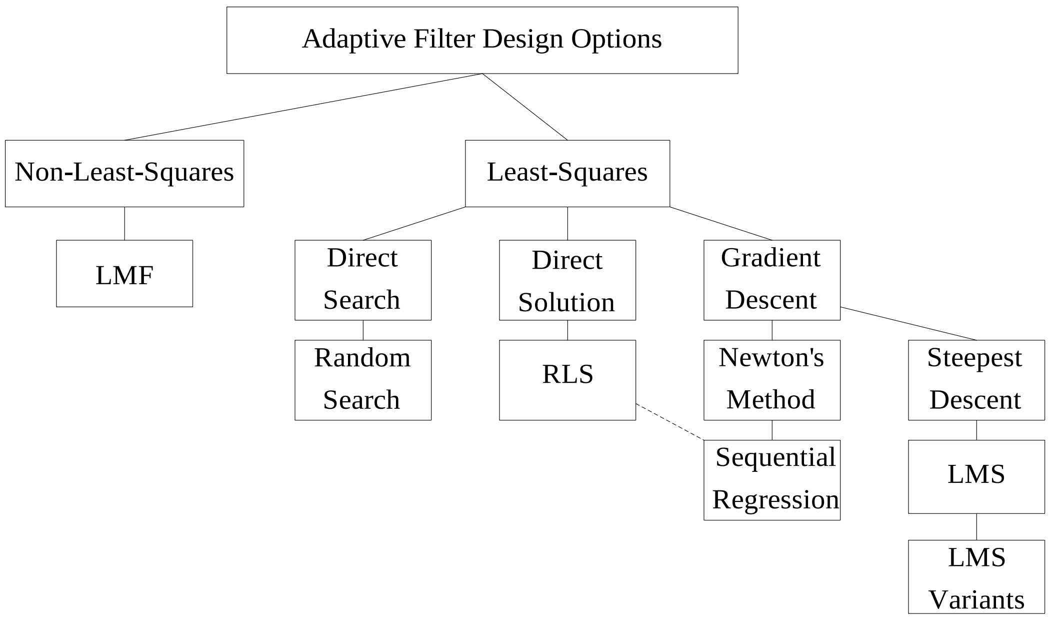
Chapter 5 Adaptive Signal Processing: Algorithms and Structures

I. General Remarks

1) So far, we have focused on just
   (a) One adaptive algorithm — LMS
   (b) One particular criterion — least-squares
   (c) One iterative philosophy — steepest descent
   (d) One particular implementation structure — tapped delay line (TDL)

   In this chapter, we shall try to indicate the principle directions in which adaptive algorithms can be developed.

2) Overview of Adaptive Algorithm Development:



■ **Figure 5.1** Options for adaptive algorithm design [1, p. 276].

3) Least Mean Fourth (LMF)
   (a) Adaptive structures are derived from minimization of a class of functions of the form:

$$J^{(N)} = E\left\{e^{2N}(n)\right\}, \quad N \geq 1 \tag{1}$$

where N is an integer constant, and where as usual

$$e(n) = d(n) - \mathbf{f}_n^T \mathbf{x}_n \tag{2}$$

Since $J^{(N)}$ is a convex function, it has no local minima and the gradient based adaptation scheme can be used without fear of convergence to a local minimum. We have the gradient

$$\nabla\left(e^{2N}(n)\right) = -2Ne^{2N-1}(n)\mathbf{x}_n \tag{3}$$

Hence, using the steepest descent update

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{\alpha}{2}\nabla\left(e^{2N}(n)\right) = \mathbf{f}_n + \alpha N e^{2N-1}(n)\mathbf{x}_n \tag{4}$$

When $N = 1$, the algorithm reduces to the conventional LMS.

(b) LMF obtained with $L > 1$ may outperform LMS for some data distributions. On the other hand, when the noise is Gaussian, the LMS is the best choice [2].

II. LMS Variants

1) These algorithms include a wide range of possibilities but have the common feature that they start from the LMS algorithm, and attempt to improve on this by incorporating some modification, usually designed on an ad-hoc basis, to achieve some specified objective. Possible objectives include:
   (a) improved convergence
   (b) reduced steady-state mean-squared error (MSE)
   (c) reduced computational requirement
   (d) other application specific objectives

2) Normalized LMS
   From Chapter 4, we have seen that the stability, convergence, and steady-state properties of the LMS algorithm are all directly influenced by the length L of the filter and the power of the input signal $x(n)$.
   Modification — normalize the adaptation constant with respect to these factors
   (a) We seek to replace the original step size α by

$$\alpha' = \frac{\alpha}{L \times E\left\{x^2(n)\right\}} = \frac{\alpha}{T_p} \tag{5}$$

In practice, the power T_P is estimated using a time average of the measured data. A convenient estimate is provided by

$$\hat{T}_P = \sum_{j=0}^{L-1} x^2(n-j) = \mathbf{x}_n^T \mathbf{x}_n \tag{6}$$

For a stationary and ergodic input $x(n)$, $\hat{T}_P$ is an unbiased and consistent estimation for T_P. Incorporation of this simple normalization into the LMS update gives the modified form

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \frac{e(n)\mathbf{x}_n}{\mathbf{x}_n^T \mathbf{x}_n} \tag{7}$$

This modified algorithm, which is generally referred to as the Normalized LMS (NLMS) algorithm.

Expense:

(i) computational overhead

(ii) possible division by zero

The probability of such zero division can be reduced by simply extending the data length over which the power estimate is made. Another approach is to add a small constant to the divisor in (7). Thus the update becomes

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \frac{e(n)\mathbf{x}_n}{c + \mathbf{x}_n^T \mathbf{x}_n} \tag{8}$$

where c is a small positive constant.

(b) For the homogeneous problem $d(n) = \mathbf{f}_n^{*T}\mathbf{x}_n$, with this normalized algorithm, it can be shown that provided $0 < \alpha < 2$: (see Appendix)

(i) $\lim_{n\to\infty} e(n) \to 0$.

(ii) $\mathbf{v}_n = \mathbf{f}_n - \mathbf{f}^*$ converges to a (finite) constant as $n \to \infty$.

We note that these results are essentially independent of the input data $x(n)$. In the more general problem where homogeneity is not assumed, we may show that the necessary and sufficient condition for the NLMS algorithm to be convergent in the mean is $0 < \alpha < 2$ as follows [3]:

In the LMS case, the step size must satisfy

$$0 < \alpha < \frac{2}{tr(\mathbf{R})} \tag{9}$$

to guarantee stability. The NLMS condition is

$$0 < \alpha < \frac{2}{tr\left(E\{\frac{\mathbf{x}_n\mathbf{x}_n^T}{\mathbf{x}_n^T\mathbf{x}_n}\}\right)} \tag{10}$$

Make the following approximation

$$E\left\{\frac{\mathbf{x}_n\mathbf{x}_n^T}{\mathbf{x}_n^T\mathbf{x}_n}\right\} \approx \frac{E\{\mathbf{x}_n\mathbf{x}_n^T\}}{E\left\{\|\mathbf{x}_n\|^2\right\}}. \tag{11}$$

Then

$$tr\left(E\left\{\frac{\mathbf{x}_n\mathbf{x}_n^T}{\mathbf{x}_n^T\mathbf{x}_n}\right\}\right) \approx \frac{tr\left(E\{\mathbf{x}_n\mathbf{x}_n^T\}\right)}{E\left\{\|\mathbf{x}_n\|^2\right\}} = \frac{E\left\{tr\left(\mathbf{x}_n\mathbf{x}_n^T\right)\right\}}{E\left\{\|\mathbf{x}_n\|^2\right\}} = \frac{E\left\{tr\left(\mathbf{x}_n^T\mathbf{x}_n\right)\right\}}{E\left\{\|\mathbf{x}_n\|^2\right\}} = 1 \tag{12}$$

3) LMS algorithm with a variable adaptation rate

(a) This algorithm is obtained by replacing the constant α by a time or data-dependent step-size $\alpha(n)$.

(b) The objective of this modification is to improve the initial convergence of the algorithm by using a high adaptation rate. Once the algorithm has converged, a smaller value for α is employed to produce a lower steady-state MSE.

(c) For *stationary* data, $\alpha(n)$ is typically chosen to reduce in magnitude with time. For example,

$$\alpha(n) = \frac{1}{n+c} \tag{13}$$

where c is a positive constant. With this policy, $\alpha(n)$ reduces monotonically as (hopefully) the optimum solution is approached.

Such solutions have *apparently* obvious advantages for stationary data, though as we shall see, there may not be any advantage even when the data is stationary.

For *nonstationary* data, such a formula would clearly be inappropriate since the reduction in steady-state error would be accompanied by inability of the algorithm to respond to changes in optimum solution $\mathbf{f}^*$.

(d) A natural extension of this approach is to develop some methods of control whereby the adaptation can be

(i) decreased in steady-state, and

(ii) increased to respond to possible nonstationarity

(e) variable step (VS) algorithm

The updated equation is

$$\mathbf{f}_{n+1} = \mathbf{f}_n + e(n)\mathbf{M}_n\mathbf{x}_n \tag{14}$$

where

$$\mathbf{M}_n = \begin{bmatrix} \alpha_0(n) & 0 & \cdots & 0 \\ 0 & \alpha_1(n) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{L-1}(n) \end{bmatrix}_{L\times L} \tag{15}$$

This represents a generalization of LMS in two ways:

(i) Each coefficient of the filter is updated using an independent adaptation coefficient.

(ii) Constant adaptation is replaced by a time-variable step size.

Hence, the ith coefficient is updated using $\alpha_i(n)$.

(f) The step-sizes are determined according to an ad-hoc strategy designed to force $\alpha_i(n)$ to

(i) decrease as the error approaches steady-state, and

(ii) increase as the solution moves away from the region of the optimum solution.

This is achieved by monitoring sign changes in the instantaneous gradient estimate $-2e(n)x(n-i)$. That is,

- if m_1 successive sign changes are observed in $e(n)x(n-i)$ (i.e., $\mathbf{f}_n$ is close to $\mathbf{f}^*$.), then $\alpha_i(n)$ is decreased by a factor c_1.
- if $e(n)x(n-i)$ has the same sign on m_2 successive updates (i.e., $\mathbf{f}_n$ is distant from $\mathbf{f}^*$.), then $\alpha_i(n)$ is increased by a factor c_2.

Successive changes in the sign of the gradient estimate indicate that the algorithm is close to the optimum solution (for which $\nabla J = 0$), and conversely when no sign changes occur, the filter coefficient vector is distant from $\mathbf{f}^*$.

Note:

1. Imposing hard limits on $\alpha_{\max}$ and $\alpha_{\min}$ in the procedure is enough to guarantee stability. m_1 and m_2 are parameters which can be adjusted to optimize performance, as can the factors c_1 and c_2.
2. This variable-step LMS algorithm can achieve considerable improvements in convergence rate for only a modest increase in computation [4].

(g) As it turns out, even in the stationary case, there are certainly some applications where time or data modification of α offers no advantages. In particular, when the steady-state error represents a high proportion of the total energy, as for example in the line enhancement application, then no advantage accrues. The problem arises because the adaptation mechanism must allow the filter to 'forget' old data as well as to acquire new data.

(i) The point is that if the filter is operated at an initially high adaptation rate, then the error in the filter coefficients due to the noise in the input will also be high.

(ii) If α subsequently falls too quickly, the filter may contain erroneous components which it is unable to forget. The filter has acquired a set of bad habits which it is subsequently stuck with!

4) The Leaky LMS algorithm

(a) The leaky LMS algorithm employs the LMS update modified by the presence of a constant 'leakage' factor γ

$$\mathbf{f}_{n+1} = \gamma \mathbf{f}_n + \alpha e(n) \mathbf{x}_n \tag{16}$$

where $0 < \gamma < 1$, but γ is typically close to one.

