Adaptive Filters

V. John Mathews
Scott C. Douglas

Contents

Chapter 11

Fast RLS Transversal Filters

The most significant disadvantage of the conventional and the QR-decomposition-based RLS adaptive filters, as compared to stochastic gradient adaptive filters, is their higher computational complexity. A realization of an RLS adaptive filter with computational complexity that is comparable to the LMS algorithm will increase the usefulness of the RLS adaptive filters in practical applications, particularly in situations where the filter length L is larger than a few samples. The purpose of this chapter is to derive efficient or "fast" algorithms for implementing the exponentially-windowed RLS adaptive filter.

11.1 Basic Approach to the Derivation of the Fast RLS Algorithms

It is useful to review the development of the conventional RLS adaptive filter before we start the derivation of the computationally-efficient RLS adaptive filters. The derivation of the conventional RLS adaptive filter is valid for all types of input vectors, *i.e.*, it does not matter whether the input vector $\mathbf{X}(n)$ consists of signal samples from several different channels or if the elements of $\mathbf{X}(n)$ belong to L successive samples of one signal. This generality of the conventional RLS adaptive filter is the main reason for its extra complexity when compared with the LMS adaptive filter. In a single-channel situation in which the input vector $\mathbf{X}(n)$ is formed as

$$\mathbf{X}^T(n) = [x(n)\; x(n-1) \cdots x(n-L+1)]\,, \tag{11.1}$$

$L-1$ samples of $\mathbf{X}(n)$ and $\mathbf{X}(n-1)$ are identical. The fact that the two vectors differ by only one sample enables us to derive exact recursive realizations of the optimal least-squares coefficient vector $\mathbf{W}(n)$ using only $O(L)$ arithmetic operations per iteration. The resulting adaptive filters are several times more complex than the LMS adaptive filter, but their computational complexity is an order of magnitude less than that of the conventional RLS adaptive filter.

Consider the recursions given in Table 10.1 for the conventional RLS adaptive filter. The equation to update the coefficient vector $\mathbf{W}(n)$ can be implemented using L multiplications and L additions, provided the gain vector $\mathbf{k}(n)$ and the *a-priori* estimation error $\epsilon(n)$ is available. The calculation of $\epsilon(n)$ requires L multiplications and L additions. However, the calculation of $\mathbf{k}(n)$ requires $O(L^2)$ arithmetical operations. Therefore, we must simplify the computation of the gain vector in order to derive an algorithm with computational complexity that is comparable to that of the LMS adaptive filter.

11.1.1 Two Interpretations of the Gain Vector

Recall from (10.32) that the gain vector is given by

$$\mathbf{k}(n) = \hat{\mathbf{R}}_{\mathbf{xx}}^{-1}(n)\mathbf{X}(n). \tag{11.2}$$

We also know that $\mathbf{k}(n)$ is the optimal least-squares coefficient vector that estimates $\pi_n(k)$ using $\mathbf{X}(k)$ during $1 \leq k \leq n$. The problem of deriving the fast RLS transversal filter recursions may be stated as follows: Given $\mathbf{k}(n-1)$, all the related variables computed at time $n-1$, and the input vector $\mathbf{X}(n)$ at time n, compute $\mathbf{k}(n)$ in an efficient manner.

Now, $\mathbf{k}(n-1)$ can be interpreted in two different ways. Obviously, it is the coefficient vector that estimates $\pi_{n-1}(k)$ using the input vector sequence $\mathbf{X}(k)$ during $1 \leq k \leq n-1$. According to this interpretation,

$$\begin{aligned}\mathbf{k}(n-1) &= \left[\sum_{k=1}^{n-1} \lambda^{(n-1)-k}\mathbf{X}(k)\mathbf{X}^T(k)\right]^{-1}\left[\sum_{k=1}^{n-1} \lambda^{(n-1)-k}\mathbf{X}(k)\pi_{n-1}(k)\right] \\ &= \hat{\mathbf{R}}_{\mathbf{xx}}^{-1}(n-1)\mathbf{X}(n-1), \end{aligned}\tag{11.3}$$

since $\pi_{n-1}(k)$ is nonzero only for $k = n-1$. The second interpretation we can give to $\mathbf{k}(n-1)$ is that it is the coefficient vector that estimates $\pi_n(k)$ using the input vector sequence $\mathbf{X}(k-1)$ during $1 \leq k \leq n$. The optimal coefficient vector for this problem is given by

$$\begin{aligned}&\left[\sum_{k=1}^{n} \lambda^{n-k}\mathbf{X}(k-1)\mathbf{X}^T(k-1)\right]^{-1}\left[\sum_{k=1}^{n} \lambda^{n-k}\mathbf{X}(k-1)\pi_n(k)\right] \\ &\qquad = \left[\sum_{k=1}^{n} \lambda^{(n-1)-(k-1)}\mathbf{X}(k-1)\mathbf{X}^T(k-1)\right]^{-1}\mathbf{X}(n-1).\end{aligned}\tag{11.4}$$

Using a change of variables and recognizing that $\mathbf{X}(0)$ is a zero vector, we can see that the above simplifies to

$$\hat{\mathbf{R}}_{\mathbf{xx}}^{-1}(n-1)\mathbf{X}(n-1) = \mathbf{k}(n-1), \tag{11.5}$$

which is exactly the same as (11.3), thus verifying the second interpretation for $\mathbf{k}(n-1)$. We will use the second interpretation for our derivation of the fast RLS transversal filter.

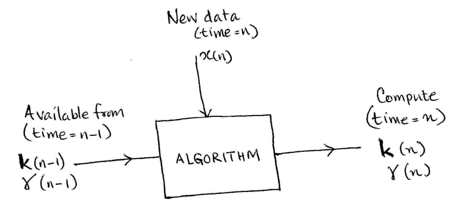


Figure 11.1: Problem statement for deriving an efficient update equation for $\mathbf{k}(n)$.

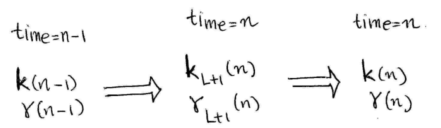


Figure 11.2: Methodology for updating $\mathbf{k}(n)$

11.2 Strategy for Updating the Gain Vector

Figure 11.1 illustrates the basic strategy for calculating the gain vector. The problem that is solved at time $n-1$ to yield $\mathbf{k}(n-1)$ and the problem that is solved at time n to yield $\mathbf{k}(n)$ both involve estimating the same signal $\pi_n(k)$. Furthermore, the first problem uses $\mathbf{X}(k-1)$ to form the estimates while the second problem uses $\mathbf{X}(k)$. As noted before, for the single channel problem, these two input vectors have $L-1$ elements in common. In fact, if we consider a slightly larger input vector with $L+1$ elements defined as

$$\mathbf{X}_{L+1}^T(k) = [x(k)\; x(k-1) \cdots x(k-L)]\,, \tag{11.6}$$

this *augmented input vector* contains both $\mathbf{X}(k)$ and $\mathbf{X}(k-1)$. We can explicitly state this fact by writing $\mathbf{X}_{L+1}(k)$ as

$$\mathbf{X}_{L+1}(k) = \begin{bmatrix} x(k) \\ \mathbf{X}(k-1) \end{bmatrix} = \begin{bmatrix} \mathbf{X}(k) \\ x(k-L) \end{bmatrix}. \tag{11.7}$$

Since the augmented input vector contains both $\mathbf{X}(k)$ and $\mathbf{X}(k-1)$, one reasonable approach for obtaining $\mathbf{k}(n)$ is to first estimate $\pi_n(k)$ using $\mathbf{X}_{L+1}(k)$ and then find a way to estimate $\pi_n(k)$ using $\mathbf{X}(k)$. The key is to relate the information we already have about how we can estimate $\pi_n(k)$ using $\mathbf{X}(n-1)$ to the estimation of $\pi_n(k)$ using the augmented input vector. This strategy is depicted in Figure 11.2. We use $\mathbf{k}_{L+1}(n)$ to denote the optimum exponentially-weighted least-squares coefficients for estimating $\pi_n(k)$ using $\mathbf{X}_{L+1}(n)$. We call this vector the *augmented gain vector*. The quantities $\gamma(n-1)$, $\gamma_{L+1}(n)$ and $\gamma(n)$ in Figure 11.2 represent the errors in estimating $\pi_n(n) = 1$ using $\mathbf{X}(n-1)$, $\mathbf{X}_{L+1}(n)$ and $\mathbf{X}(n)$, respectively.

11.2.1 Calculation of $\mathbf{k}_{L+1}(n)$ from $\mathbf{k}(n-1)$

Since $\pi_n(n) = 1$,

$$\gamma(n-1) = 1 - \mathbf{k}^T(n-1)\mathbf{X}(n-1) \tag{11.8}$$

and

$$\gamma_{L+1}(n) = 1 - \mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n). \tag{11.9}$$

The second term on the right hand side of the above equation is an estimate of $\pi_n(n)$. This estimate has contributions from $\mathbf{X}(n-1)$ and the new sample $x(n)$. We already know how to estimate $\pi_n(n)$ using $\mathbf{X}(n-1)$. This estimate is given by $\mathbf{k}^T(n-1)\mathbf{X}(n-1)$. Now, what additional information does the new sample $x(n)$ provide us in order to make a better estimate of $\pi_n(n)$? The answer to this question is straightforward, given the results of Chapter 3. It was shown there that, for a properly defined inner-product space, the error in estimating $x(n)$ using $\mathbf{X}(n-1)$ is orthogonal to $\mathbf{X}(n)$, and therefore provides the new information contained in the most recent data sample. This error is nothing but the Lth order forward prediction error.

For our derivation, the inner product is defined by the exponentially-weighted least-squares error criterion as in (10.20). Let $\mathbf{A}(n)$ denote the optimal forward predictor coefficient vector for estimating $x(k)$ using $\mathbf{X}(k-1)$ and let $f_n(k)$ be the estimation error of the forward predictor at time n, defined as

$$f_n(k) = x(k) - \mathbf{A}^T(n)\mathbf{X}(k-1). \tag{11.10}$$

We can now express the estimate of $\pi_n(n)$ using $\mathbf{X}_{L+1}(n)$ as

$$\mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n) = \mathbf{k}^T(n-1)\mathbf{X}(n-1) + r_f(n) \cdot f_n(n), \tag{11.11}$$

where $r_f(n)$ is the optimal coefficient that estimates $\pi_n(k)$ using $f_n(k)$ at time n. This coefficient is given by

$$r_f(n) = f_n(n)/\alpha(n), \tag{11.12}$$

where $\alpha(n)$ is the least-squares value of the forward prediction error given by

$$\alpha(n) = \sum_{k=1}^{n} \lambda^{n-k} f_n^2(k), \tag{11.13}$$

and the forward prediction error value $f_n(n)$ also has the interesting interpretation as the least-squares cross-correlation of $f_n(k)$ and $\pi_n(k)$, since

$$f_n(n) = \sum_{k=1}^{n} \lambda^{n-k} f_n(k)\pi_n(k). \tag{11.14}$$

An expression for $\mathbf{k}_{L+1}(n)$ can be obtained by substituting equation (11.10) in (11.11) and equating the coefficients of like terms on both sides of (11.10). This procedure gives

$$\begin{aligned}\mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n) &= \mathbf{k}^T(n-1)\mathbf{X}(n-1) + r_f(n)\Big(x(n) - \mathbf{A}^T(n)\mathbf{X}(n-1)\Big) \\ &= \left(\begin{bmatrix} 0 \\ \mathbf{k}(n-1) \end{bmatrix} + r_f(n)\begin{bmatrix} 1 \\ -\mathbf{A}(n) \end{bmatrix}\right)^T \begin{bmatrix} x(n) \\ \mathbf{X}(n-1) \end{bmatrix} \\ &= \begin{bmatrix} r_f(n) \\ \mathbf{k}(n-1) - r_f(n)\mathbf{A}(n) \end{bmatrix}^T \mathbf{X}_{L+1}(n). \end{aligned} \tag{11.15}$$

Equating coefficients of $\mathbf{X}_{L+1}(n)$ on the left-hand side of the first line of (11.15) and the right-hand side of its last line, we get

$$\mathbf{k}_{L+1}(n) = \begin{bmatrix} r_f(n) \\ \mathbf{k}(n-1) - r_f(n)\mathbf{A}(n) \end{bmatrix}. \tag{11.16}$$

Equation (11.16) is significant. This result tells us that, if both the forward predictor coefficient vector $\mathbf{A}(n)$ and the coefficient $r_f(n)$ are available, we can evaluate the augmented gain vector using L multiplications and L additions.

11.2.2 Calculation of $\mathbf{k}(n)$ from $\mathbf{k}_{L+1}(n)$

Since the augmented input vector can also be decomposed as

$$\mathbf{X}_{L+1}(n) = \begin{bmatrix} \mathbf{X}(n) \\ x(n-L) \end{bmatrix}, \tag{11.17}$$

it is reasonable to assume that the augmented gain vector $\mathbf{k}_{L+1}(n)$ and the current gain vector $\mathbf{k}(n)$ satisfy a relationship similar to that in (11.16). In fact, such a relationship exists. Once this relationship is derived, we can use it along with (11.16) to express $\mathbf{k}(n)$ in terms of $\mathbf{k}(n-1)$.

