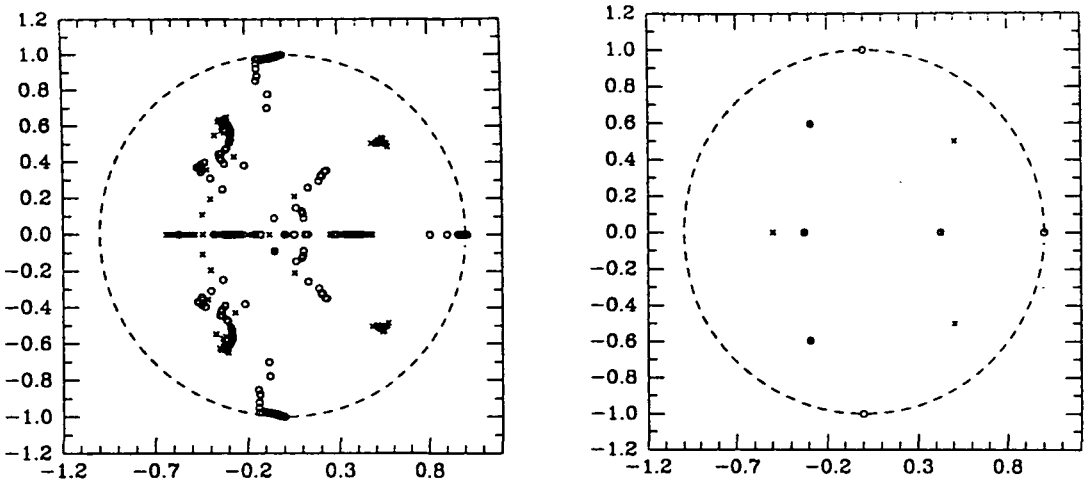

Figure 18: Pole Zero Tracks and Final Pole Zero Plot for Case C

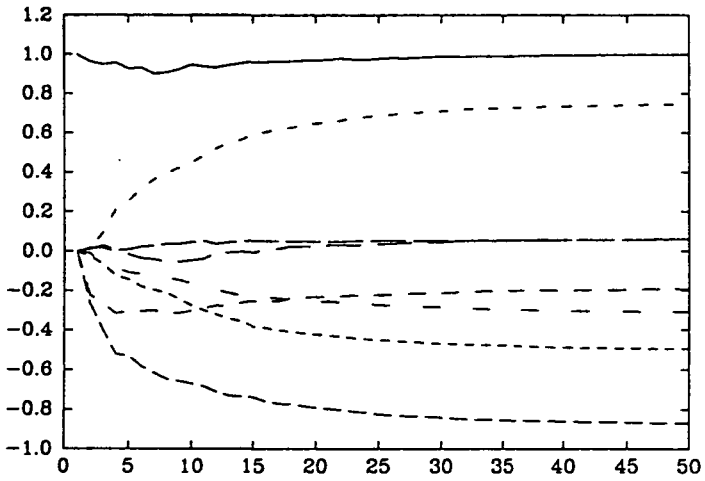

Figure 19: Forward weights for Case C

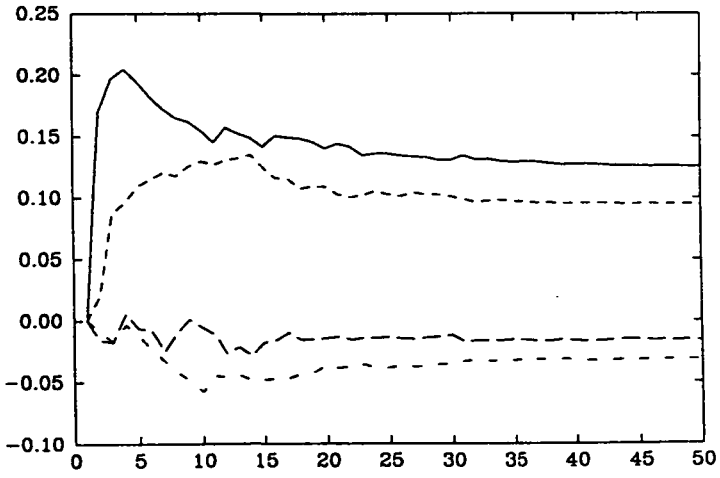

Figure 20: Backward Weights for Case C

References

[1] T. Walzman. *Automatic Equalization and Discrete Fourier Transform Techniques.* PhD thesis, Polytechnic Institute of Brooklyn, June 1971.