(b) Leakage allows the impact on the filter coefficient vector of any single input sample to decay with time. If the gradient estimates are consistently zero, then $\mathbf{f}_n$ decays to zero. That is, when $e(n)$ or the input $x(n)$ becomes zero (update procedure is failed.), the leakage coefficient is to force the filter coefficients to zero, i.e., reset [5].

(c) When the adaptive filter has modes associated with zero eigenvalues which do not converge, then (recall page 8 in Chapter 4 lecture)

$$u_n'(j) = (1 - \alpha \cdot 0)^n u_0'(j) = u_0'(j) \tag{17}$$

which does not decay to zero with n.

Since it's possible for these undamped modes to become unstable, it's important to stabilize the LMS adaptive filter by forcing these modes to zero.

(d) Performance comparison:

$$\begin{aligned}\mathbf{f}_{n+1} &= \gamma\mathbf{f}_n + \alpha e(n)\mathbf{x}_n = \gamma\mathbf{f}_n + \alpha\left[d(n) - \mathbf{x}_n^T\mathbf{f}_n\right]\mathbf{x}_n \\ &= \left(\gamma\mathbf{I} - \alpha\mathbf{x}_n\mathbf{x}_n^T\right)\mathbf{f}_n + \alpha d(n)\mathbf{x}_n\end{aligned} \tag{18}$$

Assuming independence of filter and data vectors, we have

$$E\{\mathbf{f}_{n+1}\} = (\gamma\mathbf{I} - \alpha\mathbf{R})E\{\mathbf{f}_n\} + \alpha\mathbf{g} \tag{19}$$

or

$$\begin{aligned}E\{\mathbf{f}_{n+1}\} &= (\mathbf{I} - \mathbf{I} + \gamma\mathbf{I} - \alpha\mathbf{R})E\{\mathbf{f}_n\} + \alpha\mathbf{g} \\ &= \left[\mathbf{I} - \alpha\left(\mathbf{R} + \frac{1-\gamma}{\alpha}\mathbf{I}\right)\right]E\{\mathbf{f}_n\} + \alpha\mathbf{g}\end{aligned} \tag{20}$$

From this, it can be shown that if the algorithm is stable, then

$$\lim_{n\to\infty} E\{\mathbf{f}_{n+1}\} \to \left(\mathbf{R} + \frac{1-\gamma}{\alpha}\mathbf{I}\right)^{-1}\mathbf{g} \tag{21}$$

which is clearly biased from $\mathbf{R}^{-1}\mathbf{g}$. Thus, the use of the leakage can be thought of as a compromise between bias and coefficient protection.

(e) If the eigenvalues of **R** are λ_i for i = 1, 2, …, L, then $\mathbf{R} + \frac{1-\gamma}{\alpha}\mathbf{I}$ has eigenvalues $\lambda_i + \frac{1-\gamma}{\alpha}$ for i = 1, 2, …, L. Thus, all of the eigenvalues of the transition matrix, $[\mathbf{I} - \alpha(\mathbf{R} + \frac{1-\gamma}{\alpha}\mathbf{I})]$, for the leaky algorithm are increased by a constant amount compared to LMS. This has three consequences:

(i) The stability requirement (in the mean) is

$$0 < \alpha < \frac{2}{\lambda_{\max} + (1-\gamma)/\alpha} \tag{22}$$

for $0 < \gamma < 1$ where $\lambda_{\max}$ is the largest eigenvalue of **R**.

$\Rightarrow$ It complicates the determination of stability limits.

(ii) $\mathbf{R} + \frac{1-\gamma}{\alpha}\mathbf{I}$ is strictly positive definite (for $\alpha > 0$, $0 < \gamma < 1$) and thus has no zero eigenvalues.

(iii) The time-constants for the algorithm are

$$t_i^{(L)} = \frac{1}{\alpha\lambda_i + (1-\gamma)} \le t_i \tag{23}$$

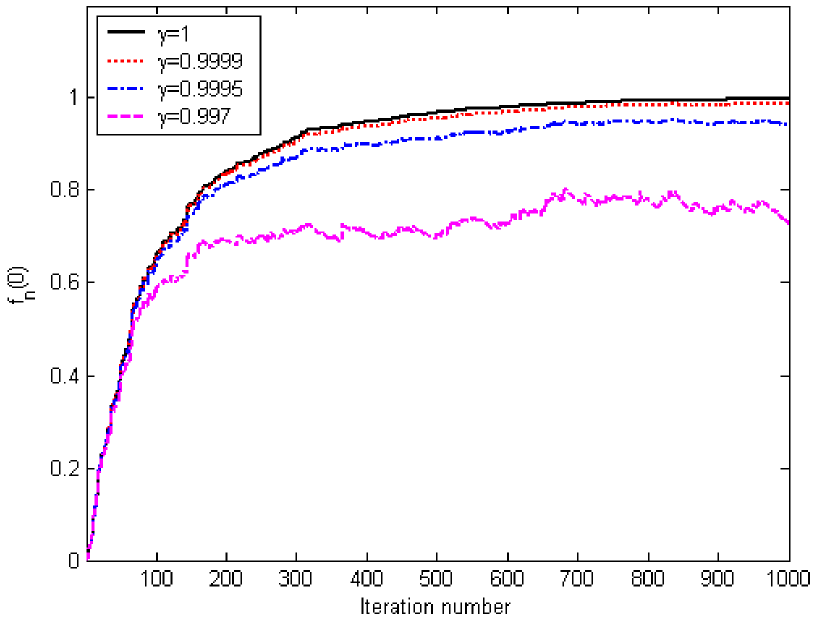
where $t_i^{(L)}$ is the time-constant for the ith mode of the leaky algorithm and t_i is the corresponding constant for the conventional LMS.

Example 1: Leaky LMS vs. conventional LMS

The bias of the leaky algorithm is illustrated numerically in Fig. 5.2, which shows the filter coefficient $f_n(0)$, for n = 0, 1, …, 999, for the leaky and regular LMS algorithms. In this experiment, the signals were

$$d(n) = x(n) + 0.5x(n-1)$$

where $x(n)$ is zero-mean, unit variance iid Gaussian. The filter length is 10 and $\alpha = 0.01$. Leakage factors are 1 (LMS), 0.999, 0.9995, and 0.997. Note that, given $\mathbf{R} = \sigma^2\mathbf{I}$ with $\sigma^2 = 1$, then from (21) for $f_n(0)$ the leaky algorithm should converge in the mean to 1, 1/1.01, 1/1.05, and 1/1.3, respectively.



■ **Figure 5.2** Leaky LMS trial. Plot shows $f_n(0)$ vs. n.

Note:

1. Because of the dependence on α of the bias, apparently small leakage factors can cause significant bias.
2. As γ is reduced, we also observe increasing fluctuations about the steady-state value. This is due to the loss of homogeneity in the problem when $\mathbf{f}_\infty$ is biased from $\mathbf{f}^*$.

(f) The leaky LMS algorithm can be derived directly by consideration of the modified functional

$$J' = e^2(n) + a\mathbf{f}_n^T\mathbf{f}_n \tag{24}$$

where a is a constant. Minimization of J' leads to the leaky update

$$\mathbf{f}_{n+1} = (1-\alpha a)\mathbf{f}_n + \alpha e(n)\mathbf{x}_n \tag{25}$$

Re-defining $\gamma = 1-\alpha a$ confirms the equivalence of this scheme and that of (16).

5) Sign algorithms

In spite of the relative computational simplicity of the LMS algorithm, there are some applications, particularly in high speed communications, where even this computational load is too high. An approach designed to reduce computation and simplify hardware requirements is provided by the sign algorithms.

(a) Three sign based LMS algorithms have been proposed. These quantize data, error or both in the LMS update to a single level representing the sign of the quantity only. The algorithms are as follows:

(i) The Pilot LMS, or Signed Error (SE), or simply Sign Algorithm (SA)

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \operatorname{sgn}\left(e(n)\right)\mathbf{x}_n \tag{26}$$

(ii) The Clipped LMS, or Signed Regressor (SR)

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha e(n) \operatorname{sgn}\left(\mathbf{x}_n\right) \tag{27}$$

(iii) The Zero-Forcing LMS, or Sign-Sign (SS)

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \operatorname{sgn}\left(e(n)\right) \operatorname{sgn}\left(\mathbf{x}_n\right) \tag{28}$$

In each case, the 'sign' operation is defined for a scalar variable *r* by

$$\operatorname{sgn}(r) = \begin{cases} 1\ , & r > 0 \\ 0\ , & r = 0 \\ -1, & r < 0 \end{cases} \tag{29}$$

and the operation is defined component by component for vector quantities.

(b) The advantage of the use of sign representations is that multiplications in the update are reduced to single bit operations. That is, the filtering requirement is unchanged in the sign algorithms, but the update is reduced to simple shifting and addition. On the other hand, compared to LMS, the update mechanism is degraded by the crude quantization of the gradient estimates, and this is manifested as either decreased convergence rate or increased steady-state error.

(c) The SE only modifies the size of each iterative step, but the SR also changes the direction of the update (in the space of filter coefficients). This is reflected in less predictable behavior for the SR algorithm.

(d) Some sequences exist which produce stable results for LMS but are unstable for the SR algorithm. This instability has nothing to do with the size of the adaptation constant α. It's caused by a change of direction produced by replacing $\mathbf{x}_n$ with $\operatorname{sgn}(\mathbf{x}_n)$.

(e) SE can be derived directly as an LMS type update using the modified functional

$$J = E\left\{\left|e(n)\right|\right\} \tag{30}$$

(f) The SS algorithm is the simplest of all, and for that reason the most widely used (e.g. CCITT ADPCM standard). The update is very crude in this case, and the algorithm is consequently the least robust of all.

6) Linear smoothing of the LMS gradient estimates

(a) An obvious drawback of the LMS algorithm is the use of a noisy instantaneous gradient estimate in the update equation.

Improvement — smoothing this estimate by averaging over several consecutive instantaneous values.

Such an average may be constructed using a moving window of the N most recent gradient estimates.

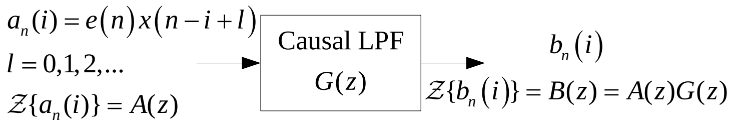
(b) The update equation for this Averaged LMS (ALMS) algorithm has the form

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \frac{\alpha}{N}\sum\nolimits_{j=n-N+1}^{n} e(j)\mathbf{x}_j \tag{31}$$

(c) General low-pass filter operation

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \mathbf{b}_n \tag{32}$$

where $\mathbf{b}_n = \begin{bmatrix} b_n(0) & b_n(1) & \cdots & b_n(L-1) \end{bmatrix}^T$ and its element is defined by $b_n(i) = LPF\left\{e(n)x(n-i), e(n-1)x(n-i-1), e(n-2)x(n-i-2), \ldots\right\}$, where $i = 0,1,2,\ldots,L-1$ and LPF represents a lowpass filter which is restricted to causal form.


$a_n(i) = e(n)x(n-i+l)$
$l = 0,1,2,\ldots$
$\mathcal{Z}\{a_n(i)\} = A(z)$
Causal LPF
$G(z)$
$b_n(i)$
$\mathcal{Z}\{b_n(i)\} = B(z) = A(z)G(z)$

■ **Figure 5.3** Gradient estimates smooth by using LPF.