To arrive at the second relationship, we proceed in the same fashion as before. The estimate of $\pi_n(n)$ using the augmented input vector can be decomposed as the sum of the estimate of $\pi_n(n)$ using $\mathbf{X}(n)$ and the estimate of $\pi_n(n)$ using the part of $x(n-L)$ that is orthogonal to $\mathbf{X}(n)$. The component of $x(n-L)$ orthogonal to $\mathbf{X}(n)$ is given by the error in estimating $x(n-L)$ using $\mathbf{X}(n)$. This component is simply the Lth order backward prediction error at time n. Let $\mathbf{G}(n)$ denote the optimal RLS backward predictor coefficient vector at time n. The corresponding prediction error sequence is given by

$$b_n(k) = x(k-L) - \mathbf{G}^T(n)\mathbf{X}(k). \tag{11.18}$$

We can now express the estimate of $\pi_n(n)$ using $\mathbf{X}_{L+1}(n)$ as

$$\mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n) = \mathbf{k}^T(n)\mathbf{X}(n) + r_b(n)\cdot b_n(n), \tag{11.19}$$

where $r_b(n)$ is the coefficient given by

$$r_b(n) = \frac{b_n(n)}{\beta(n)} \tag{11.20}$$

and

$$\beta(n) = \sum_{k=1}^{n} \lambda^{n-k} b_n^2(k) \tag{11.21}$$

is the exponentially-weighted least-squares estimate of the autocorrelation of the backward prediction error sequence at time n, and $b_n(n)$ can be interpreted as the exponentially weighted least-squares cross-correlation of $b_n(k)$ and $\pi_n(k)$. Substituting (11.18) in (11.19) and collecting like terms, we get

$$\begin{aligned} \mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n) &= \mathbf{k}^T(n)\mathbf{X}(n) + r_b(n)\left(x(n-L) - \mathbf{G}^T(n)\mathbf{X}(n)\right) \\ &= \left(\begin{bmatrix} \mathbf{k}(n) \\ 0 \end{bmatrix} + r_b(n)\begin{bmatrix} -\mathbf{G}(n) \\ 1 \end{bmatrix}\right)^T \mathbf{X}_{L+1}(n). \end{aligned} \tag{11.22}$$

Equating the coefficients of $\mathbf{X}_{L+1}(n)$ on both sides of the above equation results in the desired relationship:

$$\mathbf{k}_{L+1}(n) = \begin{bmatrix} \mathbf{k}(n) - r_b(n)\mathbf{G}(n) \\ r_b(n) \end{bmatrix}. \tag{11.23}$$

Equating the right-hand sides of (11.16) and (11.23), we arrive at a result that relates $\mathbf{k}(n)$ to $\mathbf{k}(n-1)$ through the coefficients of the forward and backward predictors and the two coefficients $r_f(n)$ and $r_b(n)$. Assuming that we know how to compute the associated variables, updating $\mathbf{k}(n)$ can be performed in two steps as follows:

1. Obtain $\mathbf{k}_{L+1}(n)$ using equation (11.16).

2. Solve for $\mathbf{k}(n)$ from equation (11.23).

We now proceed to find efficient ways of calculating $\mathbf{A}(n), \mathbf{G}(n), r_f(n)$ and $r_b(n)$ using recursive estimators.

11.3 Adaptation of the Predictor Parameters

As discussed in the previous section, efficient calculation of the gain vector requires the knowledge of the forward and backward coefficient vectors at each time. In addition, we need to calculate the coefficients $r_f(n)$ and $r_b(n)$ at every iteration. We derive the update equations for these parameters in the following subsections.

11.3.1 Updating the Forward Predictor Coefficients

Exponentially-weighted least-squares forward prediction uses $x(n-1), x(n-2), \cdots, x(n-L)$ to estimate $x(n)$ such that

$$J_f(n) = \sum_{k=1}^{n} \lambda^{n-k} f_n^2(k) \tag{11.24}$$

is minimized at each time instant. Here, $f_n(k)$ is the estimation error at time k obtained using the optimal forward predictor coefficients at time n, and is given by

$$f_n(k) = x(k) - \mathbf{A}_f^T(n)\mathbf{X}_f(k), \tag{11.25}$$

where $\mathbf{X}_f(k)$ is the input signal vector for the forward prediction task as given by

$$\mathbf{X}_f(k) = [x(k-1) \;\; x(k-2) \;\; \cdots \;\; x(k-L)]^T . \tag{11.26}$$

Note that

$$\mathbf{X}_f(k) = \mathbf{X}(k-1). \tag{11.27}$$

As in the case of previous derivations, we use special symbols for the *a priori* and *a posteriori* forward prediction errors. Let us define $\phi(n)$ and $f(n)$ as

$$\phi(n) = f_{n-1}(n) \tag{11.28}$$

and

$$f(n) = f_n(n), \tag{11.29}$$

respectively. From our previous discussion of the conventional RLS adaptive filters, the update equation for the predictor coefficients should have the form

$$\mathbf{A}(n) = \mathbf{A}(n-1) + \mathbf{k}_f(n)\phi(n) \tag{11.30}$$

where $\mathbf{k}_f(n)$ is the gain vector for the forward prediction problem given by

$$\mathbf{k}_f(n) = \hat{\mathbf{R}}_{ff}^{-1}(n)\mathbf{X}_f(n). \tag{11.31}$$

Now,

$$\begin{aligned}\hat{\mathbf{R}}_{ff}(n) &= \sum_{k=1}^{n} \lambda^{n-k}\mathbf{X}_f(k)\mathbf{X}_f^T(k) \\ &= \sum_{k=1}^{n} \lambda^{n-k}\mathbf{X}(k-1)\mathbf{X}^T(k-1) \\ &= \sum_{k=1}^{n-1} \lambda^{(n-1)-k}\mathbf{X}(k)\mathbf{X}^T(k) \\ &= \hat{\mathbf{R}}_{\mathbf{xx}}(n-1). \qquad (11.32)\end{aligned}$$

In deriving the above result, we made use of the fact that $\mathbf{X}(0)$ is a zero vector. Substituting (11.32) in (11.31), we find that

$$\mathbf{k}_f(n) = \hat{\mathbf{R}}_{\mathbf{xx}}(n-1)\mathbf{X}(n-1) = \mathbf{k}(n-1) \qquad (11.33)$$

Thus, the update equation for the forward predictor coefficients uses a delayed version of the gain vector $\mathbf{k}(n)$ used for joint process estimation. The coefficient update equation is

$$\mathbf{A}(n) = \mathbf{A}(n-1) + \mathbf{k}(n-1)\phi(n), \qquad (11.34)$$

where

$$\phi(n) = x(n) - \mathbf{A}^T(n-1)\mathbf{X}(n-1). \qquad (11.35)$$

It is important to realize that $\mathbf{A}(n)$ can be computed at time n using information that is available at time $n-1$ and the input data sample $x(n)$.

11.3.2 Updating the Backward Predictor Coefficients

We can derive a similar update equation for the backward predictor coefficients. The L-th order backward prediction problem uses $x(n), x(n-1), \cdots, x(n-L+1)$ to estimate $x(n-L)$. The exponentially weighted least-squares Lth order backward predictor minimizes the cost function

$$J_b(n) = \sum_{k=1}^{n} \lambda^{n-k} b_n^2(k), \qquad (11.36)$$

where

$$b_n(k) = x(k-L) - \mathbf{G}^T(n)\mathbf{X}(n) \qquad (11.37)$$

is the estimation error at time k using the optimal backward predictor coefficient vector $\mathbf{G}(n)$. As before, let us define the *a priori* and *a posterori* backward prediction error signals as

$$\psi(n) = b_{n-1}(n) \qquad (11.38)$$

and

$$b(n) = b_n(n), \tag{11.39}$$

respectively. It is left as an exercise to show that the backward predictor coefficients can be updated as

$$\mathbf{G}(n) = \mathbf{G}(n-1) + \mathbf{k}(n)\psi(n). \tag{11.40}$$

11.3.3 Calculation of Least-Squares Prediction Errors

Recall from (2.50) that the minimum value of the cost function for the exponentially-weighted linear least-squares estimation of the signal $d(n)$ using $\mathbf{X}(n)$ is given by

$$J_{min}(n) = \hat{r}_{dd}(n) - \hat{\mathbf{P}}^T_{\mathbf{X}\mathrm{d}}(n)\mathbf{W}(n) \tag{11.41}$$

where

$$\hat{r}_{dd}(n) = \sum_{k=1}^{n} \lambda^{n-k} d^2(k) \tag{11.42}$$

is the least-squares estimate of the autocorrelation of the signal $d(n)$ and $\hat{\mathbf{P}}_{\mathbf{X}\mathrm{d}}(n)$ is the least-squares cross-correlation vector of $\mathbf{X}(n)$ and $d(n)$. We can get a recursive equation for computing $J_{min}(n)$ by substituting the time update equations for $\hat{r}_{dd}(n), \hat{\mathbf{P}}_{\mathbf{X}\mathrm{d}}(n)$ and $\mathbf{W}(n)$ from (10.26), (10.27) and (10.36) in (11.41). This operation yields

$$\begin{aligned} J_{min}(n) &= \left(\lambda \hat{r}_{dd}(n-1) + d^2(n)\right) \\ &- \left(\lambda \hat{\mathbf{P}}_{\mathbf{X}\mathrm{d}}(n-1) + \mathbf{X}(n)d(n)\right)^T \left(\mathbf{W}(n-1) + \mathbf{k}(n)\epsilon(n)\right). \end{aligned} \tag{11.43}$$

Expanding the right-hand side of the above equation and rearranging the terms, we get

$$\begin{aligned} J_{min}(n) &= \lambda \left(\hat{r}_{dd}(n-1) - \mathbf{W}^T(n-1)\hat{\mathbf{P}}_{\mathbf{X}\mathrm{d}}(n-1)\right) \\ &+ d(n)\left(d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n)\right) - \hat{\mathbf{P}}^T_{\mathbf{X}\mathrm{d}}(n)\mathbf{k}(n)\epsilon(n). \end{aligned} \tag{11.44}$$

The terms inside the large parentheses on the first line of (11.44) is $J_{min}(n-1)$. Similarly, the terms within the large parentheses on the second line of (11.44) is the *a priori* estimation error $\epsilon(n)$. Now, since

$$\mathbf{k}(n) = \hat{\mathbf{R}}^{-1}_{\mathbf{XX}}(n)\mathbf{X}(n), \tag{11.45}$$

$$\begin{aligned} \hat{\mathbf{P}}^T_{\mathbf{X}\mathrm{d}}(n)\mathbf{k}(n) &= \hat{\mathbf{P}}^T_{\mathbf{X}\mathrm{d}}(n)\hat{\mathbf{R}}^{-1}_{\mathbf{XX}}(n)\mathbf{X}(n) \\ &= \mathbf{W}^T(n)\mathbf{X}(n). \end{aligned} \tag{11.46}$$

Substituting the above result in (11.44) gives

$$\begin{aligned} J_{min}(n) &= \lambda J_{min}(n-1) + \left(d(n) - \mathbf{W}^T(n)\mathbf{X}(n)\right)\epsilon(n) \\ &= \lambda J_{min}(n-1) + e(n)\epsilon(n). \end{aligned} \tag{11.47}$$

The above result is applicable to all exponentially-weighted recursive least-squares estimation problems, and is very useful. It states that the least-squares value of the estimation error can be determined by adding the product of the *a priori* and *a posteriori* estimation errors to λ times the previous value of the least-squares estimation error. We can apply this result directly to both the forward and the backward prediction problems. The corresponding expressions are

$$\alpha(n) = \lambda\alpha(n-1) + \phi(n)f(n) \tag{11.48}$$

and

$$\beta(n) = \lambda\beta(n-1) + \psi(n)b(n) \tag{11.49}$$

for the forward and backward prediction problems, respectively.

11.4 Putting the Pieces Together

Let us review what we have accomplished up to this point. Our objective is to compute the gain vector $\mathbf{k}(n)$ at time n from knowledge of $\mathbf{k}(n-1)$ and the new input sample $x(n)$. Our approach calculates the augmented gain vector $\mathbf{k}_{L+1}(n)$ as an intermediate step. Equation (11.16) describes the necessary relationship. To implement this step, we must calculate the forward prediction coefficient vector $\mathbf{A}(n)$ and the least-squares value of the forward prediction error. From (11.34), we see that we can calculate $\mathbf{A}(n)$ using $\mathbf{A}(n-1), \mathbf{k}(n-1)$ and the *a priori* forward prediction error $\phi(n)$. All of these quantities can be computed with the information we have at the given point in time, and therefore, the augmented gain vector can be calculated.

The next step is to evaluate $\mathbf{k}(n)$ from $\mathbf{k}_{L+1}(n)$. Equation (11.23) provides the relationship between the augmented gain vector and $\mathbf{k}(n)$. However, this relationship depends on the backward prediction coefficient vector $\mathbf{G}(n)$. We can compute $\mathbf{G}(n)$ using the relationship in (11.40), but this calculation requires knowledge of $\mathbf{k}(n)$. Thus, we have a situation in which there are two sets of unknowns that depend on each other through the relationships in (11.23) and (11.40). We must solve for the unknown parameters simultaneously. Let $\mathbf{k}_{L+1}^{(L)}(n)$ denote a vector formed by the first L elements of the augmented gain vector. Then, the two relationships in $\mathbf{k}(n)$ and $\mathbf{G}(n)$ are as follows:

$$\mathbf{k}_{L+1}^{(L)}(n) = \mathbf{k}(n) - r_b(n)\mathbf{G}(n) \tag{11.50}$$

and

$$\mathbf{G}(n-1) = \mathbf{G}(n) - \mathbf{k}(n)\psi(n). \tag{11.51}$$

We can compute $\mathbf{k}_{L+1}^{(L)}(n), \mathbf{G}(n-1)$ and the *a priori* backward prediction error $\psi(n)$ in the above equations using previously calculated quantities. From (11.23), we note that $r_b(n)$ is the last element of the augmented gain vector. Consequently, we can evaluate $r_b(n)$ directly as

$$r_b(n) = k_{L+1,L}(n), \tag{11.52}$$

Table 11.1: The first version of the fast RLS adaptive filter.