[2] R. M. Gray. On the asymptotic eigenvalue distribution of toeplitz matrices. *IEEE Transactions on Information Theory*, 725–730, November 1972.

[3] T. Walzman and M. Schwartz. Automatic equalization using the discrete frequency domain. *IEEE Transactions on Information Theory*, 59–68, Jan 1973.

[4] W. S. Hodgkiss and L. W. Nolte. Covariance between fourier coefficients representing the time waveforms observed from an array of sensors. *Journal of the Acoustical Society of America*, 582–590, March 1976.

[5] B. Widrow, J. M. McCool, M. G. Larimore, and C R Johnson Jr. Stationary and nonstationary learning charisteristics of the LMS adaptive filter. *Proceedings of the IEEE*, 1151–1162, Aug 1976.

[6] R. D. Gitlin and F. R. Magee Jr. Self-orthogonalizing adaptive equalization algorithms. *IEEE Transactions on Communications*, 666–672, July 1977.

[7] D. Maiwald, H. P. Kaeser, and F. Closs. *On Reducing the Number of Operations in Adaptive Equalizers.* Technical Report, IBM research report, Sept 1978.

[8] M. J. Dentino, J. McCool, and B. Widrow. Adaptive filtering in the frequency domain:. *Proceedings of the IEEE*, 1658–1659, Dec 1978.

[9] G. A. Clark, S. K. Mitra, and S. R. Parker. Block adaptive filtering. *Proc IEEE Int Symp Circuits Syst*, 384–387, April 1980.

[10] E. R. Ferrara. Fast implementation of LMS adaptive filters. *IEEE Transactions on Acoustics Speech and Signal Processing*, 474–475, Aug 1980.

[11] S. S. Narayan and A. M. Peterson. Frequency domain least-mean-square algorithm. *Proceedings of the IEEE*, 124–126, Jan 1981.

[12] G. A. Clark, S. K. Mitra, and S. R. Parker. Block implementation of adaptive digital filters. *IEEE Transactions on Acoustics Speech and Signal Processing*, 744–752, June 1981.

[13] R. W. Christensen, D. M. Chabries, and D. W. Lynn. Noise reduction in speech using adaptive filtering. *Journal of the Acoustical Society of America*, S7–S8, April 1982. Supplement 1.

[14] G. A. Clark, S. K. Mitra, and S. R. Parker. Efficient realization of adaptive digital filters in the time and frequency domains. *IEEE International Conference on Acoustics Speech and Signal Processing*, 1345–1348, May 1982.

[15] D. Mansour and A. H. Gray Jr. Performance characteristics of the unconstrained frequency-domain adaptive filter. *Proc 1982 IEEE Int Symp Circuits and Systems*, 695–698, May 1982.

[16] D. Mansour and A. H. Gray Jr. Unconstrained frequency domain adaptive filter. *IEEE Transactions on Acoustics Speech and Signal Processing*, 726–734, Oct 1982.

[17] S. S. Narayan, A. M. Peterson, and M. J. Narisimha. Transform domain LMS algorithm. *IEEE Transactions on Acoustics Speech and Signal Processing*, 609–615, June 1983.

[18] G. A. Clark, S. K. Mitra, and S. R. Parker. A unified approach to time and frequency domain realization of FIR adaptive digital filters. *IEEE Transactions on Acoustics Speech and Signal Processing*, 1073–1083, Oct 1983.