The ALMS algorithm is one example of this class in which the LPF has impulse response $\left\{\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}\right\}$. There is, however, no particular reason to restrict attention to filters with uniform weighting, or even to FIR structures. For example, consider the 1st-order recursive equation

$$\begin{gathered} \mathbf{b}_n = \mathbf{b}_{n-1} + \gamma\left\{e(n)\mathbf{x}_n - \mathbf{b}_{n-1}\right\} \\ (B(z) = B(z)z^{-1} + \gamma\{A(z) - B(z)z^{-1}\}) \end{gathered} \tag{33}$$

where $A(z) = \mathcal{Z}\{a_n(i)\}$, $B(z) = \mathcal{Z}\{b_n(i)\}$, $G(z) = B(z)/A(z)$, and γ is a constant with $0 < \gamma < 1$. Here, each element of $e(n)\mathbf{x}_n$ is filtered using a transfer function of the form

$$G(z) = \frac{B(z)}{A(z)} = \frac{\gamma}{1-(1-\gamma)z^{-1}} \tag{34}$$

The LMS update has an effective feedback loop of

$$\frac{\alpha}{1-z^{-1}}$$

$$\left(\begin{array}{l} \because \mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \mathbf{b}(n) \\ F(z) = F(z)z^{-1} + \alpha z^{-1}B(z) \Rightarrow \dfrac{F(z)}{B(z)} = \dfrac{\alpha z^{-1}}{1-z^{-1}} \end{array}\right) \tag{35}$$

Combining (34) and (35), we have the overall update equation

$$\mathbf{f}_{n+1} = \mathbf{f}_n + (1-\gamma)(\mathbf{f}_n - \mathbf{f}_{n-1}) + \alpha\gamma e(n)\mathbf{x}_n$$

$$\left(\frac{F(z)}{B(z)}\frac{B(z)}{A(z)} = \frac{\alpha z^{-1}}{1-z^{-1}} \cdot \frac{\gamma}{1-(1-\gamma)z^{-1}} = \frac{\alpha\gamma z^{-1}}{1-(1-\gamma)(z^{-1}-z^{-2})-z^{-1}}\right) \tag{36}$$

This algorithm is also known as the Momentum LMS.
It was shown, in the literature, that reducing misadjustment with such algorithms also reduces the convergence rate.

7) Nonlinear smoothing of the LMS gradient estimates

(a) There are certainly some situations where some form of smoothing can greatly improve the performance of the LMS algorithm. For example, the presence of impulsive interference in either *d*(*n*) or *x*(*n*) generally has a degrading effect on the filter, and in extreme cases can lead to instability.

(b) One solution to this problem subjects the gradient estimate to a nonlinear smoothing using the sample median of the *N* most recent instantaneous gradient estimates.

(i) given a sequence of *N* signal samples $s_1, s_2, \ldots, s_N$

(ii) rank the samples as $s_{(1)}, s_{(2)}, \ldots, s_{(N)}$, with $s_{(1)} \le s_{(2)} \le \ldots \le s_{(N)}$

(iii) The median is then $s_{med} = Med\{s_i\}_N = s_{(N+1)/2}$.

where, for convenience, we have assumed *N* is odd so that the middle point is an integer index.

(c) Median filters are widely used in signal and image processing. Such filters are not linear and hence cannot be characterized in terms of a simple transfer function. They are characterized by two key properties:

(i) Median filters completely remove short duration impulses from a signal.

(ii) Median filters pass ‘edges’ without smearing.

Both of these properties contrast sharply with linear filters which generally smear edges and can only partially attenuate impulses.

(d) A Median LMS (MLMS) Algorithm is defined by applying the median operation to the elements of the most recent *N* gradient estimates in the LMS algorithm. That is, the MLMS algorithm is characterized by an update of the form

$$f_{n+1}(i) = f_n(i) + \alpha Med\{e(n)x(n-i)\}_N, \quad i = 0,1,\ldots,L-1 \tag{37}$$

where the operation $Med\{e(n)x(n-i)\}_N$ denotes the application of the median to the N most recent gradient estimates:

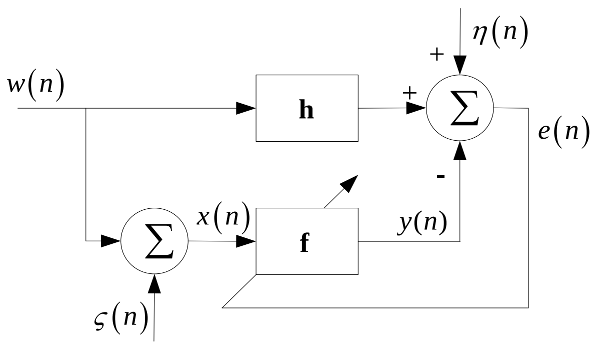
$$\begin{aligned} &Med\{e(n)x(n-i)\}_N \\ &= median\{e(n)x(n-i),\ldots,e(n-N+1)x(n-i-N+1)\} \end{aligned} \tag{38}$$

Example 2: Identification

Consider the desired and input signals: (see Fig. 5.4)

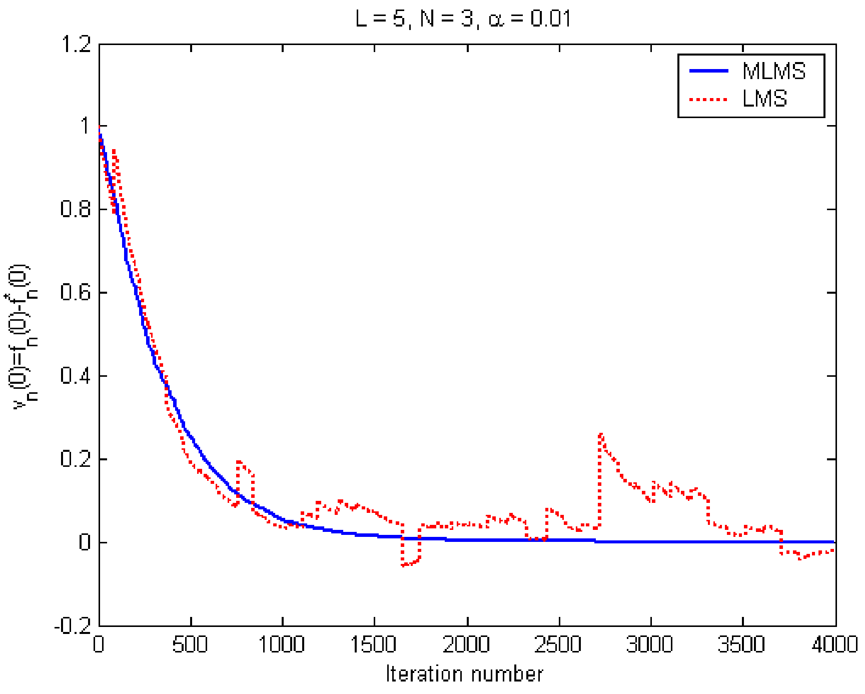
$$d(n) = \mathbf{h}^T\mathbf{w}_n + \eta(n)$$
$$x(n) = w(n) + \varsigma(n)$$

where $w(n)$ is a zero-mean iid sequence with samples uniformly distributed on [−1,1]; $\varsigma(n)$ and $\eta(n)$ are 'sparse' sequences with nonzero values (impulses) occurring with a mean intervals of 100 samples. The impulse amplitudes are Gaussian distributed with zero mean and standard deviation equal to 8, $\mathbf{h} = \begin{bmatrix} h(0) & \cdots & h(L-1) \end{bmatrix}^T$ is an LTI system, and $\mathbf{w}_n = \begin{bmatrix} w(n) & \cdots & w(n-L+1) \end{bmatrix}^T$.

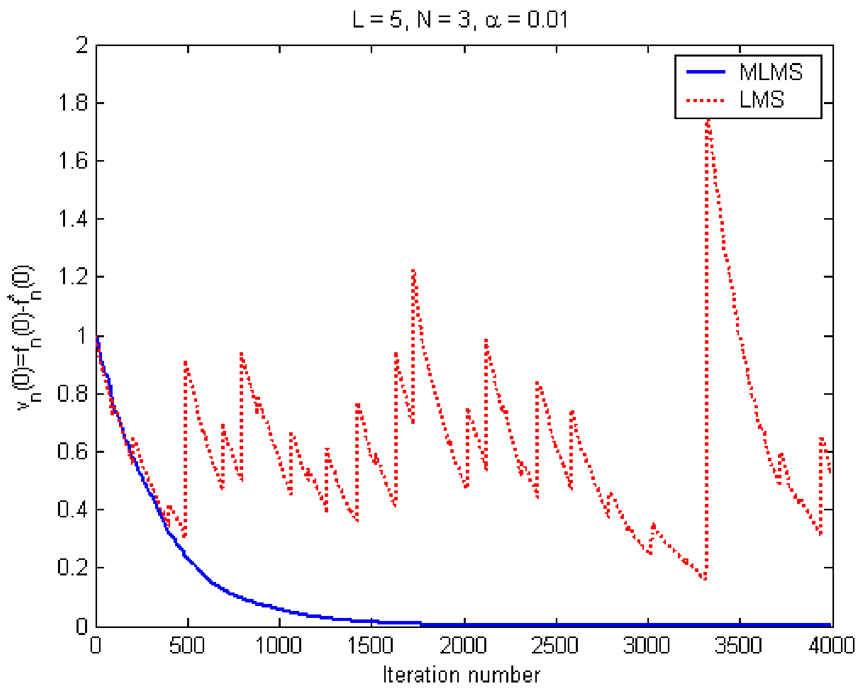


■ **Figure 5.4** Adaptive system with adaptive impulsive noises applied to input and desired signals.

We will examine the behavior of the adaptive filter for two cases: (1) $\varsigma(n) \equiv 0$ and (2) $\eta(n) \equiv 0$, in order to separately examine the influence of input and output impulsive disturbance. Assume $\mathbf{f}_0 = \mathbf{0}$ and $\mathbf{h} = \begin{bmatrix} -1 & 0 & \cdots & 0 \end{bmatrix}^T$.



■ **Figure 5.5** Adaptive system modeling with impulsive noise in the system output.



■ **Figure 5.6** Adaptive system modeling with impulsive noise in the system input.

(i) Even though MLMS has obvious advantages in such environments, we should note that this gain is offset by slightly slower convergence when the inputs are nonimpulsive.

(ii) While the performance of the algorithm has been convincingly demonstrated in many practical situations, analytic performance predictions have only proved possible for very restrictive inputs, and subject to a number of simplifying assumptions.

8) Double updating

The basic update is provided by the simple NLMS equation:

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \frac{e(n)\mathbf{x}_n}{\mathbf{x}_n^T \mathbf{x}_n} \tag{39}$$

where, as usual, $e(n) = d(n) - y(n)$ and $y(n) = \mathbf{f}_n^T \mathbf{x}_n$.

The difference between this procedure and the LMS is that in this algorithm, the updated coefficients $\mathbf{f}_{n+1}$ are applied to the same data, which is to the data at time *n*. This produces

$$\hat{y}(n) = \mathbf{f}_{n+1}^T \mathbf{x}_n \tag{40}$$

and $\hat{y}(n)$ represents the final output at time *n*. If we consider the application of the algorithm with $\alpha = 1$, the final output $\hat{y}(n)$ is then given by

$$\hat{y}(n) = \mathbf{f}_{n+1}^T \mathbf{x}_n = \mathbf{f}_n^T \mathbf{x}_n + \frac{e(n)\mathbf{x}_n^T \mathbf{x}_n}{\mathbf{x}_n^T \mathbf{x}_n} \tag{41}$$

where the second equality was obtained by substituting NLMS updated equation into $\hat{y}(n) = \mathbf{f}_{n+1}^T \mathbf{x}_n$. Hence, we have

$$\hat{y}(n) = y(n) + \left[d(n) - y(n)\right] = d(n) \tag{42}$$

In words, the output of the filter is equal to the desired input, and the result is a perfect algorithm! Paradoxically this is generally not a desirable result. In fact, further consideration suggests that this algorithm does not in general solve the least-squares problem since it can always produce *J* equal to 0.