Equation No.	*Equations*
1	$\phi(n) = x(n) - \mathbf{A}^T(n-1)\mathbf{X}(n-1)$
2	$\mathbf{A}(n) = \mathbf{A}(n-1) + \mathbf{k}(n-1)\phi(n)$
3	$f(n) = x(n) - \mathbf{A}^T(n)\mathbf{X}(n-1)$
4	$\alpha(n) = \lambda\alpha(n-1) + f(n)\phi(n)$
5	$r_f(n) = f(n)/\alpha(n)$
6	$\mathbf{k}_{L+1}(n) = \begin{bmatrix} r_f(n) \\ \mathbf{k}(n-1) - r_f(n)\mathbf{A}(n) \end{bmatrix}$
7	$\begin{bmatrix} \mathbf{k}_{L+1}^{(L)}(n) \\ r_b(n) \end{bmatrix} = \mathbf{k}_{L+1}(n)$
8	$\psi(n) = x(n-L) - \mathbf{G}^T(n-1)\mathbf{X}(n)$
9	$\mathbf{k}(n) = (1-\psi(n)r_b(n))^{-1}\left(\mathbf{k}_{L+1}^{(L)}(n) + r_b(n)\mathbf{G}(n-1)\right)$
10	$\mathbf{G}(n) = \mathbf{G}(n-1) + \mathbf{k}(n)\psi(n)$
11	$\epsilon(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n)$
12	$\mathbf{W}(n) = \mathbf{W}(n-1) + \mathbf{k}(n)\epsilon(n)$
13	$e(n) = d(n) - \mathbf{W}^T(n)\mathbf{X}(n)$

where $k_{L+1,L}(n)$ is the last element of $\mathbf{k}_{L+1}(n)$. We can eliminate $\mathbf{G}(n)$ from (11.50) and (11.51) by multiplying (11.51) with $r_b(n)$ and adding the result to (11.50). This operation gives

$$\mathbf{k}_{L+1}^{(L)} + r_b(n)\mathbf{G}(n-1) = (1-\psi(n)r_b(n))\,\mathbf{k}(n). \tag{11.53}$$

Finally, we can solve for $\mathbf{k}$(n) to get

$$\mathbf{k}(n) = (1-\psi(n)r_b(n))^{-1}\left(\mathbf{k}_{L+1}^{(L)}(n) + r_b(n)\mathbf{G}(n-1)\right) \tag{11.54}$$

This update equation, followed by the update equation for the backward prediction coefficients given by (11.51), completes the algorithm. The complete recursion for efficiently implementing the exponentially-weighted RLS adaptive filter is given in Table 11.1. Table 11.2 lists the arithmetic operations associated with each step. The computational complexity of this realization of the adaptive filter is approximately proportional to the number of coefficients for large values of L. Consequently, the computational complexity of this filter is an order of magnitude smaller than that of the conventional RLS adaptive filter. Furthermore, the computational complexity of the fast RLS adaptive filters is now comparable to that of the LMS adaptive filter, even though the faster realization still requires approximately five times as many computations as the LMS adaptive filter.

Table 11.2: Operations count for the fast RLS adaptive filter of Table 11.1.

Equation	*Number of Multiplications*	*Number of Divisions*	*Number of Additions and Substractions*
1	L	0	L
2	L	0	L
3	L	0	L
4	2	0	1
5	0	1	0
6	L	0	L
7	0	0	0
8	L	0	L
9	$2\,L+1$	1	$L+1$
10	L	0	L
11	L	0	L
12	L	0	L
13	L	0	L
Total	$11\,L+3$	2	$10\,L+2$

11.4.1 Initialization of the Fast RLS Adaptive Filter

It is possible to derive a method for initializing the recursions of Table 11.1 exactly when the coefficients are initialized to zero values [Cioffi 1984]. However, we will find in Section 11.7 that the fast RLS adaptive filters are prone to numerical instability. Exact initialization techniques tend to worsen this problem. Furthermore, this technique does not allow initialization of the coefficients to arbitrary values. In the following, we discuss the most common method of initializing fast RLS adaptive filters. The technique is known as *soft-constrained initialization.*

We saw in Exercise 10.5 that minimizing a cost function of the form

$$J(n) \;= \lambda^n \delta \parallel \mathbf{W}(n) - \mathbf{W}(0) \parallel^2 + \sum_{k=1}^{n} \lambda^{n-k} \Big(d(k) - \mathbf{W}^T(n)\mathbf{X}(k)\Big)^2, \tag{11.55}$$

where δ is a positive constant, results in a recursion that can be initialized using

$$\mathbf{R_{xx}}(0) = \delta \mathbf{I}. \tag{11.56}$$

In the soft-constrained initialization procedure, the cost function of (11.55) is slightly modified so that the adaptive filter minimizes

$$J(n) = \sum_{i=0}^{L-1} \lambda^{n+L-i} \delta (w_i(n) - w_i(0))^2 + \sum_{k=1}^{n} \lambda^{n-k} \Big(d(k) - \mathbf{W}^T(n)\mathbf{X}(k)\Big)^2 \tag{11.57}$$

at each iteration. The difference between the cost functions in (11.55) and (11.57) is that the the squared differences between the initial and current values of the coefficients are weighted uniformly in (11.55), while these squared differences are weighted according to their indices in (11.57). The cost function of (11.57) might appear unnatural at first. However, since the input signal $x(n) = 0$ for $n \leq 0$, the coefficient $w_i(n)$ does not change its value from $w_i(0)$ for $1 \leq i + 1$. The cost function in (11.57) weights the coefficient differences uniformly in accordance with how many times the particular coefficient has been changed, or equivalently, in accordance with how many times the the input signal has been used to update the ith coefficient.

One advantage of using the cost function of (11.57) is that it is possible to augment the input and desired response signals so that this cost function becomes the least-squares cost function for the modified signals. The modified signals are defined as

$$x_a(n) = \begin{cases} \sqrt{\delta} & ; \quad n = -L \\ 0 & ; \quad -L+1 \leq n \leq 0 \\ x(n) & ; \quad \text{otherwise} \end{cases} \tag{11.58}$$

and

$$d_a(n) = \begin{cases} \sqrt{\delta} w_{n+L}(0) & ; \quad -L \leq n \leq -1 \\ 0 & ; \quad n = 0 \\ d(n) & ; \quad \text{otherwise} \end{cases} \tag{11.59}$$

It is left as an exercise for the reader to show that the cost function given by

$$J_a(n) = \sum_{k=-L}^{n} \lambda^{n-k} \left(d_a(k) - \mathbf{W}^T(n)\mathbf{X}_a(n) \right)^2, \tag{11.60}$$

where $\mathbf{X}_a(n) = [x_a(n) \;\; x_a(n-1) \;\; \cdots \;\; x_a(n-L+1)]^T$, is identical to the cost function of (11.57). To initialize the adaptive filter at time $n = 0$, we note that all the elements of the least-squares forward predictor coefficient vector that minimizes

$$J_{f,a}(n) = \sum_{k=-L}^{0} \lambda^{n-k} \left(x_a(k) - \mathbf{A}^T(n)\mathbf{X}_a(n-1) \right)^2 \tag{11.61}$$

are zero. Therefore, we can initialize $\mathbf{A}(n)$ using

$$\mathbf{A}(n) = \mathbf{0}. \tag{11.62}$$

The least-squares forward prediction error value can be calculated to be

$$\alpha(0) = \lambda^L \delta. \tag{11.63}$$

Similarly, initialization of the backward predictor parameters can be performed by considering the cost function

$$J_{b,a}(n) = \sum_{k=-L}^{n} \lambda^{n-k} \left(x_a(k-L) - \mathbf{G}^T(n)\mathbf{X}_a(n) \right)^2. \tag{11.64}$$

An analysis similar to that for the forward predictor parameters results in[1]

$$\mathbf{G}(0) = \mathbf{0} \tag{11.65}$$

and

$$\beta(0) = \delta. \tag{11.66}$$

The gain vector can be initialized as

$$\mathbf{k}(0) = \mathbf{0}. \tag{11.67}$$

In the derivation of more efficient algorithms in later sections, we require the propagation of $\gamma(n) = 1 - \mathbf{k}^T(n)\mathbf{X}(n)$. The variable $\gamma(n)$ in such cases can be initialized as

$$\gamma(0) = 1. \tag{11.68}$$

Table 11.3 contains a MATLAB function that implements the fast RLS adaptive filter of Table 11.1. This MATLAB function is initialized using the soft constraint as discussed in this section.

11.5 A More Efficient RLS Adaptive Filter

The computationally-efficient realization of the recursive least squares adaptive filter given in Table 11.1 can be simplified further by making use of the relationships that exist among the different variables in the algorithm. We derive several such relationships that lead to an algorithm with much lower computational complexity in this section.

11.5.1 Relationship Between *a priori* and *a posteriori* Errors

There are three different types of estimation error signals – forward prediction, backward prediction and joint process estimation errors – that are updated during each iteration of the algorithm in Table 11.1. A simple relationship exists between the *a priori* and *a posteriori* estimation errors in all three cases. To see this, we substitute the equation for updating $\mathbf{W}(n)$ from (10.36) in the expression for $e(n)$ to get

$$\begin{aligned} e(n) &= d(n) - \mathbf{W}^T(n)\mathbf{X}(n) \\ &= d(n) - \left(\mathbf{W}(n-1) + \mathbf{k}(n)\epsilon(n)\right)^T \mathbf{X}(n). \end{aligned} \tag{11.69}$$

The estimation error given by $d(n) \; - \mathbf{W}(n-1)^T\mathbf{X}(n)$ is the *a priori* estimation error $\epsilon(n)$. Also, $\mathbf{k}^T(n)\mathbf{X}(n)$ is the least-squares estimate of the sequence $\pi_n(k)$ at time n. The corresponding estimation error is given by $\gamma(n)$, *i.e.,*

$$\gamma(n) = 1 - \mathbf{k}^T(n)\mathbf{X}(n). \tag{11.70}$$

[1]The algorithm of Table 11.1 does not require calculation of the least-squares value of the backward prediction error. However, its initialization is included here since the algorithms discussed later require propagation of this parameter.

Table 11.3: A MATLAB function that implements the fast RLS adaptive filter of Table 11.1.

```
        function[W,dhat,e] = ftf1(lambda,delta,W0,x,d);
%
%       This function adapts the coefficients of an FIR filter
%       using the first version of the fast RLS transversal filter
%       given in Table 11.1.
%
%      Input parameters:
%
%      lambda   = forgetting factor
%      delta    = soft constrained initialization constant
%      W0       = initial values of coefficients (L x 1)
%      x        = input signal (num_iter x 1)
%      d        = desired response signal (num_iter x 1)
%
%      Output of program:
%
%      W        = Evolution of coefficients (L x (num_iter + 1))
%      dhat     = output of adaptive filter (num_iter x 1)
%      e        = error of adaptive filter (num_iter x 1)
%
%
%       Initialization
%
        L = length(W0);
        start_iter = 1;
        end_iter = min([length(x),length(d)]);
        W1 = W0;
        X = zeros(size(W1));
        A = zeros(size(W1));
        G = zeros(size(W1));
        k = zeros(size(W1));
        alpha = delta * lambda^L;
%
%       Recursions
%
   for n = start_iter:end_iter,
        xL1 = X(L);
        eta = x(n) - A'*X;
        A = A + k*eta;
        f = x(n) - A'*X;
        X = [x(n);X(1:L-1)];
        alpha = lambda*alpha + eta*f;
        rf = f/alpha;
        kL1 = [rf;k-rf*A];
        rb = kL1(L+1);
        psi = xL1 - G'*X;
        den = 1/(1-psi*rb);
        k = den*(kL1(1:L) + rb*G);
        G = G + k*psi;
        epsilon = d(n) - W1'*X;
        W1 = W1 + k*epsilon;
        dhat(n) = W1'*X;
        e(n) = d(n) - dhat(n);
        W(:,n) = W1;
    end;
```

We can use the above relationship to simplify the expression for $e(n)$ as

$$\begin{aligned} e(n) &= \epsilon(n)\left(1-\mathbf{k}^T(n)\mathbf{X}(n)\right) \\ &= \epsilon(n)\gamma(n). \end{aligned} \tag{11.71}$$

In a similar manner, we can show that

$$f(n) = \phi(n)\gamma(n-1) \tag{11.72}$$

and that

$$b(n) = \psi(n)\gamma(n). \tag{11.73}$$

Thus, if we have an efficient means of computing $\gamma(n)$, each of the *a posteriori* errors can be evaluated using only one multiplication per update.