[19] J. C. Ogue, T. Saito, and Y. Hoshiko. A fast convergence frequency domain adaptive filter. *IEEE Transactions on Acoustics Speech and Signal Processing*, 1312–1314, Oct 1983.

[20] G. Picchi and G. Prati. Self-orthogonalizing adaptive equalization in the discrete frequency domain. *IEEE Transactions on Communications*, 371–379, April 1984.

[21] J. C. Lee and C. K. Un. On the interrelationships among a class of convolutions. *IEEE Transactions on Acoustics Speech and Signal Processing*, 1245–1247, Dec 1984.

[22] F. A. Reed, P. L. Feintuch, and N. J. Bershad. The application of the frequency domain LMS adaptive filter to split array bearing estimation with a sinusoidal signal. *IEEE Transactions on Acoustics Speech and Signal Processing*, 61–69, Feb 1985.

[23] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing.* Englewood Cliffs NJ: Prentice-Hall, 1975.

[24] A. Papoulis. *Signal Analysis.* New York: McGraw-Hill, 1977.

[25] H. C. Andrews and B. R. Hunt. *Digital Image Restoration.* Englewood Cliffs NJ: Prentice-Hall, 1977.

[26] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Englewood Cliffs NJ: Prentice-Hall, 1985.

[27] E. R. Ferarra Jr. Frequency-domain adaptive filtering. In C F Cowan and P M Grant, editors, *Adaptive Filters*, Englewood Cliffs NJ: Prentice Hall, 1985.

[28] D. M. Chabries, R. W. Christensen, D. W. Lynn, and G. M. Kenworthy. Adaptive noise suppressor. Oct 1985. Patent # 4,658,426.

[29] D. W. Lynn *Frequency Domain FIR and IIR Adaptive Filters.* PhD thesis, Brigham Young University, 1988.

[30] J. Chao, H. Perez and S. Tsujii. A Fast Adaptive Filter Algorithm Using Eigenvalue Reciprocals as Stepsizes. *IEEE Transactions on Acoustics Speech and Signal Processing*, 38-8:1343–1352, August 1990.

[31] S. A. White. An adaptive recursive digital filter. *Proceedings of the 9th Asilomar IEEE Conference on Circuits, Systems and Computers*, 21–25, November 1972. Pacific Grove, CA.

[32] P. L. Feintuch. An adaptive recursive LMS filter. *Proceedings of the IEEE*, 64-11:1622–1624, November 1976.

[33] C. R. Johnson Jr. Charf convergence studies. *Proceedings of the 13th Asilomar IEEE Conference on Circuits, Systems and Computers*, 403–407, November 1979. Pacific Grove, CA.

[34] M. G. Larimore, J. R. Treichler, and C. R. Johnson Jr. Sharf: an algorithm for adapting IIR digital filters. *IEEE Transactions on Acoustics Speech and Signal Processing*, 28-4:428–440, August 1980.

[35] C. R. Johnson Jr. Adaptive IIR filtering: current results and open issues. *IEEE Transactions on Information Theory*, 30-2:237–250, March 1984.

[36] B. Gold and K. L. Jordan Jr. A note on digital filter synthesis. *Proceedings of the IEEE*, 56-10:1717–1718, October 1968.

[37] H. B. Voelcker Digital Filtering Via Block Recursion. *IEEE Transactions on Audio and Electroacoustics*, 18-2:169–176, June 1970.

[38] C. S. Burrus. Block implementation of digital filters. *IEEE Transactions on Circuit Theory*, 18-6:697–701, November 1971.

[39] C. S. Burrus. Block realization of digital filters. *IEEE Transactions on Audio and Electroacoustics*, 20-4:230–235, October 1972.

[40] R. Gnanasekaran and S. K. Mitra. A note on block implementation of IIR digital filters. *Proceedings of the IEEE*, 65-7:1063–1064, July 1977.

[41] S. K. Mitra and R. Gnanasekaran. Block implementation of recursive digital filters - new structures and properties. *IEEE Transacitons on Circuits and Systems*, 25-4:200–207, April 1978.