9) Complex LMS algorithm

This algorithm is useful for adaptive arrays and for frequency domain implementations. Consider a complex data vector

$$\mathbf{x}_n = \mathbf{x}_{n,R} + j\mathbf{x}_{n,I} \tag{43}$$

where $\mathbf{x}_{n,R}$ and $\mathbf{x}_{n,I}$ are the real and imaginary parts of the data vector and where

$$\mathbf{x}_{n,R} = \left[x_R(n) \quad x_R(n-1) \quad \cdots \quad x_R(n-L+1)\right]^T \tag{44}$$

with $\mathbf{x}_{n,I}$ defined similarly. The complex weight vector $\mathbf{f}_n$ can be similarly decomposed into real and imaginary components

$$\mathbf{f}_n = \mathbf{f}_{n,R} + j\mathbf{f}_{n,I} \tag{45}$$

Likewise, definitions of the complex output, desired signal and error may be given as

$$\begin{aligned} y(n) &= \mathbf{f}_n^T \mathbf{x}_n = y_R(n) + jy_I(n) \\ &= \left(\mathbf{f}_{n,R}^T \mathbf{x}_{n,R} - \mathbf{f}_{n,I}^T \mathbf{x}_{n,I}\right) + j\left(\mathbf{f}_{n,I}^T \mathbf{x}_{n,R} + \mathbf{f}_{n,R}^T \mathbf{x}_{n,I}\right) \end{aligned} \tag{46}$$

$$d(n) = d_R(n) + jd_I(n) \tag{47}$$

and

$$e(n) = e_R(n) + je_I(n) = d(n) - y(n) \tag{48}$$

For real signals, we minimize $J = E\{e^2(n)\}$. For complex signals, this is replaced by

$$J = E\left\{|e(n)|^2\right\} = E\{e(n)e^*(n)\} = E\left\{[e_R(n)]^2 + [e_I(n)]^2\right\} \tag{49}$$

where $(\cdot)^*$ denotes complex conjugate. (Care should be taken to avoid confusion with the use of $(\cdot)^*$ to denote the optimal filter coefficients.) From (46) and (47), *e*(*n*) has the form

$$e(n) = d_R(n) + jd_I(n) - \left[\left(\left(\mathbf{f}_{n,R}^T \mathbf{x}_{n,R} - \mathbf{f}_{n,I}^T \mathbf{x}_{n,I}\right) + j\left(\mathbf{f}_{n,I}^T \mathbf{x}_{n,R} + \mathbf{f}_{n,R}^T \mathbf{x}_{n,I}\right)\right)\right] \tag{50}$$

so that $|e(n)|^2$ is given by

$$\begin{aligned} |e(n)|^2 &= \left[d_R(n) - \left(\mathbf{f}_{n,R}^T \mathbf{x}_{n,R} - \mathbf{f}_{n,I}^T \mathbf{x}_{n,I}\right)\right]^2 + \left[d_I(n) - \left(\mathbf{f}_{n,I}^T \mathbf{x}_{n,R} + \mathbf{f}_{n,R}^T \mathbf{x}_{n,I}\right)\right]^2 \\ &= [e_R(n)]^2 + [e_I(n)]^2 \end{aligned} \tag{51}$$

As we saw in Chapter 4, the LMS algorithm is derived from the steepest descent equation

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{\alpha}{2} \cdot \frac{\partial \hat{J}}{\partial \mathbf{f}_n} \tag{52}$$

Hence, for the complex LMS, the algorithm is obtained from

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{\alpha}{2} \cdot \frac{\partial |e(n)|^2}{\partial \mathbf{f}_n} \tag{53}$$

or, splitting into real and imaginary components

$$\mathbf{f}_{n+1,R} = \mathbf{f}_{n,R} - \frac{\alpha}{2} \cdot \frac{\partial |e(n)|^2}{\partial \mathbf{f}_{n,R}} \quad \text{and} \quad \mathbf{f}_{n+1,I} = \mathbf{f}_{n,I} - \frac{\alpha}{2} \cdot \frac{\partial |e(n)|^2}{\partial \mathbf{f}_{n,I}} \tag{54}$$

In total,

$$\mathbf{f}_{n+1} = \left(\mathbf{f}_{n,R} + j\mathbf{f}_{n,I}\right) - \frac{\alpha}{2} \cdot \left[\frac{\partial |e(n)|^2}{\partial \mathbf{f}_{n,R}} + j\frac{\partial |e(n)|^2}{\partial \mathbf{f}_{n,I}}\right] \tag{55}$$

Now, from (51)

$$\begin{aligned}\frac{\partial\left|e(n)\right|^2}{\partial\mathbf{f}_{n,R}}&=-2e_R(n)\mathbf{x}_{n,R}-2e_I(n)\mathbf{x}_{n,I}\\ \frac{\partial\left|e(n)\right|^2}{\partial\mathbf{f}_{n,I}}&=2e_R(n)\mathbf{x}_{n,I}-2e_I(n)\mathbf{x}_{n,R}\end{aligned}\tag{56}$$

Substituting into $\mathbf{f}_{n+1}$ gives

$$\begin{aligned}\mathbf{f}_{n+1}&=\mathbf{f}_n+\alpha\left\{\left[e_R(n)\mathbf{x}_{n,R}+e_I(n)\mathbf{x}_{n,I}\right]-j\left[e_R(n)\mathbf{x}_{n,I}-e_I(n)\mathbf{x}_{n,R}\right]\right\}\\ &=\mathbf{f}_n+\alpha\left\{\left[e_R(n)+je_I(n)\right]\left[\mathbf{x}_{n,R}-j\mathbf{x}_{n,I}\right]\right\}\end{aligned}\tag{57}$$

Hence,

$$\mathbf{f}_{n+1}=\mathbf{f}_n+\alpha e(n)\mathbf{x}_n^*\tag{58}$$

This is the update for the complex LMS, which as we can see is identical to that for LMS with real inputs, except for the conjugation of the data vector.

10) The Block LMS algorithm

Another algorithm motivated by a desire to reduce the computational burden of LMS is the Block LMS (BLMS) algorithm.

(a) The BLMS is identical to the LMS algorithm except that in the BLMS the filter weights are updated only once every *N* data samples. That is, the filter coefficients are held constant over each block of *N* data points.

(b) The filter is updated using the average of the gradient estimates for that block. Thus, the BLMS update has the form:

$$\mathbf{f}_{(j+1)N}=\mathbf{f}_{jN}+\frac{\alpha_B}{N}\sum_{i=0}^{N-1}e(jN+i)\mathbf{x}_{jN+i},\quad \begin{array}{l}i=0,1,\ldots,N-1\\ j=0,1,\ldots\end{array}\tag{59}$$

where

$$\begin{aligned}y(jN+i)&=\mathbf{f}_{jN}^T\mathbf{x}_{jN+i}\\ e(jN+i)&=d(jN+i)-y(jN+i)=d(jN+i)-\mathbf{f}_{jN}^T\mathbf{x}_{jN+i}\end{aligned}\tag{60}$$

and $\mathbf{x}_{jN+i}=\left[x(jN+i)\quad x(jN+i-1)\quad\cdots\quad x(jN+i-L+1)\right]^T$.

Hence, $\mathbf{f}_{jN}$ is the filter vector corresponding to the *j*th block of *N* data points. α_B is the adaptation constant for the algorithm.

(c) The BLMS algorithm generally has a computational advantage compared to LMS. The BLMS algorithm requires *NL*+*L* multiplications per block for the updates (*L*: filter length; *N*: block size). This compares to *NL*+*N* per block of *N* points for the LMS. (Both algorithms have an equal number of additions.) For a direct implementation, both algorithms have an equal computational requirement for the convolution (*NL* multiplications per block). Thus, the total computation is less for BLMS whenever *L* < *N*.

(d) However, the major advantage for BLMS arises when efficient techniques for the implementation of block digital filters using parallel processing (or series processing with the FFT algorithm) are employed.

(e) Comparison between LMS and BLMS

(i) LMS

$$\begin{aligned}\mathbf{f}_{jN+1} &= \mathbf{f}_{jN} + \alpha e(jN)\mathbf{x}_{jN} \\ \mathbf{f}_{jN+2} &= \mathbf{f}_{jN+1} + \alpha e(jN+1)\mathbf{x}_{jN+1} \\ &= \mathbf{f}_{jN} + \alpha\left[e(jN)\mathbf{x}_{jN} + e(jN+1)\mathbf{x}_{jN+1}\right] \\ &\vdots \\ \mathbf{f}_{(j+1)N} &= \mathbf{f}_{jN} + \alpha\sum_{i=0}^{N-1} e(jN+i)\mathbf{x}_{jN+i}\end{aligned} \tag{61}$$

where in the LMS

$$e(jN+i) = d(jN+i) - \mathbf{f}_{jN+i}^T\mathbf{x}_{jN+i} \tag{62}$$

Substituting $e(jN+i)$ into $\mathbf{f}_{(j+1)N}$ gives

$$\mathbf{f}_{(j+1)N} = \mathbf{f}_{jN} + \alpha\sum_{i=0}^{N-1}\left[d(jN+i)\mathbf{x}_{jN+i} - \mathbf{x}_{jN+i}\mathbf{x}_{jN+i}^T\mathbf{f}_{jN+i}\right] \tag{63}$$

(ii) BLMS

$$\mathbf{f}_{(j+1)N} = \mathbf{f}_{jN} + \frac{\alpha_B}{N}\sum_{i=0}^{N-1}\left[d(jN+i)\mathbf{x}_{jN+i} - \mathbf{x}_{jN+i}\mathbf{x}_{jN+i}^T\mathbf{f}_{jN}\right] \tag{64}$$

The difference is that the BLMS updates are produced by the average of the gradient values over a block, obtained with a single (fixed) filter coefficient vector. In the LMS algorithm, the filter is updated at every data point, and the updated information is fed back into the gradient estimation process.

(f) In fact, the BLMS algorithm can be derived as a steepest descent algorithm with functional

$$J = \frac{1}{N}E\left\{\sum_{i=0}^{N-1} e^2(jN+i)\right\} \tag{65}$$

While the adaptive algorithm derived from this functional may produce BLMS, the direct solution for the fixed least-squares operator (assuming, of course, the $x(n)$ and $d(n)$ are jointly stationary) is the usual form

$$\mathbf{f}^* = \mathbf{R}^{-1}\mathbf{g} \tag{66}$$

where the definitions of **R** and **g** are unchanged from those of Chapter 3.

(g) Convergence properties

(i) Starting from the BLMS update (59) and error (60), we have

$$\begin{aligned}\mathbf{f}_{(j+1)N} &= \mathbf{f}_{jN} + \frac{\alpha_B}{N}\sum_{i=0}^{N-1}\left[d(jN+i) - \mathbf{f}_{jN}^T\mathbf{x}_{jN+i}\right]\mathbf{x}_{jN+i} \\ &= \left[\mathbf{I} - \frac{\alpha_B}{N}\sum_{i=0}^{N-1}\mathbf{x}_{jN+i}\mathbf{x}_{jN+i}^T\right]\mathbf{f}_{jN} + \frac{\alpha_B}{N}\sum_{i=0}^{N-1} d(jN+i)\mathbf{x}_{jN+i}\end{aligned} \tag{67}$$

Applying expectations to both sides of (67) and using the independence assumption, we have

$$\begin{aligned}E\left\{\mathbf{f}_{(j+1)N}\right\} &= \left(\mathbf{I} - \frac{\alpha_B}{N}N\mathbf{R}\right)E\left\{\mathbf{f}_{jN}\right\} + \frac{\alpha_B}{N}N\mathbf{g} \\ &= \left(\mathbf{I} - \alpha_B\mathbf{R}\right)E\left\{\mathbf{f}_{jN}\right\} + \alpha_B\mathbf{g}\end{aligned} \tag{68}$$

Define $\mathbf{u}_{jN} = E\left\{\mathbf{f}_{jN}\right\} - \mathbf{f}^*$, then we may write

$$\mathbf{u}_{(j+1)N} = \left(\mathbf{I} - \alpha_B\mathbf{R}\right)\mathbf{u}_{jN} \tag{69}$$

which is identical to the LMS mean error vector behavior with the exception that the updates are indexed *jN* and (*j*+1)*N*, respectively. (The analysis here is directly analogous to that of Chapter 4 for LMS.) Hence, we may apply the orthogonal decomposition $\mathbf{R} = \mathbf{Q\Lambda Q}^T$, and ultimately, as in the LMS case, we can conclude that

$$\lim_{n\to\infty} E\left\{\mathbf{f}_n\right\} \to \mathbf{f}^* \tag{70}$$

subject to

$$0 < \alpha_B < \frac{2}{\lambda_{\max}} \tag{71}$$

where, as usual, $\lambda_{\max}$ is the largest eigenvalue of **R**.