11.5.2 Calculation of $\gamma(n)$

Our approach for calculating $\gamma(n)$ is similar to the method we used for updating the gain vector. To update the gain vector, we first evaluated the augmented gain vector $\mathbf{k}_{L+1}(n)$ and then solved for the gain vector $\mathbf{k}(n)$ at time n. In the same manner, we derive a relationship between $\gamma(n-1)$ and $\gamma_{L+1}(n)$ and then solve for $\gamma(n)$. Recall from (11.9) that

$$\gamma_{L+1}(n) = 1 - \mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n). \tag{11.74}$$

We see from (11.11) and (11.12) that

$$\mathbf{k}_{L+1}^T(n)\mathbf{X}_{L+1}(n) = \mathbf{k}^T(n-1)\mathbf{X}(n-1) + \frac{f^2(n)}{\alpha(n)}. \tag{11.75}$$

It follows immediately that

$$\gamma_{L+1}(n) = \gamma(n-1) - \frac{f^2(n)}{\alpha(n)}. \tag{11.76}$$

By substituting (11.48) for $\alpha(n)$ and $f(n) = \phi(n)\gamma(n-1)$ in the above equation, we get

$$\begin{aligned}\gamma_{L+1}(n) &= \gamma(n-1) - \frac{\gamma(n-1)\phi(n)f(n)}{\lambda\alpha(n-1)+\phi(n)f(n)} \\ &= \frac{\lambda\alpha(n-1)\gamma(n-1)}{\alpha(n)}.\end{aligned} \tag{11.77}$$

In a similar manner, we can show that

$$\begin{aligned}\gamma_{L+1}(n) &= \gamma(n) - \frac{b^2(n)}{\beta(n)} \\ &= \frac{\lambda\beta(n-1)\gamma(n)}{\beta(n)}.\end{aligned} \tag{11.78}$$

The above relationships allow us to derive the first few steps of a more efficient RLS adaptive filter. Knowing $\mathbf{A}(n-1)$, we evaluate the *a priori* forward prediction error $\phi(n)$ directly. We then use (11.72) to find the *a posteriori* forward prediction error $f(n)$. We then update the forward prediction error power using (11.48). Since we now have $\gamma(n-1)$, $f(n)$ and $\alpha(n)$ available, we can compute $\gamma_{L+1}(n)$ using (11.76).

Now, in order to evaluate $b(n)$ using (11.73), we need $\gamma(n)$. However, to compute $\gamma(n)$, we need $\beta(n)$, which has not been calculated yet. Furthermore, we need $b(n)$ before we can evaluate $\beta(n)$. Consequently, we must develop an alternate approach to computing one or more of these three variables.

The augmented gain vector $\mathbf{k}_{L+1}(n)$ can be evaluated from knowledge of the variables already computed. In particular,

$$k_{L+1,L}(n) = \frac{b(n)}{\beta(n)}. \tag{11.79}$$

Substituting (11.79) and (11.73) in (11.78) gives

$$\gamma_{L+1}(n) = \gamma(n)\left(1 - k_{L+1,L}(n)\psi(n)\right). \tag{11.80}$$

Solving for $\gamma(n)$ from the above expression gives the desired result as

$$\gamma(n) = \frac{\gamma_{L+1}(n)}{1 - k_{L+1,L}(n)\psi(n)}. \tag{11.81}$$

11.5.3 Calculation of $\psi(n)$

It turns out that the *a priori* backward prediction error $\psi(n)$ can be computed without using the coefficient vector $\mathbf{G}(n-1)$. Since

$$k_{L+1,L}(n) = \frac{b(n)}{\beta(n)} = \frac{\gamma(n)\psi(n)}{\beta(n)}, \tag{11.82}$$

we can solve for $\psi(n)$ from the above equation as

$$\psi(n) = \frac{k_{L+1,L}(n)\beta(n)}{\gamma(n)}. \tag{11.83}$$

The backward prediction error power $\beta(n)$ in the above equation has not yet been computed. However, we see from (11.78) that

$$\frac{\beta(n)}{\gamma(n)} = \frac{\lambda\beta(n-1)}{\gamma_{L+1}(n)}. \tag{11.84}$$

We obtain an expression for $\psi(n)$ that depends only on quantities that have been previously evaluated by substituting (11.84) in (11.83). This operation gives

$$\psi(n) = \frac{\lambda k_{L+1,L}(n)\beta(n-1)}{\gamma_{L+1}(n)}. \tag{11.85}$$

11.5.4 A Faster RLS Adaptive Filter

We are now in a position to realize a more efficient version of the exponentially-weighted recursive least-squares adaptive filter than the one given in Table 11.1. The order in which we perform the various steps in the coefficient updates is as follows: We first compute the *a priori* forward prediction error $\phi(n)$ directly using the predictor coefficients at time $n-1$. With the help of $\phi(n)$, we update $\mathbf{A}(n)$. The *a posteriori* forward prediction error is estimated efficiently using (11.72). Now that we have computed the *a priori* and *a posteriori* forward prediction errors, we update the forward prediction error power using (11.48). The variable $\gamma_{L+1}(n)$ and the augmented gain vector $\mathbf{k}_{L+1}(n)$ are then calculated using (11.77) and (11.16), respectively. We then compute the *a priori* backward prediction error and $\gamma(n)$ using (11.85) and (11.81), respectively. The gain vector and the backward predictor coefficients are then updated using (11.54) and (11.51). At this point, the only quantities that need to be updated to complete the algorithm are $b(n)$, $\beta(n)$, $\epsilon(n)$, $\mathbf{W}(n)$ and $e(n)$. The *a priori* estimation error $\epsilon(n)$ is calculated directly using $\mathbf{W}(n-1)$. The rest of the variables are computed using (11.73), (11.49), (10.36) and (11.71), respectively. The complete algorithm is given in Table 11.4. The table explicitly shows that the first element of the augmented gain vector is $r_f(n)$. The number of computations associated with each step is given in Table 11.5. As can be seen, the simplifications have reduced the number of multiplications from $O(11L)$ to $O(8L)$.

Table 11.4: A more efficient version of the fast RLS adaptive filter.

Equation No.	*Equations*
1	$\phi(n) = x(n) - \mathbf{A}^T(n-1)\,\mathbf{X}(n-1)$
2	$\mathbf{A}(n) = \mathbf{A}(n-1) + \mathbf{k}(n-1)\,\phi(n)$
3	$f(n) = \phi(n)\,\gamma(n-1)$
4	$\alpha(n) = \lambda\alpha(n-1) + f(n)\,\phi(n)$
5	$\gamma_{L+1}(n) = \dfrac{\lambda\alpha(n-1)\,\gamma(n-1)}{\alpha(n)}$
6	$r_f(n) = f(n)/\alpha(n)$
7	$\mathbf{k}_{L+1}(n) = \begin{bmatrix} \mathbf{k}_{L+1}^{(L)}(n) \\ k_{L+1,L}(n) \end{bmatrix} = \begin{bmatrix} r_f(n) \\ \mathbf{k}(n-1) - \mathbf{A}(n)r_f(n) \end{bmatrix}$
8	$\psi(n) = \dfrac{\lambda k_{L+1,L}(n)\beta(n-1)}{\gamma_{L+1}(n)}$
9	$\gamma(n) = \dfrac{\gamma_{L+1}(n)}{1-\psi(n)k_{L+1,L}(n)}$
10	$\mathbf{k}(n) = \dfrac{1}{1-\psi(n)k_{L+1,L}(n)}\left(\mathbf{k}_{L+1}^{(L)}(n) + \mathbf{G}(n-1)k_{L+1,L}(n)\right)$
11	$\mathbf{G}(n) = \mathbf{G}(n-1) + \mathbf{k}(n)\,\psi(n)$
12	$b(n) = \psi(n)\,\gamma(n)$
13	$\beta(n) = \lambda\beta(n-1) + \psi(n)b(n)$
14	$\epsilon(n) = d(n) - \mathbf{W}^T(n-1)\,\mathbf{X}(n)$
15	$\mathbf{W}(n) = \mathbf{W}(n-1) + \mathbf{k}(n)\epsilon(n)$
16	$e(n) = \epsilon(n)\gamma(n)$

Table 11.5: Operations count for the algorithm of Table 11.4.

Equation	*Number of Multiplications*	*Number of Divisions*	*Number of Additions and Subtractions*
1	L	0	L
2	L	0	L
3	0	0	1
4	2	0	1
5[1]	1	1	0
6	0	1	0
7	L	0	L
8	2	1	0
9	1	1	1
10[2]	$2L$	1	L
11	L	0	L
12	1	0	0
13	2	0	1
14	L	0	L
15	L	0	L
16	1	0	0
Total	$8\ L + 10$	5	$7\ L + 4$

[1] $\lambda\alpha(n-1)$ was computed in step 4.
[2] $(1 - \psi(n) k_{L+1,L}(n))$ was computed in step 9.

11.6 The Fast Transversal Filter

The algorithm derived in the last section can be simplified further by using the normalized gain vector defined as

$$\tilde{\mathbf{k}}(n) = \frac{\mathbf{k}(n)}{\gamma(n)}. \tag{11.86}$$

The resulting algorithm is commonly known as the *fast transversal filter* (FTF). To see the advantage of using the normalized gain vector, we consider the update equation for the gain vector given in (11.54). Dividing both sides of this equation by $\gamma(n)$, and substituting

$$1 - k_{L+1,L}(n)\psi(n) = \frac{\gamma_{L+1}(n)}{\gamma(n)} \tag{11.87}$$

in the resulting expression, we get

$$\frac{\mathbf{k}(n)}{\gamma(n)} = \frac{\mathbf{k}_{L+1}^{(L)}(n) + \mathbf{G}(n-1)k_{L+1,L}(n)}{\gamma_{L+1}(n)}. \tag{11.88}$$

We can derive an efficient update equation for the normalized gain vector by using the definitions of the normalized gain vector as well as the normalized augmented gain vector defined as

$$\tilde{\mathbf{k}}_{L+1}(n) = \frac{\mathbf{k}_{L+1}(n)}{\gamma_{L+1}(n)} \tag{11.89}$$

in (11.88). The result is

$$\tilde{\mathbf{k}}(n) = \tilde{\mathbf{k}}_{L+1}^{(L)}(n) + \mathbf{G}(n-1)\tilde{k}_{L+1,L}(n). \tag{11.90}$$

This update equation can be implemented using only L multiplications instead of the $2L$ multiplications needed to implement the update equation for the gain vector. Converting the other steps in the iterations of Table 11.4 to variables defined using the normalized gain vectors is a straightforward task. The complete algorithm is given in Table 11.6. The only part of the derivation that is somewhat lengthy is the development of the update equation for $\tilde{\mathbf{k}}_{L+1}(n)$. This derivation is left as an exercise for the reader.

Table 11.7 lists the operation count for the fast transversal filter. Comparing the number of multiplications and divisions required for the FTF, we can see that this algorithm reduces the computational cost by approximately L multiplications over the previous version.

11.7 Numerical Error Propagation in Fast Transversal RLS Filters

Even though we have been able to obtain significant computational savings in the implementation of the recursive least-squares adaptive filters in this chapter, the algorithms of Tables

Table 11.6: The fast transversal filter.

Equation No.	Equations
1	$\phi(n) = x(n) - \mathbf{A}^T(n-1)\,\mathbf{X}(n-1)$
2	$f(n) = \phi(n)\gamma(n-1)$
3	$\alpha(n) = \lambda\alpha(n-1) + \phi(n)f(n)$
4	$\gamma_{L+1}(n) = \lambda\frac{\alpha(n-1)}{\alpha(n)}\gamma(n-1)$
5	$\tilde{\mathbf{k}}_{L+1}(n) = \begin{bmatrix} \phi(n)/\lambda\alpha(n-1) \\ \tilde{\mathbf{k}}(n-1) - \mathbf{A}(n-1)\phi(n)/\lambda\alpha(n-1) \end{bmatrix}$
6	$\mathbf{A}(n) = \mathbf{A}(n-1) + \tilde{\mathbf{k}}(n-1)\,f(n)$
7	$\psi(n) = \lambda\tilde{k}_{L+1,L}(n)\,\beta(n-1)$
8	$\gamma(n) = \dfrac{\gamma_{L+1}(n)}{1-\psi(n)\tilde{k}_{L+1,L}(n)\gamma_{L+1}(n)}$
9	$b(n) = \psi(n)\gamma(n)$
10	$\beta(n) = \lambda\beta(n-1) + \psi(n)b(n)$
11	$\tilde{\mathbf{k}}(n) = \tilde{\mathbf{k}}^{(L)}_{L+1}(n) + \mathbf{G}(n-1)\tilde{k}_{L+1,L}(n)$
12	$\mathbf{G}(n) = \mathbf{G}(n-1) + \tilde{\mathbf{k}}(n)b(n)$
13	$\epsilon(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n)$
14	$e(n) = \epsilon(n)\gamma(n)$
15	$\mathbf{W}(n) = \mathbf{W}(n-1) + \tilde{\mathbf{k}}(n)e(n)$

Table 11.7: Operations count for the algorithm of Table 11.6.

Equation	*Number of Multiplications*	*Number of Divisions*	*Number of Additions and Subtractions*
1	L	0	L
2	1	0	0
3	2	0	1
4	2	1	0
5[1]	L	1	L
6	L	0	L
7	2	0	0
8	2	1	1
9	1	0	0
10[2]	2	0	1
11	L	0	L
12	L	0	L
13	L	0	L
14	1	0	0
15	L	0	L
Total	$7\ L + 13$	3	$7\ L + 3$

[1] $\lambda\alpha(n-1)$ was computed in step 3.
[2] $\lambda\beta(n-1)$ was computed in step 7.

11.1, 11.4 and 11.6 suffer from poor numerical properties. We begin our discussion of the numerical properties of fast RLS transversal filters with a simulation example.