(ii) The BLMS time-constants are increased by a factor of *N* relative to LMS. That is, for LMS

$$t_i = \frac{1}{\alpha\lambda_i} \tag{72}$$

and for BLMS

$$t_{Bi} = \frac{N}{\alpha_B\lambda_i} \tag{73}$$

Thus, the speed of convergence of each mode is reduced by a factor of *N* compared to that of LMS. On the other hand, this reduction is compensated by a reduction in excess MSE due to the smoothing effect of the average operation.

(iii) The steady-state MSE for BLMS is given as

$$J = J_{\min}\left(1+\frac{\alpha_B}{2N}tr\{\mathbf{R}\}\right) \tag{74}$$

This compares with LMS as

$$J = J_{\min}\left(1+\frac{\alpha}{2}tr\{\mathbf{R}\}\right) \tag{75}$$

Thus, the excess MSE is reduced by a factor of N relative to LMS. Hence, subject to the stability considerations of $0<\alpha<\frac{2}{\lambda_{\max}}$, we may conclude that with stationary input data the choice $\alpha_B = N\alpha$ gives equivalent convergence rates (time-constants) and steady-state behavior for the BLMS and LMS algorithms.

11) LMS Volterra filter

(a) The well-known Volterra processor has been widely used in nonlinear system identification and filtering. While technological advances have increased available computational power, and thus improved the practicability of nonlinear processing schemes, most efforts have focused on second-order systems, for which the computation is most manageable.

(b) A digital second-order Volterra filter with sampled input signals $x(n)$ produces an output $y(n)$ according to the relation

$$y(n)=\sum_{j=0}^{L-1} f_{(1)}(j)x(n-j)+\sum_{j_1=0}^{L-1}\sum_{j_2=j_1}^{L-1} f_{(2)}(j_1,j_2)x(n-j_1)x(n-j_2) \tag{76}$$

where $f_{(1)}(j)$ are the usual linear filter coefficients, $f_{(2)}(j_1,j_2)$ are the $L(L+1)/2$ quadratic response coefficients.

The restriction of both linear and quadratic 'lengths' to L is for convenience, and where the limits j_1, j_2 have been chosen to avoid multiplicities in the quadratic components.

(c) As with the linear filter, the solution of MMSE may be obtained by differentiating $J=E\{e^2(n)\}$ with respect to the individual coefficients and equating the results to zero.

$$\frac{\partial J}{\partial f_{(1)}(j)} = 0 = E\{e(n)x(n-j)\},\ \ 0\le j\le L-1 \tag{77}$$

which expresses the orthogonality of the error and the data, as occurs with the linear filter. Differentiating with respect to the second order terms gives

$$\frac{\partial J}{\partial f_{(2)}(j_1,j_2)} = 0 = E\{e(n)x(n-j_1)x(n-j_2)\},\ \ 0\le j_1\le j_2\le L-1 \tag{78}$$

Equation (78) expresses the fact that for the quadratic MMSE filter, a second condition holds; *orthogonality of the error and the data correlation*.

(d) For zero-mean stationary inputs, the optimal filter is obtained from the solution of a set of normal equations which mirror the form of those for the linear filter:

$$\mathbf{R}\mathbf{f} = \mathbf{g} \tag{79}$$

but where for the second-order filter

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_{(1)}^T & \mathbf{f}_{(2)}^T \end{bmatrix}^T \tag{80}$$

with

$$\mathbf{f}_{(1)} = \begin{bmatrix} f_{(1)}(0) & \cdots & f_{(1)}(L-1) \end{bmatrix}_{L\times 1}^T \tag{81}$$

$$\begin{aligned}\mathbf{f}_{(2)} = \Big[& f_{(2)}(0,0) \quad f_{(2)}(0,1) \quad \cdots \quad f_{(2)}(0,L-1) \quad f_{(2)}(1,1) \\ & \cdots \quad f_{(2)}(L-1,L-1)\Big]_{L(L+1)/2\times 1}^T\end{aligned} \tag{82}$$

We also define an extended $\left[L+L(L+1)/2\right]\times 1$ data vector $\mathbf{x}_n$ as

$$\mathbf{x}_n = \begin{bmatrix} \mathbf{x}_{n,(1)}^T & \mathbf{x}_{n,(2)}^T \end{bmatrix}^T \tag{83}$$

where

$$\mathbf{x}_{n,(1)} = \begin{bmatrix} x(n) & x(n-1) & \cdots & x(n-L+1) \end{bmatrix}^T \tag{84}$$

$$\begin{aligned}\mathbf{x}_{n,(2)} = \Big[& x^2(n) \quad x(n)x(n-1) \quad \cdots \quad x(n)x(n-L+1) \quad x^2(n-1) \\ & \cdots \quad x^2(n-L+1)\Big]^T\end{aligned} \tag{85}$$

with these definitions, the filter output can be written in the usual form

$$y(n) = \mathbf{f}_n^T \mathbf{x}_n \tag{86}$$

In (79)**,** **R** is the extended correlation matrix

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_2 & \mathbf{R}_3 \\ \mathbf{R}_3^T & \mathbf{R}_4 \end{bmatrix} \tag{87}$$

where

$$\mathbf{R}_2 = E\left\{\mathbf{x}_{n,(1)}\mathbf{x}_{n,(1)}^T\right\}_{L\times L}, \quad \mathbf{R}_3 = E\left\{\mathbf{x}_{n,(1)}\mathbf{x}_{n,(2)}^T\right\}_{L\times[L(L+1)/2]},$$

$$\mathbf{R}_4 = E\left\{\mathbf{x}_{n,(2)}\mathbf{x}_{n,(2)}^T\right\}_{[L(L+1)/2]\times[L(L+1)/2]}. \tag{88}$$

g is the extended cross-correlation vector given by

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_2^T & \mathbf{g}_4^T \end{bmatrix}^T \tag{89}$$

where

$$\mathbf{g}_2 = E\left\{d(n)\mathbf{x}_{n,(1)}\right\}_{L\times 1} \quad \text{and} \quad \mathbf{g}_4 = E\left\{d(n)\mathbf{x}_{n,(2)}\right\}_{[L(L+1)/2]\times 1} \tag{90}$$

For the second-order case, we note that while **R** is symmetric it is not Toeplitz or block-Toeplitz. As in the linear case, **R** is positive semi-definite but not positive definite.

Note:

1. If the third-order moments are zero, the **R** reduces to the block-diagonal form

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_4 \end{bmatrix} \tag{91}$$

From this $\mathbf{R}_2\mathbf{f}_{(1)} = \mathbf{g}_2$ and $\mathbf{R}_4\mathbf{f}_{(2)} = \mathbf{g}_4$. That is, the linear and quadratic components decouple.
One particular case where this occurs is with Gaussian inputs. More generally, any linear process generated using an iid input with samples drawn from a symmetric density will satisfy this condition. We refer to such signals as *symmetric*.

2. If $d(n)$ and $x(n)$ are jointly Gaussian, then $\mathbf{g}_4 = \mathbf{0}$ and hence $\mathbf{f}_{(2)} = \mathbf{0}$ (provided $\mathbf{R}_4$ has full rank). This confirms the well-known result that for Gaussian signals, quadratic (or other nonlinear) terms offer no reduction in MMSE compared to that obtained by the linear least-squares filter.

(e) Adaptive second-order Volterra filter

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha e(n)\mathbf{x}_n \tag{92}$$

with

$$\begin{aligned} y(n) &= \mathbf{f}_n^T \mathbf{x}_n \\ e(n) &= d(n) - y(n) \end{aligned} \tag{93}$$

As the same in the LMS, $\lim_{n\to\infty} E\{\mathbf{f}_n\} \to \mathbf{f}^*$ provided $0 < \alpha < \dfrac{2}{\lambda_{\max}}$, where $\lambda_{\max}$ is the largest eigenvalue of **R**.

(i) In the case where $\mathbf{R}_3 = 0$, that is where $x(n)$ is symmetric, the expectation of the filter vector is

$$E\{\mathbf{f}_{n+1}\} = \left(\mathbf{I} - \alpha \begin{bmatrix} \mathbf{R}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_4 \end{bmatrix}\right) E\{\mathbf{f}_n\} + \alpha \mathbf{g} \tag{94}$$

That is, in expectation at any rate, the linear and quadratic components of the filter may be decoupled to yield

$$E\{\mathbf{f}_{n+1,(1)}\} = (\mathbf{I} - \alpha\mathbf{R}_2) E\{\mathbf{f}_{n,(1)}\} + \alpha\mathbf{g}_2 \tag{95}$$

$$E\{\mathbf{f}_{n+1,(2)}\} = (\mathbf{I} - \alpha\mathbf{R}_4) E\{\mathbf{f}_{n,(2)}\} + \alpha\mathbf{g}_4 \tag{96}$$

For this decoupled case, we may conclude that the linear and quadratic components of the filter converge independently (again in expectation). Furthermore, the stability and relative convergence rates of the linear and quadratic modes will be determined by the eigenvalues of $\mathbf{R}_2$ and $\mathbf{R}_4$, respectively.

(ii) When $x(n)$ is symmetric, it would seem expedient to use distinct adaptation constants, α_1 and α_2, for the linear and quadratic sections. If $\lambda_{\max 2}$ and $\lambda_{\max 4}$ are the maximum eigenvalues of $\mathbf{R}_2$ and $\mathbf{R}_4$, respectively, then stability in the mean is assured provided $0 < \alpha_1 < \dfrac{2}{\lambda_{\max 2}}$ for the linear updates, and $0 < \alpha_2 < \dfrac{2}{\lambda_{\max 4}}$ for the quadratic.

Note:

1. One feature that distinguishes the second-order Volterra LMS from the linear algorithm is that **R** is not diagonal even if $x(n)$ is iid. This in turn produces eigenvalue disparity, even with iid inputs.
2. Consequently, unlike the linear filter, the quadratic LMS generally undergoes nonuniform convergence of the individual modes even when the input is white. This nonuniform behavior includes both distinct convergence factors and cross-coupling which distributes error across the filter coefficients.

<u>Example</u> 3: $x(n) \sim N(0,\sigma^2)$ and $d(n) = x^2(n)$ with $L = 2$. Defining the mean error vector

$$\mathbf{u}_n = E\{\mathbf{f}_n\} - \mathbf{f}^*$$

then

$$\mathbf{u}_{n+1} = (\mathbf{I} - \alpha\mathbf{R})\mathbf{u}_n$$

with

$$\mathbf{R} = \left[\begin{array}{c|cccc} \sigma^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 & 0 \\ \hline 0 & 0 & 3\sigma^4 & 0 & \sigma^4 \\ 0 & 0 & 0 & \sigma^4 & \\ 0 & 0 & \sigma^4 & 0 & 3\sigma^4 \end{array}\right]$$

By the eigen-decomposition, $\mathbf{R} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$,

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix}, \ \boldsymbol{\Lambda} = diag\left(\sigma^2, \sigma^2, 2\sigma^4, \sigma^4, 4\sigma^4\right)$$

If we assume that initially $\mathbf{f}_0 = \mathbf{0}$, then $\mathbf{u}_0 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \end{bmatrix}^T$,

$$\mathbf{u}_n = \mathbf{Q}\mathbf{u}'_n = \begin{bmatrix} u'_n(1) \\ u'_n(2) \\ \frac{1}{\sqrt{2}}\left[u'_n(3)+u'_n(5)\right] \\ u'_n(4) \\ \frac{1}{\sqrt{2}}\left[-u'_n(3)+u'_n(5)\right] \end{bmatrix} = \begin{bmatrix} (1-\alpha\lambda_1)^n u'_0(1) \\ (1-\alpha\lambda_2)^n u'_0(2) \\ \frac{1}{\sqrt{2}}\left[(1-\alpha\lambda_3)^n u'_0(3)+(1-\alpha\lambda_5)^n u'_0(5)\right] \\ (1-\alpha\lambda_4)^n u'_0(4) \\ \frac{1}{\sqrt{2}}\left[-(1-\alpha\lambda_3)^n u'_0(3)+(1-\alpha\lambda_5)^n u'_0(5)\right] \end{bmatrix}$$

$$\mathbf{u}'_0 = \mathbf{Q}^T\mathbf{u}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\therefore \mathbf{u}_n = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{2}\left[(1-\alpha\lambda_3)^n + (1-\alpha\lambda_5)^n\right] \\ 0 \\ \frac{1}{2}\left[(1-\alpha\lambda_3)^n - (1-\alpha\lambda_5)^n\right] \end{bmatrix}$$

Note that since $\lambda_3 \neq \lambda_5$ then $u_n(5) \neq 0$ for finite *n*. This says that even though no error is initially present in this filter component, $u_n(5)$ is nonzero for finite *n*. As we have seen in earlier sections, this type of behavior can arise with the linear LMS filter but not with white inputs.

(iii) By following similar analysis to that for the linear filter (see Chapter 4), and without invoking any further assumptions on the input, we may extend the relationship $J_\infty = J_{\min} +$ excess MSE to the second-order filter. The expression for the excess MSE becomes

$$\text{excess MSE} \approx \sum_{j=0}^{L-1}\frac{\alpha_1}{2}J_{\min}\lambda_j^{(2)} + \sum_{k=0}^{L_2-1}\frac{\alpha_2}{2}J_{\min}\lambda_k^{(4)} \qquad (97)$$

where $L_2 = L(L+1)/2$. $\lambda_j^{(2)}$ and $\lambda_k^{(4)}$ are the eigenvalues of the matrix $\mathbf{R}_2$ and $\mathbf{R}_4$, respectively.
Equivalently we may write

$$J_\infty \approx J_{\min}\left(1+\frac{\alpha_1}{2}tr\{\mathbf{R}_2\}+\frac{\alpha_2}{2}tr\{\mathbf{R}_4\}\right) \qquad (98)$$

which is the direct extension of

$$J_\infty \approx J_{\min}\left(1+\frac{\alpha}{2}tr\{\mathbf{R}\}\right) \qquad (99)$$

for the conventional LMS.

III. Recursive Least-Squares Algorithms

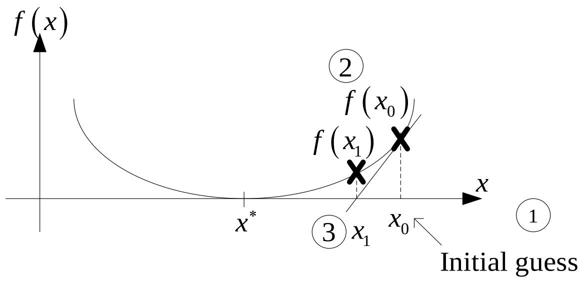
1) Algorithms Derived from Newton's Method

In Chapter 4, we first considered iterative solutions to the normal equations. As we indicated, the main weakness of the steepest descent philosophy in general, and the LMS algorithm in particular, are slow and generally nonuniform convergence.

The further disadvantage of LMS is that it attempts to optimize a stochastic criterion using a local, single point estimate of the gradient at each iterative update. An alternative approach would be to solve for the optimum filter at time *n*, using all of the data available up to that point.

(a) Newton's method provides a technique for iteratively obtaining the roots of a function *f*(*x*). The method is known to converge rapidly for a wide class of functions.

Consider a function having a single zero *f*(*x*) = 0 at $x = x^*$ as depicted in Fig. 5.7.



■ **Figure 5.7** Finding the zeros of a function via Newton's iteration.

The equation of this tangent is the derivative of the function at x_0 and is given simply by

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \quad \left(\because \frac{y}{x - x_1} = \frac{f(x_0)}{x_0 - x_1} \right) \tag{100}$$

$$\Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{101}$$

$$\Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{102}$$

which, as is illustrated by the example in the diagram, converges rapidly for 'reasonably' behaved functions. Of particular interest is the case where *f*(*x*) is linear, since in this case the tangent is identical to the function itself, and hence convergence to the zero takes place in a single iterative step.

(b) To apply this algorithm to the problem of least-squares minimization, we must find the zero of the gradient of the MSE rather than of the function itself. Since the MSE is quadratic, however, the gradient is linear, and consequently convergence takes place in a single step.

(c) Recall from Chapter 4,

$$J = E\left\{e^2(n)\right\} = E\left\{\left[d(n) - \mathbf{f}_n^T \mathbf{x}_n\right]^2\right\} = \sigma_d^2 - 2\mathbf{f}_n^T \mathbf{g} + \mathbf{f}_n^T \mathbf{R} \mathbf{f}_n$$

$$\nabla J = \frac{\partial J}{\partial \mathbf{f}_n} = 2\mathbf{R}\mathbf{f}_n - 2\mathbf{g} = 0 \tag{103}$$

$$\Rightarrow \mathbf{f}^* = \mathbf{R}^{-1}\mathbf{g}$$

J is quadratic and we anticipate that Newton's method should find the zero of the linear function ∇J in one step. For this multidimensional problem, we may demonstrate that the Newton algorithm is given by

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{1}{2}\mathbf{R}^{-1}\nabla J \quad \left(\because \frac{\partial}{\partial \mathbf{f}_n}\nabla J = 2\mathbf{R}\right) \tag{104}$$

To verify this result, we need only show that this update does indeed produce the result in one step. Substituting $\nabla J = 2(\mathbf{R}\mathbf{f}_n - \mathbf{g})$ for the gradient into the update (104), giving

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{1}{2}\mathbf{R}^{-1}\left(2\mathbf{R}\mathbf{f}_n - 2\mathbf{g}\right) = \mathbf{R}^{-1}\mathbf{g} = \mathbf{f}^* \tag{105}$$

so that (104) is indeed the Newton iteration.

(d) However, this method requires

$$\begin{cases} \mathbf{R}^{-1} \Rightarrow \text{the computation is very expensive} \\ \nabla J \Rightarrow \text{using the LMS gradient estimate, } \hat{\nabla} J = -2e(n)\mathbf{x}_n \end{cases}$$

With the introduction of the estimate for ∇J into the update, the $\mathbf{f}_n$ will certainly be degraded by noise, and thus the result will produce excess MSE. To allow a greater degree of control, a general constant α $(\alpha < 1)$ is introduced, and the update becomes

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha \mathbf{R}^{-1} e(n)\mathbf{x}_n \tag{106}$$

In general, convergence to $\mathbf{f}^*$ will no longer occur in a single step.

(e) Using the approach of Chapter 4, we may rewrite the update (106) as

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha\mathbf{R}^{-1}\left[d(n) - \mathbf{f}_n^T\mathbf{x}_n\right]\mathbf{x}_n = \left(\mathbf{I} - \alpha\mathbf{R}^{-1}\mathbf{x}_n\mathbf{x}_n^T\right)\mathbf{f}_n + \alpha\mathbf{R}^{-1}d(n)\mathbf{x}_n \tag{107}$$

Taking expectations and employing the usual independence assumption, we have

$$E\{\mathbf{f}_{n+1}\} = (1-\alpha)E\{\mathbf{f}_n\} + \alpha\mathbf{R}^{-1}\mathbf{g} \tag{108}$$

If $\alpha = 1$, then $E\{\mathbf{f}_{n+1}\} = \mathbf{R}^{-1}\mathbf{g} = \mathbf{f}^*$, that is, mean convergence in a single step. For $\alpha < 1$, we may employ the usual mean error vector

$$\mathbf{u}_n = E\{\mathbf{f}_n\} - \mathbf{f}^* \tag{109}$$

together with $\mathbf{f}^* = \mathbf{R}^{-1}\mathbf{g}$. This yields

$$\mathbf{u}_{n+1} = E\{\mathbf{f}_{n+1}\} - \mathbf{f}^* = (1-\alpha)\mathbf{u}_n \tag{110}$$

Hence, using repeated substitution

$$\mathbf{u}_n = (1-\alpha)^n \mathbf{u}_0 \tag{111}$$

From this equation, we may observe the following:

(i) The algorithm converges in the mean provided $|1-\alpha| < 1$. That is, if $0 < \alpha < 2$.

(ii) Convergence proceeds exponentially, at a rate determined by $(1-\alpha)$.

(iii) The convergence rate of each coefficient is identical, that is the convergence is independent of the eigenvalue spread of **R**.

This last point is a key advantage of this Newton algorithm and contrasts sharply with the LMS algorithm where, as we have seen, the convergence of different methods of the adaptive filter is dependent on the eigenvalue spread of **R**.

(f) The steady-state MSE for

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha\mathbf{R}^{-1}e(n)\mathbf{x}_n \tag{112}$$

is given by (for small α)

$$\text{excess MSE} \approx \frac{\alpha}{2} J_{\min} tr\{\mathbf{R}\} \tag{113}$$

This is an identical result for the excess MSE for the LMS algorithm. Thus, the advantages of the hybrid algorithm are obvious – it suffers no eigenvalue problems and has comparable misadjustment to the LMS for the same adaptation rate. However, we must keep in mind that these results relate to the idealized algorithm employing the true correlation matrix **R**.

At time *n*, we may estimate **R** by

$$\mathbf{R}_n = \sum_{l=0}^{n} \mathbf{x}_l \mathbf{x}_l^T, \quad n = 0,1,\ldots \tag{114}$$

The update (106) becomes

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \alpha\mathbf{R}_n^{-1}e(n)\mathbf{x}_n \tag{115}$$

To avoid the computation of the inverse $\mathbf{R}_n$, $\mathbf{R}_n$ itself may computed recursive using

$$\mathbf{R}_{n+1} = \mathbf{R}_n + \mathbf{x}_{n+1}\mathbf{x}_{n+1}^T \tag{116}$$

The update for $\mathbf{f}_n$ still has a very high computational requirement, however, not least because $\mathbf{R}_n$ is not Toeplitz and therefore we require on the order of L^3

operations for each inversion of the matrix. This computational complexity can be reduced by using the well-known matrix inversion lemma. That is,

$$\left(\mathbf{A}+\mathbf{u}\mathbf{u}^T\right)^{-1}=\mathbf{A}^{-1}-\frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{A}^{-1}}{1+\mathbf{u}^T\mathbf{A}^{-1}\mathbf{u}} \tag{117}$$

where **A** is a square (nonsingular) matrix, and **u** is an arbitrary (N×1) vector. Applying this to the update for the correlation matrix we have

$$\mathbf{R}_{n+1}^{-1}=\left(\mathbf{R}_n+\mathbf{x}_{n+1}\mathbf{x}_{n+1}^T\right)^{-1}=\mathbf{R}_n^{-1}-\frac{\mathbf{R}_n^{-1}\mathbf{x}_{n+1}\mathbf{x}_{n+1}^T\mathbf{R}_n^{-1}}{1+\mathbf{x}_{n+1}^T\mathbf{R}_n^{-1}\mathbf{x}_{n+1}} \tag{118}$$

In spite of the apparent complexity of this equation, given $\mathbf{R}_n^{-1}$ no matrix inversions are required because $1+\mathbf{x}_{n+1}^T\mathbf{R}_n^{-1}\mathbf{x}_{n+1}$ is a scalar. The computation is actually proportional to L^2 rather than L^3 as for the direct inversion.

Note: $\mathbf{R}_n$ itself is not calculated at any stage in the iteration. The solution begins with an initial value for $\mathbf{R}_{-1}^{-1}$ and simply updates the inverse using (118).