Example 11.1: Realization of Fast RLS Adaptive Filters

The input signal and desired response signal in this example were generated as in Example 5.2 for $\epsilon = 0.1$ and 0.001. Each adaptive filter employed ten coefficients, even though there were just two non-zero coefficients in the system that generated the desired response signal. The algorithms were implemented in MATLAB. The additions and subtractions were performed with the full precision available in MATLAB. The filter outputs were computed with full precision and then rounded to a small number of decimal digits. The output of each multiplication and division operation in the calculation of every other variable in the adaptive filter was also rounded to the same number of decimal digits. In this way, we simulated the effect that finite precision arithmetic has on the adaptive filters.

Figure 11.3a-c shows the evolution of the sum of the squared coefficient deviations from their optimal values for the fast RLS adaptive filters of Tables 11.1 and 11.4 and the fast transversal filter of Table 11.6, respectively, for $\epsilon = 0.1$, $\lambda = 0.98$, and $\delta = 0.1$. In each case, the adaptive filter calculations were rounded to the nearest third decimal place as explained above.

The corresponding results for the conventional RLS adaptive filter implemented using the maximum precision available in MATLAB is shown in Figure 11.4. It can be seen that each of the fast RLS adaptive filters exhibit numerically unstable behavior. Of the three methods, the first version we derived exhibits the slowest growth of numerical errors. However, even this method eventually diverged in this example.

In what follows, we explain the reasons for the poor numerical properties of the fast RLS adaptive filters and present a modified FTF algorithm that is robust to numerical inaccuracies in its implementation for a limited, but practical range of choices of algorithm parameters.

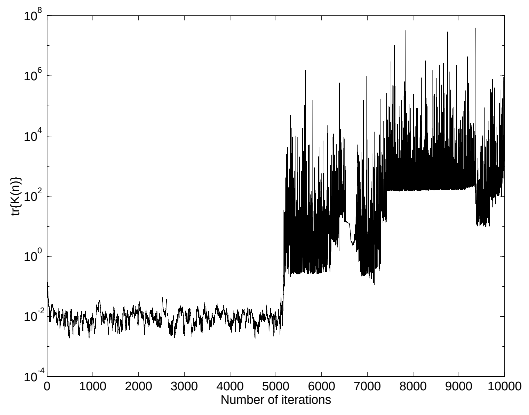
11.7.1 A Heuristic Explanation

In order to get a heuristic feel for the poor numerical properties of the fast RLS adaptive filters, we consider two sets of coefficient update equations from Table 11.1. Similar analyses can be performed for the versions in Tables 11.4 and 11.6 also. The first set involves the forward predictor coefficients and the last L elements of the augmented gain vector, denoted by $\mathbf{k}'_{L+1}(n)$. The update equations are given by

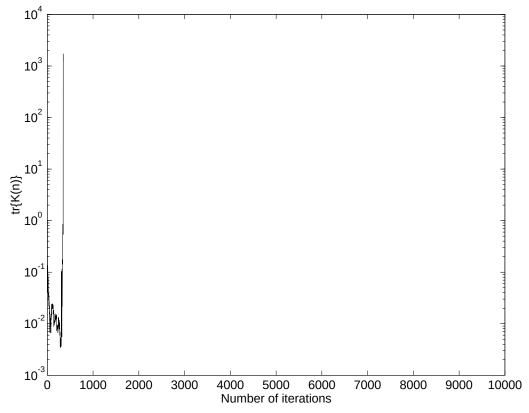
$$\mathbf{A}(n) = \mathbf{A}(n-1) + \mathbf{k}(n-1)\phi(n) \qquad (11.91)$$

and

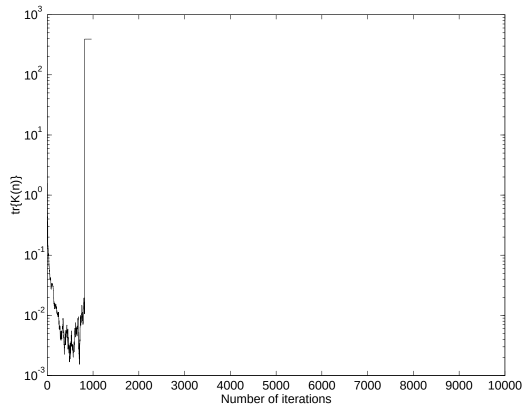
$$\mathbf{k}'_{L+1}(n) = \mathbf{k}(n-1) - r_f(n)\mathbf{A}(n)$$



(a) Algorithm of Table 11.1.



(b) Algorithm of Table 11.4.



(c) Fast Transversal Filter of Table 11.6.

Figure 11.3: Evolution of the sum of squared coefficient deviations from their optimal values for three variations of the fast transversal RLS adaptive filter.

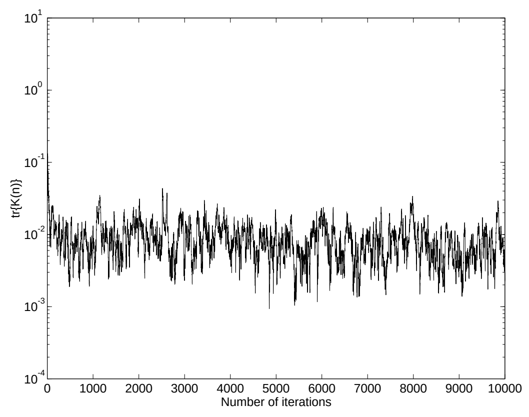


Figure 11.4: Evolution of the sum of the squared coefficient deviations from their true values for the conventional RLS adaptive filter.

$$= \mathbf{k}(n-1) - \frac{f(n)}{\alpha(n)}\left(\mathbf{A}(n-1) + \phi(n)\mathbf{k}(n-1)\right), \tag{11.92}$$

respectively. We can rewrite the above update equations using block matrix notation as

$$\begin{bmatrix} \mathbf{A}(n) \\ \mathbf{k}'_{L+1}(n) \end{bmatrix} = \begin{bmatrix} 1 & \phi(n) \\ -\dfrac{f(n)}{\alpha(n)} & 1 - \dfrac{\phi(n)f(n)}{\alpha(n)} \end{bmatrix} \begin{bmatrix} \mathbf{A}(n-1) \\ \mathbf{k}(n-1) \end{bmatrix}. \tag{11.93}$$

The determinant of the 2×2-element matrix on the right-hand-side of (11.93) is

$$\det\left(\begin{bmatrix} 1 & \phi(n) \\ -\dfrac{f(n)}{\alpha(n)} & 1 - \dfrac{\phi(n)f(n)}{\alpha(n)} \end{bmatrix}\right) = 1. \tag{11.94}$$

The above result implies that either both eigenvalues of the matrix have magnitude one, or that one of its eigenvalues is larger than one. In the latter case, any numerical error introduced in the calculation of the parameters $\mathbf{A}(n)$ and $\mathbf{k}'_{L+1}(n)$ will grow exponentially with time. In the former case, the best scenario that one could hope for is that the errors propagate without attenuation. Even in this case, the errors will accumulate linearly over time, resulting in eventual overflow. Both situations are unacceptable in practice.

It is important to recognize that the above analysis is not a rigorous one. This is because the system matrix that we analyzed depends on the coefficient vector $\mathbf{A}(n-1)$, and therefore, the actual relationship between the parameters at successive times is a nonlinear one. We neglected such dependencies to simplify our discussion.

A similar analysis can be performed on the update equations for the backward predictor coefficients and the gain vector. It is left as an exercise to show that these two vectors can be updated as

$$\left[\begin{array}{c}\mathbf{k}(n)\\ \mathbf{G}(n)\end{array}\right]=\frac{1}{\lambda\beta(n-1)}\left[\begin{array}{cc}b(n) & \beta(n)\\ \beta(n) & \beta(n)\psi(n)\end{array}\right]\left[\begin{array}{c}\mathbf{k}_{L+1}^{(L)}(n)\\ \mathbf{G}(n-1)\end{array}\right]. \tag{11.95}$$

The determinant of the 2×2-element system matrix in the above equation can be shown to be -$\beta(n)/\lambda\beta(n-1)$. Since

$$\beta(n)=\lambda\beta(n-1)+\psi(n)b(n) \tag{11.96}$$

and $\psi(n)b(n)$ is non-negative, we can see that the determinant is greater than one except when the backward prediction error is zero. This implies that at least one of the eigenvalues of the system matrix is larger than one, which means that numerical errors grow exponentially with time for these update equations also.

11.8 Stabilization of Fast Transversal Filters

Since the fast algorithms described above are exponentially unstable from a numerical point of view, they are not useful in practical applications without employing modifications that make them numerically stable. Early works on the stabilization of fast transversal RLS filters involved attempts to find indicators for the onset of instability of the adaptive filter during its operation. When the onset of instability is indicated, the algorithm is re-initialized so that the accumulated numerical errors are removed from the updates. If the system correctly identifies the onset of divergence all the time, we can run the fast transversal filter and its variants in a stable manner with the help of the "rescue" device that re-initializes the adaptive filter whenever necessary. We start our discussion with a commonly used rescue method. Experimental evidence indicates that this approach prolongs the explosive divergence of the system parameters significantly in practice. It should be recognized, however, that the rescue method is not guaranteed to completely eliminate the numerical instability problem. We then discuss a stabilization technique that employs feedback of the numerical errors. This technique appears to be the most successful in combating the poor numerical properties of the fast RLS adaptive transversal filters.

11.8.1 The Rescue Method

Experiments with the fast transversal filter have indicated that the behavior of $\gamma(n)$ is a reliable predictor of the onset of divergence of the system. The value of $\gamma(n)$ should always lie between zero and one when the adaptive filter is implemented with infinite amount of precision. Therefore, the only way in which $\gamma(n)$ can become negative or exceed one is from the accumulation of numerical errors. It has been observed that $\gamma(n)$ becomes negative usually just before the algorithm diverges [Lin 1984]. Consequently, the *rescue method* for stabilizing the fast transversal filter re-initializes the algorithm whenever $\gamma(n)$ becomes negative. In variations of the fast RLS adaptive filter in which the variable $\gamma(n)$ is not calculated explicitly, the onset of divergence can be detected using other variables. For example, in the original algorithm of Table 11.1, it can be shown that $0 \leq 1 - \psi(n)r_b(n) \leq 1$ when the calculations are performed with infinite precision. Consequently, the rescue procedure can be implemented by monitoring the behavior of $1-\psi(n)r_b(n)$ in that algorithm. During the re-initialization, the coefficient vector $\mathbf{W}(n)$ is left unchanged, and the parameters $\mathbf{A}(n)$, $\mathbf{G}(n)$ and $\mathbf{k}(n)$ are set to zero. The variable $\gamma(n)$ is set to one, the least-squares value of the forward prediction error is set to $\alpha(n) = \lambda^L\delta$, and the least-squares value of the backward prediction error is set to $\beta(n) = \delta$ as discussed in Section 11.4.1.

Even though the rescue methods often detect the onset of instability and make necessary corrections to prevent divergence of the fast RLS adaptive transversal filters, the approach is only a heuristic method, and cannot guarantee the complete elimination of the inherent numerical problems of the algorithms. The divergence of the adaptive filters observed in Example 11.1 may be delayed by the use of rescue methods. However, such divergence eventually occurs unless more effective stabilization techniques are incorporated in the algorithm. Consequently, the rescue method is not recommended as a serious candidate for stabilizing the fast RLS adaptive transversal filters.

11.8.2 Stabilization Using Error Feedback

The main idea behind the stabilization technique employing error feedback is that of introducing redundancy in the calculation of several variables that are known to propagate numerical errors in an unstable manner in the fast transversal filter. For example, consider the *a priori* backward prediction error. Direct evaluation of $\psi(n)$ can be performed as

$$\psi^f(n) = x(n-L) - \mathbf{G}^T(n-1)\mathbf{X}(n). \tag{11.97}$$

We have also seen from the derivation of the fast transversal filter that $\psi(n)$ can also be evaluated as

$$\psi^s(n) = \lambda\tilde{k}_{L+1,L+1}(n)\beta(n-1). \tag{11.98}$$

In the above equations, we have used the superscripts f and s to indicate that the variable $\psi(n)$ has been calculated using direct filtering operation and by manipulating scalar variables, respectively. In an infinite precision environment, the values of $\psi^f(n)$ and $\psi^s(n)$ are identical. However, finite precision calculations will in general result in different values for $\psi^f(n)$ and $\psi^s(n)$ because of the numerical inaccuracies in the computations. In the stabilized version of the FTF, a linear combination of the two realizations of the *a priori* backward prediction error is computed as

$$\psi(n) = K\psi^f(n) + (1-K)\psi^s(n). \tag{11.99}$$

Then, $\psi(n)$ is employed in the calculation of subsequent variables.

We can assume with a reasonable amount of confidence that the numerical errors introduced into the two sets of calculations of the same variable are uncorrelated with each other. In the simplest of terms, the averaging in (11.99) allows for the propagation of the numerical errors in $\psi^f(n)$ and $\psi^s(n)$ with attenuation rather than amplification, thus permitting numerically stable operation of the overall system. It is important to recognize that the infinite precision realization of the above approach provides the exact solution to the RLS adaptive filtering problem.

The computation in (11.99) can be thought of as a form of error feedback in the following manner. Let us rewrite this equation as

$$\psi(n) = \psi^s(n) + K(\psi^f(n) - \psi^s(n)). \tag{11.100}$$

If we assume that the filtered form of the *a priori* estimation error is relatively accurate, we can consider the quantity $\psi^f(n) - \psi^s(n)$ to be a measurement of the numerical error in the computation of $\psi(n)$. Consequently, the structure of (11.99) effectively feeds back a measurement of the numerical error to stabilize the error propagation mechanism. This system thus uses the principles of control theory to achieve stable operation of the fast transversal filter.