(g) The operation of the overall algorithm proceeds as follows:

(i) Initialize $\mathbf{f}_0$, $\mathbf{R}_{-1}^{-1}$

(ii) For n = 0, 1, …

$$\mathbf{R}_n^{-1}=\mathbf{R}_{n-1}^{-1}-\frac{\mathbf{R}_{n-1}^{-1}\mathbf{x}_n\mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}}{1+\mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}\mathbf{x}_n} \tag{119}$$

$$e(n)=d(n)-\mathbf{f}_n^T\mathbf{x}_n \tag{120}$$

$$\mathbf{f}_{n+1}=\mathbf{f}_n+\alpha\mathbf{R}_n^{-1}e(n)\mathbf{x}_n \tag{121}$$

Treichler *et al*. give two alternative procedures for the initial phase of computation for algorithm of this form [6]:

- $\mathbf{R}_n$ and $\mathbf{g}_n$ are built up using the initial input vectors until the rank of $\mathbf{R}_n$ is full. This requires at least L input vectors. $\mathbf{R}_n$ is then inverted.
- $\mathbf{R}_{-1}^{-1}$ is chosen as $\sigma^2\mathbf{I}$ where σ^2 is a small constant.

Note:

1. The first approach requires a one-off $\mathcal{O}(L^3)$ calculation of the inverse, and implies a delay of at least L data points before any updates are performed.
2. The second approach is simple and provides immediate adaptation but is obviously inaccurate initially in most cases. The use of a large constant for **R** is consistent with our uncertainty about the true correlation matrix. In practice, this method is most often employed.

2) Recursive Least-Squares (RLS)

(a) Derivation of the RLS algorithm

(i) This algorithm is derived from a quite different standpoint however; by considering the minimization of the deterministic measure:

$$J_n = \sum_{l=0}^{n} e^2(l) = \sum_{l=0}^{n} \left[d(l) - y(l)\right]^2 \tag{122}$$

where *d*(*n*) and *y*(*n*) are the desired signal and filter output, respectively. J_n is a recursive-in-time measure which changes at each point to reflect the arrival of a new data sample. Differentiating J_n with respect to $\mathbf{f}_n$ and equating to zero leads, as usual, to a set of normal equations:

$$\mathbf{R}_n\mathbf{f}_n = \mathbf{g}_n \tag{123}$$

with

$$R_n(i,j) = \sum_{l=0}^{n} x(l-i)x(l-j) \tag{124}$$

and

$$g(i) = \sum_{l=0}^{n} d(l)x(l-i) \tag{125}$$

Note that, as in (116), $\mathbf{R}_n = \mathbf{R}_{n-1} + \mathbf{x}_n\mathbf{x}_n^T$ and $\mathbf{g}_n = \mathbf{g}_{n-1} + d(n)\mathbf{x}_n$. So the optimal solution at the *n*th iteration is

$$\mathbf{f}_n = \mathbf{R}_n^{-1}\mathbf{g}_n \tag{126}$$

Given this information at time *n*, we seek

$$\mathbf{f}_{n+1} = \mathbf{R}_{n+1}^{-1}\mathbf{g}_{n+1} \tag{127}$$

at time index (*n*+1). With (119) and $\mathbf{g}_{n+1} = \mathbf{g}_n + d(n+1)\mathbf{x}_{n+1}$, we have

$$\mathbf{f}_{n+1} = \left[\mathbf{R}_n^{-1} - \frac{\mathbf{R}_n^{-1}\mathbf{x}_{n+1}\mathbf{x}_{n+1}^T\mathbf{R}_n^{-1}}{1+\mathbf{x}_{n+1}^T\mathbf{R}_n^{-1}\mathbf{x}_{n+1}}\right]\left[\mathbf{g}_n + d(n+1)\mathbf{x}_{n+1}\right] \tag{128}$$

Now, let $\mathbf{R}_n^{-1}\mathbf{x}_{n+1} = \mathbf{z}$, and using $\mathbf{f}_n = \mathbf{R}_n^{-1}\mathbf{g}_n$. We have

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \frac{\mathbf{z}\mathbf{x}_{n+1}^T\mathbf{f}_n}{1+\mathbf{x}_{n+1}^T\mathbf{z}} + d(n+1)\mathbf{z} - \frac{d(n+1)\mathbf{z}\mathbf{x}_{n+1}^T\mathbf{z}}{1+\mathbf{x}_{n+1}^T\mathbf{z}} \tag{129}$$

Also, let $\mathbf{x}_{n+1}^T\mathbf{z} = k$. Then, we may simplify (129) to

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \left[\frac{\mathbf{x}_{n+1}^T\mathbf{f}_n - d(n+1)}{1+k}\right]\mathbf{z} \tag{130}$$

or

$$\mathbf{f}_{n+1} = \mathbf{f}_n - \left[\frac{\mathbf{x}_{n+1}^T\mathbf{f}_n - d(n+1)}{1+k}\right]\mathbf{R}_n^{-1}\mathbf{x}_{n+1} \tag{131}$$

(ii) We denote by $e(n+1|n)$, the quantity

$$e(n+1|n) = d(n+1) - \mathbf{f}_n^T \mathbf{x}_{n+1} \tag{132}$$

This is often referred to as the *a priori error*, reflecting the fact that it is the error obtained using the old filter (that is prior to updating with the new data). The *a priori error* is actually the form used by the LMS – we calculate the error before updating the filter coefficients (although this isn't obvious from the LMS update – the filters are indexed differently.)

(iii) Finally, the RLS algorithm may be written

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \frac{e(n+1|n)}{1+\mathbf{x}_{n+1}^T \mathbf{R}_n^{-1} \mathbf{x}_{n+1}} \mathbf{R}_n^{-1} \mathbf{x}_{n+1} \tag{133}$$

The RLS algorithm in summary is thus:

- Initialize $\mathbf{f}_{-1}$, $\mathbf{R}_{-1}^{-1}$
- For n = 0, 1, …

$$e(n|n-1) = d(n) - \mathbf{f}_{n-1}^T \mathbf{x}_n \tag{134}$$

$$\alpha(n) = \frac{1}{1+\mathbf{x}_n^T \mathbf{R}_{n-1}^{-1} \mathbf{x}_n} \tag{135}$$

$$\mathbf{f}_n = \mathbf{f}_{n-1} + \alpha(n) e(n|n-1) \mathbf{R}_{n-1}^{-1} \mathbf{x}_n \tag{136}$$

$$\mathbf{R}_n^{-1} = \mathbf{R}_{n-1}^{-1} - \alpha(n) \mathbf{R}_{n-1}^{-1} \mathbf{x}_n \mathbf{x}_n^T \mathbf{R}_{n-1}^{-1} \tag{137}$$

Keeping in mind the fact that the LMS and Newton methods use the *a priori error*, and the scalar nature of the gain $\alpha(n)$, we see the similarity of the RLS algorithm and the Newtonian algorithms of equations (119)~(121). However, this algorithm may be expected to converge more quickly than that given in (119)~(121) due to the use of the step-size $\alpha(n)$.

(b) Exponentially weighted RLS

A more general form for the RLS filter is obtained by replacing (122) by the modified functional

$$\tilde{J}_n = \sum_{l=0}^{n} \lambda^{n-l} e^2(l) \tag{138}$$

where now a weighting factor λ is included where $0 < \lambda \le 1$.

Values of $\lambda < 1$ give more 'weight' (in $\tilde{J}_n$) to the most recent errors. This is useful in nonstationary data where changes make the inclusion of old data less appropriate.

(1) Using a similar procedure to that for the unweighted RLS, we obtain a similar set of normal equations with

$$\mathbf{R}_n\mathbf{f}_n = \mathbf{g}_n \tag{139}$$

but where now

$$\begin{aligned}\mathbf{R}_n &= \sum_{l=0}^{n}\lambda^{n-l}\mathbf{x}_l\mathbf{x}_l^T\\ \mathbf{g}_n &= \sum_{l=0}^{n}\lambda^{n-l}d(l)\mathbf{x}_l\end{aligned} \tag{140}$$

Once again, we may write $\mathbf{R}_n$ and $\mathbf{g}_n$ in recursive form

$$\begin{cases}\mathbf{R}_n = \sum_{l=0}^{n-1}\lambda^{n-l}\mathbf{x}_l\mathbf{x}_l^T + \mathbf{x}_n\mathbf{x}_n^T\\ \mathbf{g}_n = \sum_{l=0}^{n-1}\lambda^{n-l}d(l)\mathbf{x}_l + d(n)\mathbf{x}_n\end{cases} \tag{141}$$

or

$$\begin{cases}\mathbf{R}_n = \lambda\mathbf{R}_{n-1} + \mathbf{x}_n\mathbf{x}_n^T\\ \mathbf{g}_n = \lambda\mathbf{g}_{n-1} + d(n)\mathbf{x}_n\end{cases} \tag{142}$$

(2) Using a similar approach to that for the simple RLS, we may obtain an Exponentially Weighted RLS (EWRLS) algorithm as:

(i) Initialize $\mathbf{f}_{-1}$, $\mathbf{R}_{-1}^{-1}$

(ii) For n = 0, 1, …

$$e(n|n-1) = d(n) - \mathbf{f}_{n-1}^T\mathbf{x}_n \tag{143}$$

$$\alpha(n) = \frac{1}{\lambda + \mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}\mathbf{x}_n} \tag{144}$$

$$\mathbf{f}_n = \mathbf{f}_{n-1} + \alpha(n)e(n|n-1)\mathbf{R}_{n-1}^{-1}\mathbf{x}_n \tag{145}$$

$$\mathbf{R}_n^{-1} = \frac{1}{\lambda}\left[\mathbf{R}_{n-1}^{-1} - \alpha(n)\mathbf{R}_{n-1}^{-1}\mathbf{x}_n\mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}\right] \tag{146}$$

$$\left(\because \mathbf{R}_n^{-1} = [\lambda\mathbf{R}_{n-1} + \mathbf{x}_n\mathbf{x}_n^T]^{-1} = \lambda^{-1}\mathbf{R}_{n-1}^{-1} - \frac{\lambda^{-1}\mathbf{R}_{n-1}^{-1}\mathbf{x}_n\mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}\lambda^{-1}}{1+\lambda^{-1}\mathbf{x}_n^T\mathbf{R}_{n-1}^{-1}\mathbf{x}_n}\right)$$

For $\lambda = 1$, the algorithm reduces to the unweighted form. This choice for λ is obviously appropriate for stationary data. For other data, $0.95 < \lambda < 0.9995$ has been suggested. Values at the higher end of that range are more likely to prove satisfactory [7]. The lower values are more likely to result in instability for the algorithm.

(c) Notes on the RLS Algorithm

(i) Computational complexity

Computationally, RLS is considerably more expensive than simple LMS.

LMS: $\mathcal{O}(L)$ operations per update

($2L$+1) multiplications and $2L$ additions for the update and filtering combined.

RLS: $\mathcal{O}(L^2)$ operations per update

$(4L^2+4L)$ multiplications/divisions and $3L^2+L-1$ additions/ subtractions for the update and filtering combined.

(ii) Convergence and eigenvalue disparity

In principle, the rotation which occurs because of the pre-multiplication by $\mathbf{R}_n^{-1}$, removes any eigenvalue dependence from the result (see (106)~(111)). In practice, however, as with the Newton algorithm discussed above, we usually start with an initial guess for $\mathbf{R} = c^2\mathbf{I}$. Clearly, the inverse of this matrix applies no rotation and therefore does not reduce the eigenvalue disparity.

During convergence, if our estimate of the inverse correlation matrix tends to the true value, then the eigenvalue disparity disappears. In fact, starting with $\mathbf{R}_{-1} = c^2\mathbf{I}$ and recursively computing an estimate of $\mathbf{R}_n$ denoted $\mathbf{R}'_n$ leads to

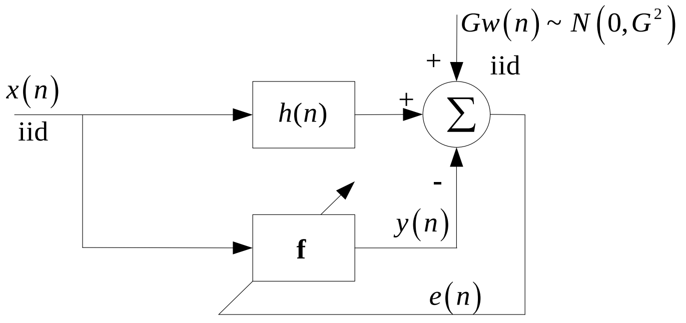
$$\mathbf{R}'_n = \sum_{l=0}^{n} \mathbf{x}_l \mathbf{x}_l^T + c^2\mathbf{I}, \quad n = 0,1,\ldots \tag{147}$$

$$E\{\mathbf{R}'_n\} = E\left\{\sum_{l=0}^{n} \mathbf{x}_l \mathbf{x}_l^T\right\} + c^2\mathbf{I} = E\{\mathbf{R}_n\} + c^2\mathbf{I} \neq E\{\mathbf{R}_n\} \tag{148}$$

Hence, the effect of this initialization is to bias the solution away from $\mathbf{f}_n = \mathbf{R}_n^{-1}\mathbf{g}_n$. However, we see that as n increases, the effect of this initial inaccuracy diminishes and $\mathbf{R}'_n$ is asymptotically unbiased.

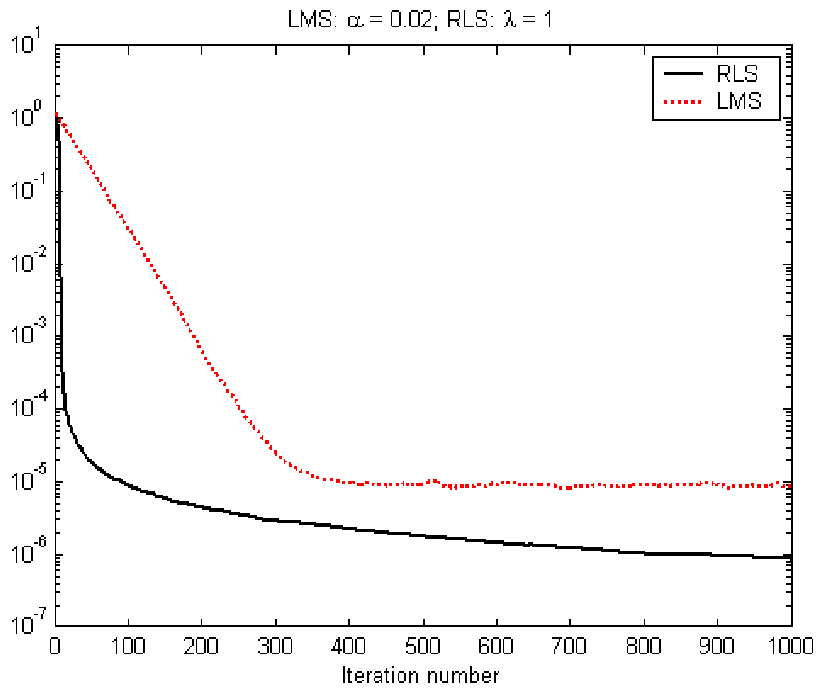
(iii) RLS vs. LMS

<u>*Example* 4</u>: Consider the system identification problem illustrated in Fig. 5.8. An iid sequence, $x(n)$, with zero-mean unit-variance is inputted to a simple two-point impulse response $h(0) = 1$ and $h(1) = 0.5$. The output is corrupted by measured noise $w(n)$ with gain G. $w(n)$ is assumed to be iid zero-mean Gaussian with unit variance. The adaptive filter is required to identify the unknown system using the input, and the noisy system output.



■ **Figure 5.8** Adaptive filter model of the unknown system $h(n)$.

Fig. 5.9 shows a comparison of the RLS and LMS algorithms. In this trial, G was selected to given an SNR of 40 dB. The diagram shows the norm of the coefficient errors $\mathbf{v}_k^T \mathbf{v}_k$ for k = 1, 2, …, 1000 where $\mathbf{v}_k = \mathbf{f}_k - \mathbf{f}^*$. The results shown were obtained by averaging over an 'ensemble' of 100 trials. These results were obtained using L = 8 and zero initial conditions for the filter coefficients for both algorithms. The adaptation constant for LMS was set as $\alpha = 0.02$ and the initial guess of the autocorrelation matrix was $\mathbf{R}_0 = 0.001\mathbf{I}$.



■ **Figure 5.9** System modeling using LMS and RLS for an iid input.

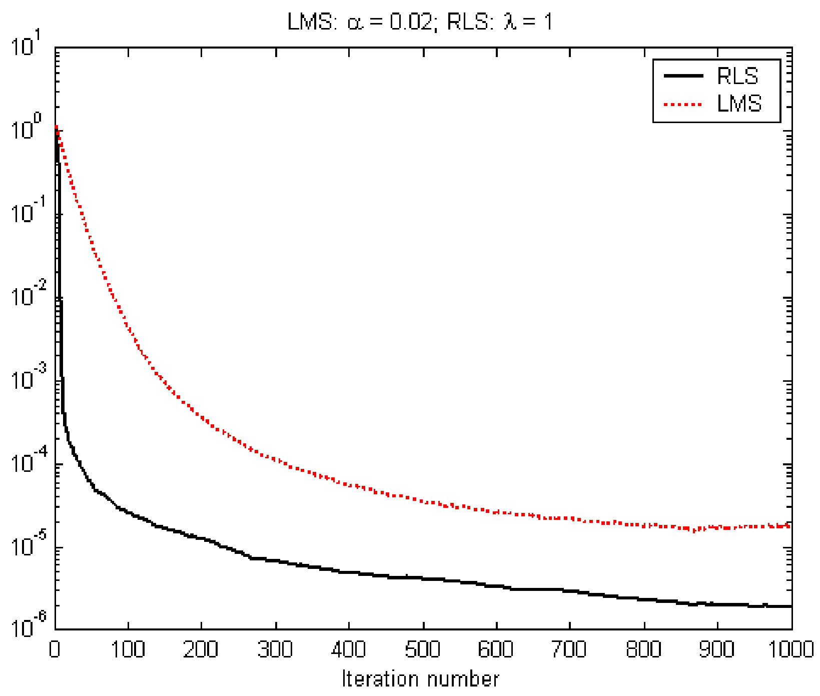
The plot shows clear superiority for RLS in terms of both faster convergence and lower steady-state MSE. This increased convergence speed was obtained in spite of the fact that the adaptation step-size used for LMS was the largest value found that gave stable results. Moreover, this superiority was obtained with white inputs. For a colored input, the performance difference between the two methods should be even more dramatic since RLS should be relatively unaffected by the eigenvalue disparity.

Fig. 5.10 shows a similar performance comparison but the input is an MA(1) process

$$x(n) = s(n) + 0.8s(n-1)$$

where $s(n)$ is assumed to be zero-mean Gaussian with unit variance.

Comparing Figs. 5.9 and 5.10 confirms that for the colored input the LMS convergence is considerably slowed, whereas that for RLS is virtually unchanged.



■ **Figure 5.10** System modeling using LMS and RLS for an MA(1) input.

Note:

1. Generally, RLS algorithms have been employed extensively in parameter estimation and system identification problems. In signal processing applications, RLS has been less widely accepted. In part, this is a reflection of the higher computation and implementation complexity of RLS algorithms compared to LMS. It's also due to a perception that RLS is in some sense less robust than is LMS.
2. For the higher computation and implementational complexity of RLS algorithms, it has been addressed by the development of Fast RLS algorithms [9]. However, the numerical sensitivity of these algorithms remains problematic [10], and the development of stable fast algorithms remains an area of active research.

(d) In many practical problems, the initial RLS estimates may fluctuate substantially, and the LMS algorithm may in fact offer smoother convergence towards $\mathbf{f}^*$, with no loss of speed.

(e) Additionally, the superiority of RLS over LMS is often lost with nonstationary data. Without exponential weighting, RLS suffers poor tracking ability due to the equal weighting that is applied to all previous data in the correlation estimates. Exponential weighting can help tracking, but it is often not clear how λ should be chosen. Additionally, as we indicated, $\lambda < 1$ tends to increase numerical problems in the algorithm.

(f) The major advantage of RLS over LMS lies in faster convergence and reduced sensitivity to eigenvalue disparity for stationary inputs. The RLS algorithm is less useful in low eigenvalue disparity situations, in cases where unmodeled noise is high, and in tracking nonstationary data.

Appendix [8]

Consider the nonnegative scalar function

$$P(n)=\mathbf{v}_n^T\mathbf{v}_n \tag{A1}$$

Under the reasonable assumption that the initial summed squared parameter error *P*(0) is finite, if

$$\Delta P(n)=P(n)-P(n-1) \tag{A2}$$

is nonpositive for all *n*, then $\Delta P(n)\to 0$ as $n\to\infty$.

This conclusion that a nonpositive $\Delta P(n)\to 0$ (and, consequently, that *P*(*n*) converges to a constant.) as $n\to\infty$ arises from

$$\infty > P(0)\ge \left|P(n)-P(0)\right| = \sum\nolimits_{i=1}^{n}\left|\Delta P(i)\right|\ge 0\,. \tag{A3}$$

From the update procedure of NLMS

$$\begin{aligned}P(n)&=\mathbf{v}_n^T\mathbf{v}_n\\&=\left(\mathbf{v}_{n-1}-\frac{\alpha\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{v}_{n-1}}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\right)^T\left(\mathbf{v}_{n-1}-\frac{\alpha\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{v}_{n-1}}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\right)\end{aligned} \tag{A4}$$

$$\begin{aligned}\Delta P(n)&=P(n)-P(n-1)\\&=-\frac{2\alpha\mathbf{v}_{n-1}^T\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{v}_{n-1}}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}+\frac{\alpha^2\mathbf{v}_{n-1}^T\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{v}_{n-1}}{\left(c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}\right)^2}\\&=-\frac{\alpha\left[d(n)-\hat{d}(n)\right]^2}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\left(2-\frac{\alpha\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\right)\\&=-\frac{\alpha\left[d(n)-\hat{d}(n)\right]^2}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\left(\frac{2c+(2-\alpha)\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\right)\end{aligned} \tag{A5}$$

Given $c > 0$ and $0<\alpha<2$, the right side of the above equation is nonpositive and thus $\Delta P(n)$ must converge to zero. Since the term in the braces is always positive,

$$\frac{\left[d(n)-\hat{d}(n)\right]^2}{c+\mathbf{x}_{n-1}^T\mathbf{x}_{n-1}}\to 0 \text{ as } n\to\infty\,. \tag{A6}$$

That is, $e(n)\to 0$ as $n\to\infty$ and $\mathbf{v}_n=\mathbf{f}_n-\mathbf{f}^*$ converges to a finite value.

References

[1] P. M. Clarkson, *Optimal and Adaptive Signal Processing*. Boca Raton, FL: CRC, 1993.

[2] E. Walach and B. Widrow, “The least mean fourth (LMF) adaptive algorithm and its family,” *IEEE Trans. Inform. Theory,* vol. IT-30, pp. 275-283, Mar. 1984.

[3] S. Haykin, *Adaptive Filter Theory*. 3rd Ed., Prentice-Hall, 1996.

[4] R. W. Harris, D. M. Chabries, and F. A. Bishop, “A variable step (VS) adaptive filter algorithm,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp.309-316, April 1986.

[5] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, New York, 1996.

[6] J. R. Treichler, C. R. Johnson Jr., and M. G. Larimore, *Theory and Design of Adaptive Filters*. John Wiley & Sons, New York, 1987.

[7] S. T. Alexander, *Adaptive Signal Processing Theory and Applications*. Springer-Verlag, 1986.

[8] C. R. Johnson Jr., *Lectures on Adaptive Parameter Estimation*. Prentice-Hall, 1988.

[9] J. M. Cioffi and T. Kailath, “Fast recursive least squares transversal filters for adaptive filtering,” *IEEE Trans*. *Acoust*., *Speech*, *Signal Processing*, vol. ASSP-32, pp. 304-337, 1984.

[10] C. F. N. Cowan, “Performance comparisons of finite linear adaptive filters,” *IEE Proceedings F*, vol. 134, pp. 211-216, 1987.