There are three main variables in the fast transversal filter that require stabilization as described above. They are the *a priori* backward prediction error $\psi(n)$, the last element of the normalized, augmented gain vector $\tilde{k}_{L+1,L}(n)$, and $\gamma^{-1}(n)$. Because of the computational simplifications possible and the existence of several methods for calculating $\gamma(n)$ and $\gamma^{-1}(n)$, the stabilized FTF propagates $\gamma^{-1}(n)$ as well as $\gamma(n)$ in its operation. It is left as an exercise for the reader to show that $\gamma^{-1}(n)$ can be computed as

$$\gamma^{-f}(n) = 1 - \tilde{\mathbf{k}}^T(n)\mathbf{X}(n) \tag{11.101}$$

as well as

$$\gamma^{-s}(n) = \gamma_{L+1}^{-1}(n) - \psi(n)\tilde{k}_{L+1,L}(n). \tag{11.102}$$

Furthermore, when the adaptive filter is initialized using the soft constraint, $\gamma(n)$ is given by

$$\gamma(n) = \lambda^L \beta(n)\alpha^{-1}(n). \tag{11.103}$$

Using equations 3, 4 and 5 in Table 11.6, we can show that $\gamma_{L+1}^{-1}(n)$ can be updated recursively as

$$\gamma_{L+1}^{-1}(n) = \gamma^{-1}(n-1) + \tilde{k}_{L+1,0}(n)\phi(n), \tag{11.104}$$

where $\tilde{k}_{L+1,1}(n)$ is the first element of $\tilde{\mathbf{k}}_{L+1}(n)$. Starting from the equations of the fast transversal filter, we can show that $\tilde{k}_{L+1,L}(n)$ can be computed as

$$\tilde{k}_{L+1,L}^f(n) = \frac{\psi(n)}{\lambda\beta(n-1)} \tag{11.105}$$

and

$$\tilde{k}^s_{L+1,L}(n) = \tilde{k}_L(n-1) - \tilde{k}_{L+1,0}(n)a_L(n-1), \tag{11.106}$$

where $\tilde{k}_L(n-1)$ is the last element of $\tilde{\mathbf{k}}(n-1)$ and $a_L(n-1)$ is the last element of the forward predictor coefficient vector $\mathbf{A}(n-1)$ at time $n-1$. Finally, we leave it as an exercise to show that the inverse of the least-squares forward and backward prediction error values can be recursively updated as

$$\alpha^{-1}(n) = \lambda^{-1}\alpha^{-1}(n-1) - \tilde{k}^2_{L+1,0}(n)\gamma_{L+1}(n) \tag{11.107}$$

and

$$\beta^{-1}(n) = \lambda^{-1}\beta^{-1}(n-1) - \tilde{k}^2_{L+1,L}(n)\gamma_{L+1}(n), \tag{11.108}$$

respectively.

The *a priori* backward prediction error $\psi(n)$ is employed for updating several variables in the fast transversal filter. To obtain more freedom in controlling the error dynamics in the FTF, the stabilized algorithm employs different linear combinations of $\psi^s(n)$ and $\psi^f(n)$ for the different update equations. The complete algorithm is shown in Table 11.8. Six different feedback constants K_1 through K_6 are employed in the algorithm.

Slock and Kailath [Slock 1991] performed an extensive analysis on the numerical properties of this adaptive filter. They showed that the algorithm is exponentially stable for all values of L and for λ in the range $(1-1/2L, 1)$ when the feedback constants are properly chosen. They conducted a numerical optimization to evaluate the feedback constants and suggested the use of

$$[K_1\ \ K_2\ \ K_3\ \ K_4\ \ K_5\ \ K_6] = [1.5\ \ 2.5\ \ 0\ \ 0\ \ 1\ \ 1]. \tag{11.109}$$

The number of operations required to implement the stabilized FTF is tabulated in Table 11.9. When the feedback constants are chosen as above, the computational complexity reduces to $O(8L)$ arithmetical operations per iteration from the $O(9L)$ complexity when arbitrary feedback constants are employed. While the computational complexity of this algorithm is somewhat higher than that of the FTF, it is still linear in the number of coefficients. Since the algorithm is numerically stable, it is significantly more useful than the three other algorithms derived in this chapter.

A MATLAB function that implements the stabilized FTF is given in Table 11.10. The feedback parameters suggested in (11.109) have been hard-coded into this function. They may be changed for the purposes of analysis and experimentation.

Example 11.2: Evaluation of the Stabilized FTF

In this example, we repeat the experiment of Example 11.1 with the stabilized fast transversal filter. Figure 11.5 displays the evolution of the sum of the squared coefficient deviations from their optimal values for 10,000 iterations. The adaptive filter employed a precision of five decimal

Table 11.8: The stabilized fast transversal filter.

Equation No.	Equations
1	$\phi(n) = x(n) - \mathbf{A}^T(n-1)\,\mathbf{X}(n-1)$
2	$\tilde{k}_{L+1,0}(n) = \lambda^{-1}\alpha^{-1}(n-1)\phi(n)$
3	$\tilde{\mathbf{k}}_{L+1}(n) = \begin{bmatrix} \tilde{k}_{L+1,0}(n) \\ \tilde{\mathbf{k}}(n-1) - \mathbf{A}(n-1)\tilde{k}_{L+1,0}(n) \end{bmatrix}$; Except $\tilde{k}_{L+1,L}(n)$
4	$\gamma^{-1}_{L+1}(n) = \gamma^{-1}(n-1) + \tilde{k}_{L+1,0}(n)\phi(n)$
5	$\psi^f(n) = x(n-L) - \mathbf{G}^T(n-1)\mathbf{X}(n)$
6	$\tilde{k}^f_{L+1,L}(n) = \lambda^{-1}\beta^{-1}(n-1)\psi^f(n)$
7	$\tilde{k}^s_{L+1,L}(n) = k_L(n-1) - a_L(n-1)\tilde{k}_{L+1,0}(n)$
8	$\psi^s(n) = \lambda\tilde{k}^s_{L+1,L}(n)\beta(n-1)$
9	$\psi^{(i)}(n) = K_i(\psi^f(n) - \psi^s(n)) + \psi^s(n)$; i =1, 2, 5
10	$\tilde{k}_{L+1,L}(n) = K_4(\tilde{k}^f_{L+1,L}(n) - \tilde{k}^s_{L+1,L}(n)) + \tilde{k}^s_{L+1,L}(n)$
11	$f(n) = \phi(n)\gamma(n-1)$
12	$\mathbf{A}(n) = \mathbf{A}(n-1) + \tilde{\mathbf{k}}(n-1)\,f(n)$
13	$\tilde{\mathbf{k}}(n) = \tilde{\mathbf{k}}^{(L)}_{L+1}(n) + \mathbf{G}(n-1)\tilde{k}_{L+1,L}(n)$
14	$\gamma^{-s}(n) = \gamma^{-1}_{L+1}(n) - \psi^{(5)}(n)\tilde{k}_{L+1,L}(n)$
15	$\gamma^{-f}(n) = 1 + \tilde{\mathbf{k}}^T(n)\mathbf{X}(n)$
16	$\gamma^{-i}(n) = K_3(\gamma^{-f}(n) - \gamma^{-s}(n)) + \gamma^{-s}(n)$
17	$\alpha^{-1}(n) = \lambda^{-1}\alpha^{-1}(n-1) - \tilde{k}^2_{L+1,0}(n)\gamma_{L+1}(n)$
18	$b^{(i)}(n) = \psi^{(i)}(n)\gamma^s(n)$; i =1, 2
19	$\mathbf{G}(n) = \mathbf{G}(n-1) + \tilde{\mathbf{k}}(n)b^{(1)}(n)$
20	$\beta(n) = \lambda\beta(n-1) + \psi^{(2)}(n)b^{(2)}(n)$
21	$\beta^{-1}(n) = \lambda^{-1}\beta^{-1}(n-1) - \tilde{k}^2_{L+1,L}(n)\gamma_{L+1}(n)$
22	$\gamma(n) = K_6(\lambda^L\beta(n)\alpha^{-1}(n) - \gamma^i(n)) + \gamma^i(n)$
23	$\epsilon(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n)$
24	$e(n) = \epsilon(n)\gamma(n)$
25	$\mathbf{W}(n) = \mathbf{W}(n-1) + \tilde{\mathbf{k}}(n)e(n)$

Table 11.9: Operations count for the algorithm of Table 11.8.

Equation	*Number of Multiplications*	*Number of Divisions*	*Number of Additions and Subtractions*
1	L	0	L
2	2	0	0
3	L-1	0	L-1
4	1	0	1
5	L	0	L
6	2	0	0
7	1	0	1
8	2	0	0
9	3	0	4
10	1	0	2
11	1	0	0
12	L	0	L
13	L	0	L
14	1	0	1
15	L	0	L
16	1	0	2
17[1]	3	1	1
18[2]	2	1	0
19	L	0	L
20	2	0	1
21	3	0	1
22[3]	3	1	2
23	L	0	L
24	1	0	0
25	L	0	L
Tota$L$	$9\ L + 28$	3	$9\ L + 15$

[1] $1/\gamma_{L+1}(n)$ is computed first.
[2] $1/\gamma^{-s}(n)$ is computed first.
[3] $\gamma^{i}(n)$ is computed first.

Table 11.10: A MATLAB function that implements the stabilized fast transversal filter.

```
        function[W,dhat,e] = sftf(lambda,delta,W0,x,d);
%
%       This function adapts the coefficients of an FIR filter
%       using the fast transversal filter with error feedback
%       stabilization as given in Table 11.8.
%
%      Input parameters:
%
%      lambda   = forgetting factor
%      delta    = initialization constant for autocorrelation matrix
%      W0       = Initial value of W(0) coefficients (L x 1)
%      x        = input signal (num_iter x 1)
%      d        = desired response signal (num_iter x 1)
%
%      Output of program:
%
%      W        = Evolution of coefficients (L x (num_iter + 1))
%      dhat     = output of adaptive filter (num_iter x 1)
%      e        = error of adaptive filter (num_iter x 1)
%
%
%       Initialization
%
        L = length(W0);
        lambda1 = 1/lambda;
        lambdaL = lambda^L;
        start_iter = 1;
        end_iter = min([length(x),length(d)]);
        W1 = W0;
        X = zeros(size(W1));
        A = zeros(size(W1));
        G = zeros(size(W1));
        ktilde = zeros(size(W1));
        alpha = delta * lambdaL;
        alpha1 = 1/alpha;
        beta = delta;
        beta1 = 1/beta;
        gamma = 1;
        gamma1 = 1/gamma;
        K = [1.5,2.5,0,0,1,1];
%
%       Recursions
%
   for n = start_iter:end_iter,
        xL1 = X(L);
        eta = x(n) - A'*X;
        rf1 = lambda1*alpha1*eta;
        ktildeL1(1:L) = [rf1; ktilde(1:L-1) - A(1:L-1)*rf1];
        gammaL11 = gamma1 + rf1*eta;
        gammaL1 = 1/gammaL11;
        X = [x(n);X(1:L-1)];
        psif = xL1 - G'*X;
        rb1f = lambda1*beta1*psif;
        rb1s = ktilde(L) - A(L)*rf1;
        psis = lambda*rb1s*beta;
        psierr = psif-psis;
        psi1 = K(1)*psierr + psis;
        psi2 = K(2)*psierr + psis;
        psi5 = K(5)*psierr +   psis;
        rb1err = rb1f - rb1s;
        ktildeL1(L+1) = K(4)*rb1err + rb1s;
        f = eta*gamma;
        A = A + ktilde*f;
        ktilde = ktildeL1(1:L)' + ktildeL1(L+1)*G;
        gamma1s = gammaL11 - psi5*ktildeL1(L+1);
        gammas = 1/gamma1s;
        gamma1f = 1 + ktilde'*X;
        gamma1err = gamma1f - gamma1s;
        gamma1 = K(3)*gamma1err + gamma1s;
        alpha1 = lambda1*alpha1 - rf1*rf1*gammaL1;
        b1 = psi1*gammas;
        b2 = psi2*gammas;
        G = G + ktilde*b1;
        beta = lambda*beta + psi2*b2;
        beta1 = lambda1*beta1 - ktildeL1(L+1)*ktildeL1(L+1)*gammaL1;
        gamma1inv = 1/gamma1;
        gamma = lambdaL*beta*alpha1;
        gammaerr = gamma - gamma1inv;
        gamma = K(6)*gammaerr + gamma1inv;
        epsilon = d(n) - W1'*X;
        e(n) = epsilon*gamma;
        W1 = W1 + ktilde*e(n);
        dhat = d(n) - e(n);
        W(:,n) = W1;
    end;
```

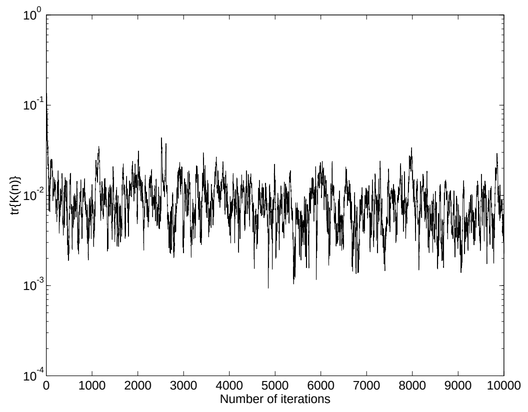


Figure 11.5: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.1$, $\lambda = 0.98$, $\delta = 0.1$ and finite precision accuracy of five decimal places.

places and the same parameter values as those employed in Example 11.1. We can see from this figure that the stabilized FTF operates in a robust manner, at least for the first 10,000 iterations. This adaptive filter was also monitored for a million iterations, and it did not show any signs of numerically-unstable behavior over this extended period. The experiment was repeated for a precision corresponding to three decimal places. The corresponding performance curves are shown in Figure 11.6. This experiment was also run for a million iterations without any signs of instability.

In the rest of this example, we discuss the effects of input signal statistics and the choice of the parameters for implementing the stabilized FTF on the numerical properties of the system. The stabilized FTF exhibited divergence when the adaptive filter was implemented with a precision of two decimal places. The performance curve for this case is given in Figure 11.7. This indicates that a certain amount of accuracy is required for the stabilized FTF to operate in a numerically-stable manner. When the input signal was generated using $\epsilon = 0.001$ and all other experimental parameters were the same as in the previous experiment, the adaptive filter required more accurate calculations provided by at least four decimal places of precision, indicating that the required precision varies with the input signal characteristics. The performance curves for this input signal and finite precision arithmetic corresponding to three and four decimal places of accuracy are displayed in Figure 11.8. We can see from these figures that the algorithm exhibited divergence

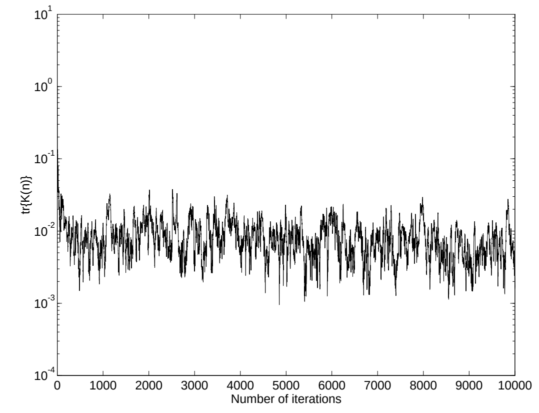


Figure 11.6: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.1$, $\lambda = 0.98$, $\delta = 0.1$ and finite precision accuracy of three decimal places.

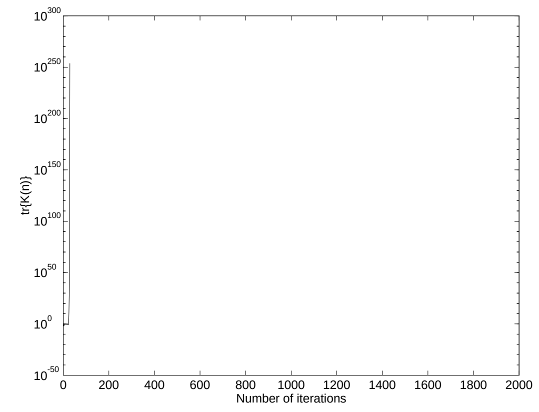


Figure 11.7: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.1$, $\lambda = 0.98$, $\delta = 0.1$ and finite precision accuracy of two decimal places.

for the lower-precision realization. Recall that a choice of $\epsilon = 0.001$ results in a more narrowband signal than that obtained from choosing $\epsilon = 0.1$.

Figure 11.9 depicts the performance of the adaptive filter for $\epsilon = 0.1$, $\delta = 0.01$, $\lambda = 0.98$ and a finite precision accuracy of three decimal places. The coefficients diverged in this case, suggesting that the stabilized FTF is sensitive to the choice of the initialization parameter. Choosing δ smaller than 0.04 resulted in unstable behavior of the stabilized FTF for $\epsilon = 0.1$. In general, the rescue device discussed in Section 11.8.1 did not prevent divergence in this situation.

Finally, we present the results of an experiment with $\epsilon = 0.1$, $\lambda = 0.95$, $\delta = 0.1$ and the maximum precision available in the MATLAB environment in Figure 11.10. Our choice of λ in this case is at the boundary of the range of stable operation determined by the analysis in [Slock 1991]. We can see that the algorithm diverged in spite of the higher precision employed in the calculations. In the experiments involving three-digit precision, the system was unstable when a value of λ smaller than 0.97 was chosen.

It is worthwhile reiterating the significance of the experimental results presented in the above example. The stabilized fast transversal filter is capable of operating in a stable manner for most input signals if the parameters are chosen carefully, and the implementation allows the precision required to provide stable operation. Furthermore, the range of parameter values and the precision required for most signals are typical choices in practical applications. Consequently, the stabilized FTF is a viable and efficient alternative to the conventional RLS adaptive filter.

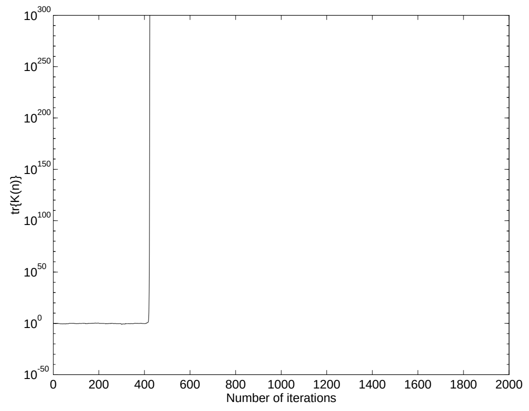
Some Remarks on the Stabilized FTF

As remarked earlier, the stabilized FTF is the most reliable of the fast transversal RLS adaptive filters available today. However, this statement does not imply that this adaptive filter operates in a stable manner for all possible input signals and parameter values. Based on the experimental results presented in Example 11.2 and the analysis in [Slock 1991], we can make the following statements regarding the stabilized FTF:

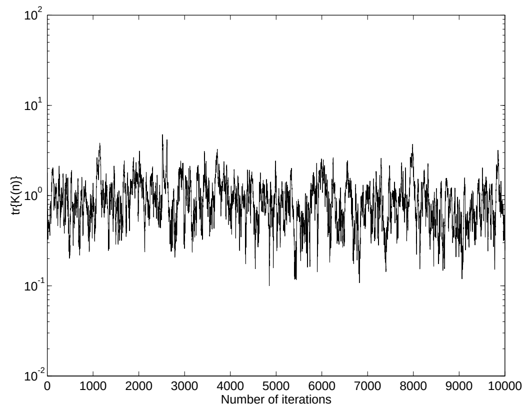
1. The choice of λ should be close to one such that $1 - 1/2L < \lambda < 1$. In general, it is safer to choose a value of λ somewhat higher than the lower bound described above.

2. A small choice of δ, the initialization constant, can precipitate unstable behavior of the adaptive filter during the transient stage. In practice, we must choose δ to be much larger than the smallest positive value that can be represented in the finite precision arithmetic. Reliable values of δ are typically at least as large as a tenth of the input signal power.

3. The stabilized fast transversal filter is not guaranteed to operate in a numerically stable manner for very low-precision arithmetic. The low precision of the arithmetic employed in the calculations may make the least-squares autocorrelation matrix effectively singular, and this situation may drive the system to instability even when the rest of the



(a) Accuracy of three decimal places.



(b) Accuracy of four decimal places.

Figure 11.8: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.001$, $\lambda = 0.98$ and $\delta = 0.1$.

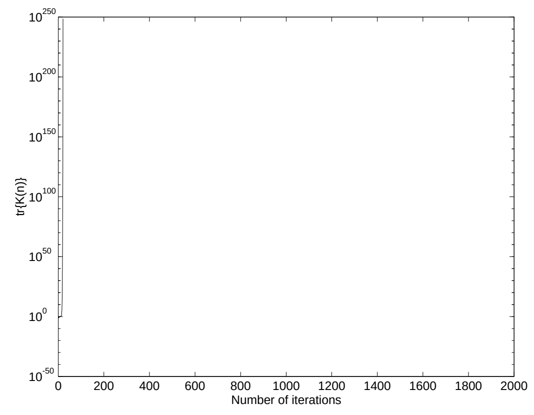


Figure 11.9: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.1$, $\lambda = 0.98$, $\delta = 0.01$ and finite precision accuracy of three decimal places.

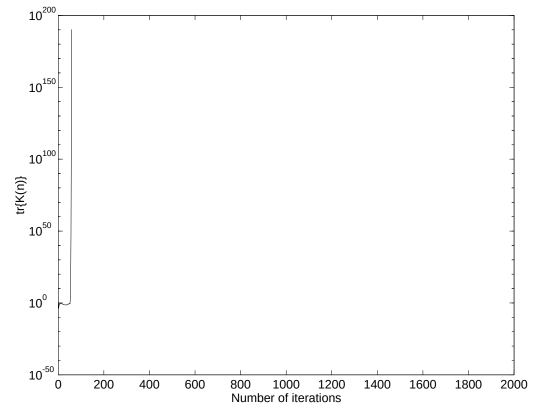


Figure 11.10: Performance of the stabilized FTF in Example 11.2 with $\epsilon = 0.1$, $\lambda = 0.95$, $\delta = 0.1$ and the maximum precision available in the MATLAB environment.

parameters are chosen in accordance with the recommendations above. The accuracy of the arithmetic required for the stable operation of the stabilized FTF depends on the characteristics of the input signal. In general, narrowband signals require higher precision arithmetic than wideband input signals.

11.9 Main Points of This Chapter

- Single-channel exponentially-weighted RLS adaptive filters can be implemented efficiently using fast transversal RLS adaptive filters. The computational complexity of such algorithms is an order of magnitude smaller than that of the conventional RLS adaptive filters.
- The first three versions of the fast transversal RLS adaptive filters derived in this chapter are numerically unstable. The poor numerical properties can be traced to certain unstable transformations employed in these algorithms.
- The most successful method for combating the poor numerical properties of fast transversal RLS adaptive filters uses error feedback techniques to stabilize the system. The stabilized fast transversal filter can operate in a numerically stable manner for a limited range of λ and proper choices of the initialization constant δ, provided that the adaptive filter is implemented with adequate precision. The required precision is a function of the input signal statistics.
- The fast RLS transversal adaptive filters can be extended to the case of multichannel signals. The derivations are included as exercises at the end of this chapter.

11.10 Bibliographical Notes

Derivation of Fast RLS Adaptive Filters. The first successful derivation of an exponentially-windowed adaptive RLS filter with $O(L)$ computational complexity was performed by Morf, Ljung and Falconer [Morf 1978]. Their algorithm is commonly known as the fast Kalman algorithm, and the single channel version of the fast Kalman algorithm is identical to the adaptive filter given in Table 11.1. The derivations in [Morf 1978] also included the case of multichannel filters. It is possible to derive an exponentially-weighted RLS adaptive filter that uses $O(LK^2)$ arithmetical operations for a K-channel estimation problem with L-sample memory It appears that the first published derivation of the more efficient algorithm of Table 11.4 was in [Proakis 1983]. Cioffi and Kailath also derived similar results independently [Cioffi 1983]. Carayannis, Manolakis and Kalouptsidis [Carayannis 1983] were the first to derive a single-channel RLS adaptive filter with $O(7L)$ complexity. Their algorithm is known as the *Fast A Posteriori Error Sequential Technique* (FAEST). The fast transversal filter of Table 11.6 is only slightly different from the FAEST algorithm and was derived by Cioffi and Kailath [Cioffi 1984]. An alternate description of the development of fast RLS adaptive filters is provided in [Alexander 1987]. A comprehensive review of fast RLS adaptive filters may be found in [Carayannis 1986].

Variations of the Transversal Structure. Fast algorithms for RLS Transversal filters have been derived for the sliding window cost function by Cioffi and Kailath [Cioffi 1985]. They also derived normalized versions of the fast transversal filter that requires $O(11L)$ arithmetical operations per iteration. Fast transversal RLS adaptive filters with a time-varying weighting function $\lambda(n)$ are presented in [Pasupathy 1988] and [Slock 1989].

Numerical Stability Issues. From the very beginning, it was recognized that the fast RLS filters described in Tables 11.1, 11.4 and 11.6 had poor numerical properties [Cioffi 1984, Ling 1984]. Heuristic arguments similar to those in Section 11.7.1 for explaining the poor numerical properties of the fast transversal RLS filters may be found in [Cioffi 1984]. Ljung and Ljung [Ljung 1985] demonstrated with a numerical example that the the fast Kalman algorithm is numerically unstable.

The use of the rescue variable to combat the numerical problems of fast transversal RLS filters was proposed by Lin [Lin 1984]. Bellanger [Bellanger 1987] suggested introducing a leakage term into the update equation for the prediction error powers so that these variables are bounded from below by a finite positive value. Evci and Bellanger [Evci 1988] formulated another approximate solution by forcing the mirror image symmetry of the backward and forward predictor coefficients. According to [Slock 1991], Cioffi attempted to derive a leaky version of the RLS adaptive filter similar to the Leaky LMS adaptive filter to combat its numerical problems. However, he found that the performance penalty was too great to be of use. Binde has recently published a fast RLS algorithm that employs leakage [Binde 1995]. One potential drawback of this approach is that the algorithm that results is no longer the exact solution to the least-squares problem, but only an approximation.

Botto and Moustakides [Botto 1986, Botto 1987, Botto 1989] appear to be the first to try stabilization of the fast RLS filters by introducing computational redundancy in certain calculations and then applying error feedback techniques to modify the dynamical behavior of the error propagation. Bennalal and Gilloire [Bennalal 1988] also derived a similar method. Slock and Kailath [Slock 1991] extended the results to develop what is known as the stabilized FTF. They also provided a relatively rigorous analysis to identify the causes of numerical problems in fast transversal filters and to show that the stabilized FTF operates in a numerically robust manner when the parameters of the adaptive filter are chosen carefully.

11.11 Exercises

11.1. *Another Interpretation of* $\gamma(n)$:

a. Show that

$$\gamma(n) = \sum_{k=1}^{n} \lambda^{n-k} \left(\pi_n(k) - \mathbf{k}^T(n)\mathbf{X}(k)\right)^2 .$$

That is, $\gamma(n)$ is the exponentially-weighted least-squares error in estimating $\pi_n(k)$ using $\mathbf{X}(k)$ and the coefficient vector $\mathbf{k}(n)$.

b. Show that $0 \leq \gamma(n) \leq 1$.

11.2. *Another Expression for $\gamma(n)$:* Show that

$$\gamma(n) = \lambda^L \beta(n)\alpha^{-1}(n)$$

when the fast RLS adaptive filter employs the soft constrained initialization procedure. *Hint:* First show using (11.77) and (11.78) that

$$\frac{\gamma(n)}{\gamma(n-1)} = \frac{\beta(n)\alpha(n-1)}{\beta(n-1)\alpha(n)}.$$

Then solve for $\gamma(n)$ by evaluating

$$\prod_{k=2}^{n} \frac{\gamma(k)}{\gamma(k-1)}$$

and substituting the initial values of the various parameters in the resulting expression.

11.3. *Updating the Coefficients Using the Normalized Gain Vector:* Show that

$$\mathbf{W}(n) = \mathbf{W}(n-1) + \tilde{\mathbf{k}}(n)e(n).$$

Derive similar results for the forward and backward predictor coefficients.

11.4. *Updating the Normalized, Augmented Gain Vector:* Derive the update equation for the normalized augmented gain vector given by

$$\tilde{\mathbf{k}}_{L+1}(n) = \begin{pmatrix} \phi(n)/\lambda\alpha(n-1) \\ \tilde{\mathbf{k}}(n-1) - \mathbf{A}(n-1)\phi(n)/\lambda\alpha(n-1) \end{pmatrix}.$$

Hint: Substitute the coefficient update equation for $\mathbf{A}(n)$ in (11.77) and then use the relationship

$$\frac{\lambda\alpha(n-1)}{\alpha(n)} = \frac{\gamma_{L+1}(n)}{\gamma(n-1)}.$$

11.5. *Derivation of the Fast Transversal Filter:* Derive the fast transversal filter of Table 11.6

11.6. *A Constrained Fast RLS Adaptive Filter:* Recall from our discussion in Chapter 3 that the minimum mean-square error forward and backward predictor coefficients satisfy the mirror image symmetry given by

$$a_k = g_{L-k} \ ; \text{ for } 1 \le k \le L$$

when the input signal and the desired response signal are jointly wide-sense stationary. Derive a fast RLS adaptive filter that imposes the mirror image symmetry on the forward and backward predictor coefficients at each iteration. Compare the computational complexity of your adaptive filter with that of the fast RLS adaptive filters derived in this chapter. Explain the possible advantages and disadvantages of the constrained RLS adaptive filter in practical applications.

11.7. *Fast Sliding Window RLS Adaptive Filter:* Derive a fast RLS solution for the estimation problem that minimizes

$$J(n) = \sum_{k=0}^{M-1} \Big(d(n-k) - \mathbf{W}^T(n)\mathbf{X}(n-k) \Big)^2$$

at each time.

Hint: Start with the two-step solution for Exercise 10.3 and extend the result by deriving a fast realization for each step.

11.8. *Calculation of $\gamma_{L+1}^{-1}(n)$:* Show that

$$\gamma_{L+1}^{-1}(n) = \gamma^{-1}(n-1) + \tilde{k}_{L+1,0}(n)\phi(n).$$

Hint: Substitute $k_{L+1,0}(n) = f(n)/\alpha(n)$ in (11.76), divide both sides of the result by $\gamma(n-1)\gamma_{L+1}(n)$ and simplify.

11.9. *Calculation of $\gamma^{-1}(n)$:* Show, starting from the definition of $\gamma(n)$, that

$$\gamma^{-1}(n) = 1 + \tilde{\mathbf{k}}^T(n)\mathbf{X}(n).$$

11.10. *Calculation of the First and Last Elements of $\tilde{\mathbf{k}}_{L+1}(n)$:* Show that the first and last elements of $\tilde{\mathbf{k}}_{L+1}(n)$ are given by

$$\tilde{k}_{L+1,0}(n) = \frac{\phi(n)}{\lambda\alpha(n-1)}$$

and

$$\tilde{k}_{L+1,L}(n) = \frac{\psi(n)}{\lambda\beta(n-1)},$$

respectively.

Table 11.11: The Fast *A posteriori* Error Sequential Technique.

Equation No.	*Equations*
1	$\phi(n) = x(n) - \mathbf{A}^T(n-1)\,\mathbf{X}(n-1)$
2	$f(n) = \phi(n)/\gamma^{-1}(n-1)$
3	$\alpha(n) = \lambda\alpha(n-1) + \phi(n)f(n)$
4	$\tilde{\mathbf{k}}_{L+1}(n) = \begin{bmatrix} \tilde{\mathbf{k}}^{(L)}_{L+1}(n) \\ \tilde{k}_{L+1,L}(n) \end{bmatrix} = \begin{bmatrix} f(n)/\alpha(n) \\ \tilde{\mathbf{k}}(n-1) - \mathbf{A}(n-1)f(n)/\alpha(n) \end{bmatrix}$
5	$\mathbf{A}(n) = \mathbf{A}(n-1) + \tilde{\mathbf{k}}(n-1)\,f(n)$
6	$\psi(n) = \lambda\tilde{k}_{L+1,L}(n)\,\beta(n-1)$
7	$\tilde{\mathbf{k}}(n) = \tilde{\mathbf{k}}^{(L)}_{L+1}(n) + \mathbf{G}(n-1)\tilde{k}_{L+1,L}(n)$
8	$\gamma^{-1}_{L+1}(n) = \gamma^{-1}(n) + \tilde{k}_{L+1,0}(n)\psi(n)$
9	$\gamma^{-1}(n) = \gamma^{-1}_{L+1}(n) - \psi(n)\tilde{k}_{L+1,L}(n)$
10	$b(n) = \psi(n)/\gamma^{-1}(n)$
11	$\beta(n) = \lambda\beta(n-1) + \psi(n)b(n)$
13	$\mathbf{G}(n) = \mathbf{G}(n-1) + \tilde{\mathbf{k}}(n)b(n)$
14	$\epsilon(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n)$
15	$e(n) = \epsilon(n)/\gamma^{-1}(n)$
16	$\mathbf{W}(n) = \mathbf{W}(n-1) + \tilde{\mathbf{k}}(n)e(n)$

11.11. *Computation of the Reciprocals of the Least-Squares Prediction Error Values:* Show that $\alpha^{-1}(n)$ and $\beta^{-1}(n)$ can be updated as

$$\alpha^{-1}(n) = \lambda^{-1}\alpha^{-1}(n-1) - \tilde{k}^2_{L+1,0}(n)\gamma_{L+1}(n)$$

and

$$\beta^{-1}(n) = \lambda^{-1}\beta^{-1}(n-1) - \tilde{k}^2_{L+1,L}(n)\gamma_{L+1}(n).$$

Hint: For the first expression, manipulate the update equation for $\alpha(n)$ using the fact that $\tilde{k}_{L+1,0}(n) = \phi(n)/\lambda\alpha(n-1) = \alpha^{-1}(n)f(n)/\gamma_{L+1}(n)$.

11.12. *Bounds on the Rescue Variable:* Show that

$$0 \le 1 - \psi(n)r_b(n) \le 1.$$

11.13. *The FAEST Algorithm:* Show that the fast *a posteriori* error sequential technique (FAEST) [Carayannis 1984] given in Table 11.11 is an exact implementation of the exponentially-weighted RLS adaptive filter by establishing the accuracy of the steps that were not derived in this chapter.

11.14. *Fast RLS Multichannel Filters:* Consider a multichannel estimation problem in which we desire to estimate a scalar signal $d(n)$ using the most recent L samples of a two channel signal $\mathbf{x}(n) = [x_1(n)\ x_2(n)]^T$, *i.e.,* the adaptive filter has $2L$ coefficients, and the input vector receives two new samples at each time. Derive a fast transversal filter for this problem.

Hint: Mimic the derivation of the single channel adaptive filters. For the two channel problem, the coefficient vectors $\mathbf{W}(n)$ and $\mathbf{k}(n)$ are $2L$-element vectors and $\mathbf{A}(n)$ and $\mathbf{G}(n)$ are $L\times 2$-element matrices. Similarly, the autocorrelation matrices for the forward and backward prediction error signals $\alpha(n)$ and $\beta(n)$ are 2×2-element matrices.

11.15. *Fast RLS Multichannel Filters With Different Number of Coefficients in Each Channel:* Consider the two-channel problem of Exercise 11.5, with the difference that the first channel contains L_1 coefficients that scales the samples $x_1(n)$, $x_1(n-1)$, $\cdots$, $x_1(n-L_1-1)$, and the second channel contains L_2 coefficients that scales $x_2(n)$, $x_2(n-1)$, $\cdots$, $x_2(n-L_2-1)$. Derive a fast RLS adaptive filter for this case.

Hint: Define the augmented input vector as

$$\mathbf{X}_A(n) = \begin{bmatrix} \mathbf{X}(n) \\ x_1(n-L_1) \\ x_2(n-L_2) \end{bmatrix} = \mathbf{S} \begin{bmatrix} x_1(n) \\ x_2(n) \\ \mathbf{X}(n-1) \end{bmatrix},$$

where $\mathbf{S}$ is a permutation matrix containing only ones and zeros such that operating on the vector $[x_1(n)\ \ x_2(n)\ \ \mathbf{X}^T(n-1)]^T$ with $\mathbf{S}$ rearranges the elements of the input vector to bring the elements to be discarded from $\mathbf{X}(n-1)$ at time n to the bottom two rows. Show that a similar permutation of the augmented gain vector and performing the remaining computations using this rearranged vector results in the desired fast RLS adaptive filter.

11.16. *A Fast RLS Adaptive IIR Filter:* Consider the problem of identifying an IIR system of the form

$$y(n) = \sum_{i=1}^{N} a_i y(n-i) + \sum_{j=0}^{M} b_j x(n-j)$$

using the input signal $x(n)$, its past samples and the past samples of the measured values of the output signal. The measured signal is modeled as

$$d(n) = y(n) + \eta(n),$$

where $\eta(n)$ is an additive noise sequence. Thus, the estimate of the output may be obtained as

$$\hat{y}(n) = \sum_{i=1}^{N} \hat{a}_i d(n-i) + \sum_{j=0}^{M} \hat{b}_j x(n-j).$$

a. Derive a fast exponentially-weighted RLS adaptive filter to estimate the parameters of the system model.

b. We wish to develop an adaptive filter that employs the model

$$\hat{y}(n) = \sum_{i=1}^{N} \hat{a}_i \hat{y}(n-i) + \sum_{j=0}^{M} \hat{b}_j x(n-j).$$

Suppose that we developed an adaptive filter by substituting $\hat{y}(n-i)$ at the appropriate locations of the system you developed in part a. Would this be an exact RLS solution? Why or why not? Describe the possible advantages and disadvantages of the algorithms of part a and b.

11.17. *Computing Assignment: Evaluation of the Fast Transversal Filter.* Develop a computer program to implement the fast transversal filter given in Table 11.6. Incorporate the rescue procedure into this program. Using the input signal and desired response models of Example 11.1, evaluate the performance of the FTF as a function of λ, δ and the eigenvalue spread of the autocorrelation matrix of the input signal when the adaptive filter is implemented with two coefficients and the maximum precision available in your computing system. Experiment with $\epsilon = 1.0, \ 0.1, \ 0.01$ and 0.001, $\delta = 0.001, \ 0.01, \ 0.1$ and 1.0, and $\lambda = 0.9, \ 0.95, \ 0.98, \ 0.99$ and 0.999. What inferences are obtained from the experiments regarding the relationship of these parameters to the numerical robustness (or its lack!) of the FTF? How does the rescue device affect the performance of the system.

11.18. *Computing Assignment: Evaluation of the Stabilized Fast Transversal Filter.* Modify the MATLAB program for implementing the stabilized FTF to truncate the output of each step to a desired level of precision. A simple MATLAB function that rounds its input to achieve a precision of $k2 = 2^K$ fractional bits may be written as follows:

```
function [y] = round1(x,k2)
y = round(x*k2)/k2;
```

The stabilized FTF program may be modified by applying the output of each step to the above function. The fractional part of the computations will then be rounded to K bits. This modifications assume that the internal calculations are performed with the maximum precision available in the computer. For our evaluations such an approach simplifies the software implementation without significantly altering the validity of the results. For the purpose of this assignment, we assume that the system is numerically stable if the adaptive filter operates without divergence for 10,000 iterations. Use the same input signal and desired response models as in Example 11.1 in this assignment.

a. For each value of ϵ in the set [1, 0.1, 0.01, 0.001], and $\delta = 1$, determine the minimum number of bits of precision required in the computations to provide numerically stable operation of the stabilized FTF for each λ in the set [0.999, 0.99, 0.98, 0.96, 0.95] and $L = 5$.

b. For $\epsilon = 0.1$ and 16-bit precision, determine the minimum value of λ that enables stable operation of the system for each value of L in the set [5,10,20,40].

c. For $L = 10$, $\epsilon = 0.1$ and 0.01, $\lambda = 0.99$ and 16-bit precision, determine the minimum value of δ for which the adaptive filter operates in a stable manner.